

Специальное издание

ИСПОЛЬЗОВАНИЕ

Macromedia®

Flash™ MX

ЕДИНСТВЕННАЯ НЕОБХОДИМАЯ КНИГА ПО FLASH MX

Специальное издание

**ИСПОЛЬЗОВАНИЕ
Macromedia®
Flash™ MX**

Special Edition

USING Macromedia® Flash™ MX

Michael Hurwicz, Laura McCabe, et al.

que®

A Division of Macmillan Computer Publishing, USA
201 W. 103rd Street
Indianapolis, Indiana 46290

Специальное издание

ИСПОЛЬЗОВАНИЕ Macromedia® Flash™ MX

Майкл Гурвиц, Лора Мак-Кейб



Москва • Санкт-Петербург • Киев
2003

ББК 32.973.26-018.2.75

Г95

УДК 681.3.07

Издательский дом "Вильяме"

Зав. редакцией *С.Н. Тригуб*

Руководитель проекта *В.В. Александров*

Перевод с английского и редакция *Ю.Н. Скорород*

По общим вопросам обращайтесь в Издательский дом "Вильяме" по адресу:
info@williamspublishing.com, <http://www.williamspublishing.com>

Гурвиц, Майкл, **Мак-Кейб**, Лора.

Г95 Использование Macromedia Flash MX. Специальное издание. : Пер. с англ. — М. :
Издательский дом "Вильяме", 2003. — 704 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0493-5 (рус.)

Данная книга представляет собой исчерпывающее и профессионально написанное руководство по Flash MX. Эта книга поможет вам в самом начале пути освоения Flash MX и останется ценным источником информации, даже когда вы станете настоящим мастером. На компакт-диске, прилагаемом к книге, содержатся файлы примеров, множество полезных программ, рассматриваемых в книге, а также Web-ссылки, посвященные данному вопросу. Использование Macromedia Flash MX — это именно та книга, которая будет полезна любому профессиональному Web-дизайнеру, разработчику и аниматору.

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Que Corporation.

Authorized translation from the English language edition published by Macmillan Computer Publishing Copyright © 2003.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Russian language edition published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2003.

ISBN 5-8459-0493-5 (рус.)
ISBN 0-7897-2762-5 (англ.)

© Издательский дом "Вильяме", 2003
© Que Corporation, 2003

ОГЛАВЛЕНИЕ

Введение	24
ЧАСТЬ I. СРЕДА И ИНСТРУМЕНТЫ FLASH	27
Глава 1. Что нового во Flash MX	29
Глава 2. Интерфейс Flash	39
Глава 3. Рисование во Flash	61
Глава 4. Использование растровых изображений во Flash	81
Глава 5. Работа с текстом	97
ЧАСТЬ II. АНИМАЦИЯ И ЗВУК	115
Глава 6. Символы, экземпляры и элементы библиотек	117
Глава 7. Анимация во Flash	137
Глава 8. Использование звука	157
Глава 9. Кнопки, меню и поля ввода данных	173
Глава 10. Интеграция видео	187
ЧАСТЬ III. РАСШИРЕНИЕ ИНТЕРАКТИВНОСТИ С ПОМОЩЬЮ ACTIONSCRIPT	197
Глава 11. Знакомство с ActionScript	199
Глава 12. Управление переменными, данными и типами данных	217
Глава 13. Использование операторов	245
Глава 14. Работа с данными: использование предложений	279
Глава 15. Объединение предложений в функции	295
Глава 16. Взаимодействие, события и установление последовательности	329
Глава 17. Демонстрация мощи видеоклипов	357
Глава 18. Процедура рисования с помощью ActionScript	395
ЧАСТЬ IV. РАСШИРЕННЫЕ СЦЕНАРИИ ACTIONSCRIPT	417
Глава 19. Использование встроенных базовых объектов	419
Глава 20. Использование встроенных объектов фильмов	467
Глава 21. Упаковка данных и функций с помощью пользовательских объектов	515
Глава 22. Компоненты	537
Глава 23. Использование обучающих взаимодействий	559
Глава 24. Коллективное использование ActionScript	577
Глава 25. Тестирование, отладка и устранение проблем	589

ЧАСТЬ V. РЕАЛИЗАЦИЯ ВНЕШНИХ СВЯЗЕЙ С ПОМОЩЬЮ FLASH	601
Глава 26. Локальная связь	603
Глава 27. Взаимосвязь с сервером	625
Глава 28. XML-данные	637
ЧАСТЬ VI. ВЫХОДНЫЕ ПАРАМЕТРЫ FLASH	653
Глава 29. Печать Flash-фильма	655
Глава 30. Оптимизация, публикация и экспортирование фильмов	669
ПРИЛОЖЕНИЕ А. Улучшение доступности Flash	688
Предметный указатель	691

СОДЕРЖАНИЕ

Об авторах	21
О соавторах	21
Посвящение	22
Благодарности Майкла	22
Благодарности Лоры	23
Введение	24
Что нового во Flash MX	24
Для кого предназначена эта книга	25
Как пользоваться книгой	25
Условные обозначения, принятые в книге	26
ЧАСТЬ I. СРЕДА И ИНСТРУМЕНТЫ FLASH	27
Глава 1. Что нового во Flash MX	29
Усовершенствование интерфейса	29
Группы панелей	29
Инспектор свойств	30
Новые инструменты и элементы меню	30
Инструмент Free Transform	30
Команды Break Apart и Distribute to Layers	30
Команда Snap to Pixels	31
Папки слоев	31
Шаблоны	32
Компоненты	32
Поддержка видео	32
Усовершенствование сценариев	34
Усовершенствованная объектная модель и модель событий	34
Маскирование по сценарию	35
Программное рисование	35
Загрузка внешних изображений и звука	35
Усовершенствованный редактор ActionScript	36
Совместимость с Flash 5	36
Доступ к содержимому	36
Возможные проблемы	37
Глава 2. Интерфейс Flash	39
Наборы панелей	39
Инспектор свойств	40
Компоновка панелей	41
Задание атрибутов документа	41
Временная шкала	42
Кадры	42
Ключевые кадры	42
Слой и папки слоев	43
Калькирование и редактирование нескольких кадров	44

Панель инструментов	45
Подраздел Tools	45
Подраздел View	49
Подраздел Colors	50
Клавиши быстрого доступа	50
Настройка клавиш быстрого доступа	51
Библиотека	52
Создание общих библиотек	52
Проводник фильма	53
Задание параметров Flash	53
Общие настройки	54
Параметры редактирования	55
Параметры буфера обмена	57
Настройка предупреждающих сообщений	57
Параметры редактора ActionScript	58
Возможные проблемы	58
Flash за работой: создание идеального интерфейса Flash MX	59
Глава 3. Рисование во Flash	61
Знакомство с векторной графикой	61
Использование инструментов рисования	62
Инструмент Line	63
Инструмент Pencil	63
Инструменты Oval и Rectangle	63
Инструмент Реп	63
Создание прямых отрезков	64
Создание криволинейных сегментов	64
Корректировка анкерных точек	65
Корректировка сегментов	66
Инструмент Brush	67
Редактирование и корректирование изображений	68
Использование инструментов Arrow и Subselection	68
Выпрямление и сглаживание	68
Оптимизация кривых	69
Работа со штрихами	69
Использование заливки	72
Заливка чистым цветом	72
Градиентное заполнение: линейное и радиальное	73
Растровое изображение	74
Стирание	75
Использование вспомогательных элементов компоновки	76
Сетки, направляющие и направляющие слои	76
Панель Align	77
Привязка	78
Создание масок	78
Возможные проблемы	79
Flash за работой: насколько большим получится изображение	79
Глава 4. Использование растровых изображений во Flash	81
Импортирование растровых изображений	81
Подготовка растровых изображений к импортированию	83
Трассировка растровых изображений	87

Оптимизация трассированных изображений	90
Разбивка растровых изображений	92
Сжатие растровых изображений	93
Возможные проблемы	94
Flash за работой: анимация растровых изображений	95
Глава 5. Работа с текстом	97
Инструмент Text	99
Инспектор свойств	100
Атрибуты символов	101
Выбор шрифта и кегля	101
Направление текста	102
Кернинг и межзнаковый интервал	102
Положение символа	103
Атрибуты абзаца	103
Выравнивание	104
Отступы, междустрочный интервал и поля	104
Параметры текста	105
Вводимый текст	105
Динамический текст	106
Параметры форматирования текста	107
Внедренные и встроенные шрифты	108
Редактирование и преобразование текста	110
Разбивка текста	110
Преобразование текста	111
Эффективные приемы работы с текстом	112
Возможные проблемы	113
Flash за работой: копирование текстовых полей	114
ЧАСТЬ II. АНИМАЦИЯ И ЗВУК	115
Глава 6. Символы, экземпляры и элементы библиотек	117
Понятие символов	117
Графические символы	118
Кнопки	118
Видеоклипы	118
Создание символов	119
Регистрация	120
Преобразование в символ	122
Редактирование символов	123
Редактирование на месте	123
Редактирование в новом окне	125
Режим редактирования символов	127
Редактирование точек регистрации	128
Экземпляры	129
Изменение свойств экземпляра	129
Замена символов	131
Имена символов	132
Библиотека	132
Возможные проблемы	134
Flash за работой: вложение символов	135

Глава 7. Анимация во Flash	137
Знакомство с анимацией	137
Кадры и ключевые кадры	138
Добавление, редактирование, удаление и копирование кадров	138
Управление ключевыми кадрами	140
Частота смены кадров	140
Создание промежуточных отображений	141
Промежуточное отображение движения	142
Направляющие движения	143
Регулировка	147
Калькирование	148
Создание промежуточных отображений с изменением формы	149
Указатели формы	151
Перемещение по временной шкале	151
Мультипликация во Flash	152
Создание эффекта панорамы	152
Циклы символов	153
Основные правила при создании анимации	153
Возможные проблемы	154
Flash за работой: упрощение анимации	155
Глава 8. Использование звука	157
Знакомство со звуком	157
Подготовка звука для Flash	158
Импортирование звука	160
Синхронизация звуков	161
Что первично? Звук против анимации	162
Параметры синхронизации	162
Редактирование звука	163
Управление звуком	165
Ключевые кадры	165
Связывание и объект Sound	166
Оптимизация звука	167
ADPCM	168
MP3-сжатие	169
Возможные проблемы	169
Flash за работой: отключение звука	170
Глава 9. Кнопки, меню и поля ввода данных	173
Интерактивность Flash	173
Кнопки	174
Простые кнопки	174
Сложные кнопки	178
Меню	180
Структурирование документов с помощью временной шкалы	180
Свойства Track as Button и Track as Menu Item	181
Использование компонентов	182
Добавление компонентов	182
Редактирование компонентов	184
Возможные проблемы	184
Flash за работой: команда Open as Library	185

Глава 10. Интеграция видео	187
Видео во Flash	187
Импортирование видео	188
Внедренное и связанное видео	189
Параметры сжатия кодека Sorenson Spark	189
Работа с импортированным видео	191
Управление воспроизведением видео	192
Возможные проблемы	195
Flash за работой: интерактивность видео	195
ЧАСТЬ III. РАСШИРЕНИЕ ИНТЕРАКТИВНОСТИ С ПОМОЩЬЮ ACTIONSCRIPT	197
Глава 11. Знакомство с ActionScript	199
Добавление интерактивности с помощью ActionScript	199
Доступ к панели Actions	200
Использование панели Actions	200
Выбор действий для кадра или объекта	200
Обычный и экспертный режимы	202
Скрытие и отображение списка действий	204
Знакомство с объектно-ориентированными языками	204
Три кита: удобочитаемость, повторное использование и достижимость	206
Удобочитаемость	206
Установка интервалов, отступов и применение прописных букв	206
Добавление комментариев к сценарию	207
Создание удобочитаемых имен	208
Упрощение, стандартизация и оптимизация кода	208
Повторное использование: расчленение кода на модули	208
Достижимость: систематизация и централизация кода	210
Систематизация данных и функций с помощью видеоклипов	211
Использование временной шкалы для систематизации кода	212
Поиск кода с помощью панели Movie Explorer	212
Возможные проблемы	213
Flash за работой: удобочитаемый, многократно используемый, достижимый код	214
Глава 12. Управление переменными, данными и типами данных	217
Работа с переменными	217
Назначение значений переменных	218
Где объявлять переменные	219
Когда не нужно централизовать код	219
Как систематизировать переменные в основной временной шкале	220
Использование видеоклипов для систематизации переменных	220
Использование объектов для систематизации переменных	221
Правила присвоения имен переменным (и другим элементам)	223
Группирование данных по типам	225
Девять типов данных ActionScript	225
Мощь чисел	226
Строки	227
Управление и принятие решений с помощью булевых данных	229
Моделирование области действия задачи с помощью объектов	229
Массив	230
Функция	232
Неявное и явное преобразование данных	233

Явное преобразование типов данных	236
Преобразование в число	236
Преобразование в строку	238
Неявное преобразование типов данных	238
Числа и строки против числовых и строковых литералов	239
Литералы объектов и массивов	240
Использование типов данных null и undefined	240
Возможные проблемы	241
Flash за работой: детектирование версии	242
Глава 13. Использование операторов	245
Операнды, выражения и предложения	245
Определение взаимосвязей простых данных с помощью выражений	248
Старшинство, ассоциативность и группирование операторов	249
Арифметические операторы	250
Специальное значение NaN ("Не число")	250
Положительное и отрицательное приращение	250
Сложение и вычитание	251
Полиморфный оператор +	252
Перегруженный оператор -	252
Умножение и деление	253
Деление по модулю (%)	253
Побитовые операторы	254
Побитовые логические операторы	255
Операторы со сдвигом бита	259
Присваивание и сложное присваивание	260
Операторы сравнения	262
Операторы равенства	262
Автоматическое преобразование типов данных для сравнения	263
Более не используемые операторы сравнения версии Rash 4	264
Логические (булевы) операторы	266
Условный оператор	268
Оператор "запятая"	270
Именованные операторы	271
Оператор new	271
Оператор typeof	271
Оператор instanceof	272
Оператор delete	272
Оператор void	272
Другие операторы	273
Доступ к свойствам объекта: оператор "точка"	273
Оператор "массив—элемент/объект—свойство"	273
Оператор "круглые скобки/обращение к функции"	274
Возможные проблемы	274
Flash за работой: применение побитовых операторов к игре в крестики-нолики	275
Глава 14. Работа с данными: использование предложений	279
Образование автономных предложений и управляющих блоков	279
Образование автономных предложений	280
Объединение предложений в управляющие блоки	281
Управление ходом выполнения программы	283
Принятие решений с помощью условных предложений и ключевого слова switch	284

Предложение if	284
Предложение if-else	284
Вложение условных предложений	285
Предложение switch/case	285
Предложение ifFrameLoaded	286
Повторение действий с помощью циклов	287
Предложение while	287
Предложение do-while	288
Предложение for	288
Предложение for-in	289
Возможные проблемы	290
Flash за работой: использование цикла for для создания списка	290
Глава 15. Объединение предложений в функции	295
Улучшение кода с помощью функций	295
Создание функций	296
Объявление функций	296
Использование функций без имен: функциональные литералы	297
Вызов функций	298
Использование ключевого слова var для создания локальных переменных внутри функций	298
Ключевое слово var не используется вне функций	299
Доступ к переменным временной шкалы внутри функций	299
Передача данных в функции с помощью аргументов	300
Передача аргументов по значению и по ссылке	300
Объект arguments	301
Извлечение результатов с помощью возвращаемых значений	305
Функции как классы: функции-конструкторы	306
Создание объектов с помощью оператора new	307
Свойство prototype	308
Назначение методов для класса	309
Применение наследования для создания повторно используемого кода	311
Использование оператора new для создания цепочек наследования	313
Функции как данные	316
Методы Function.apply() и Function.call()	316
Явный обзор данных	319
Ограничения жесткого кодирования	319
Явный обзор данных с помощью относительных обращений	319
Явный обзор данных с помощью ключевого слова this	320
Автоматический обзор данных	322
Возможные проблемы	325
Flash за работой: расчет факториалов — пример рекурсии	326
Глава 16. Взаимодействие, события и установление последовательности	329
Иницилирующее действие: временная шкала и обработчики событий	329
Создание сценариев на основе временной шкалы	330
Создание сценариев на основе событий	331
Расширение обработки событий во Flash MX	333
Системные события и события ввода пользователем	334
Регистрация приемников	336
Обзор данных и использование ключевого слова this с обработчиками событий	337
Явное обращение к обработчикам событий	338

Обработчики событий и фокусировка	339
Отключение и удаление обработчиков событий	340
События, связанные с кнопками	341
События, связанные с нажатием клавиш	343
События, связанные с мышью	345
События, связанные с видеоклипами	346
События, связанные с выбором	350
События, связанные со звуком	351
Событие <code>onResize</code> , связанное с рабочим полем	351
События, связанные с текстовыми полями	352
Установление последовательности действий с помощью порядка выполнения	353
Возможные проблемы	354
Flash за работой: универсальные процессоры событий Flash	354
Глава 17. Демонстрация мощи видеоклипов	357
Представление видеоклипов	357
Новые свойства видеоклипов во Flash MX	363
Свойство <code>enabled</code>	364
Свойство <code>focusEnabled</code>	364
Свойство <code>hitArea</code>	365
Свойство <code>tabChildren</code>	365
Свойство <code>tabEnabled</code>	366
Свойство <code>tabIndex</code>	366
Свойство <code>trackAsMenu</code>	366
Свойство <code>useHandCursor</code>	366
Создание и удаление видеоклипов	367
Создание видеоклипов	367
Удаление видеоклипов	369
Загрузка и выгрузка внешнего содержимого	369
Загрузка внешних SWF- или JPEG-файлов	369
Выгрузка внешнего SWF- или JPEG-файла	371
Управление визуальным порядком наложения видеоклипов	372
Уровни визуального порядка наложения	373
Использование исходных объектов для задания свойств новых видеоклипов	377
Использование метода <code>Object.registerClass()</code> для выделения нового видеоклипа в подкласс	378
Знакомство с методом <code>registerClass()</code>	379
Совместное использование методов <code>registerClass()</code> и <code>attachMovie()</code>	379
Использование метода <code>registerClass()</code> с видеоклипом, созданным вручную	380
Использование свойства <code>_proto_</code> вместо метода <code>registerClass()</code>	380
Как решить проблему передачи параметров при использовании метода <code>registerClassO</code>	381
Проверка перекрытия видеоклипов	382
Методы видеоклипов <code>localToGlobal()</code> и <code>globalToLocal()</code>	383
Уменьшение "пропусков" перекрывающихся областей при тестировании с помощью невидимых дочерних видеоклипов	384
Создание иллюзии идеального совпадения	384
Выполнение нескольких проверок на наличие совпадений в каждом кадре	385
Повторный вызов функции с помощью глобальной функции <code>setInterval()</code>	385
Определение фактического интервала	386
Остановка и "сброс" повторяющегося вызова функции	386
Перетаскивание видеоклипов	387

Предложения <code>startDrag()</code> и <code>stopDrag()</code>	387
Параметры метода <code>startDrag()</code>	387
Свойство видеоклипа <code>_droptarget</code>	389
Динамические маски	389
Возможные проблемы	391
Flash за работой: игра "счастливый случай"	392
Глава 18. Процедура рисования с помощью ActionScript	395
Знакомство с методиками рисования во Flash MX	395
Использование объектов для отслеживания операций рисования	398
Рисование линий и кривых	399
Рисование линий	399
Рисование кривых	402
Оптимизация функций рисования кривых	405
Использование сплошной заливки	406
Работа с градиентной заливкой	407
Работа с тремя градиентными массивами	408
Матрица преобразования	409
Краткий формат матрицы преобразования	410
Использование традиционной матрицы преобразования 3x3	410
Понимание операций с матрицей 3x3	412
Возможные проблемы	414
Flash за работой: рисование круга, состоящего из восьми сегментов	415
Определение координаты x контрольной точки	415
Определение координаты y контрольной точки	415
ЧАСТЬ IV. РАСШИРЕННЫЕ СЦЕНАРИИ ACTIONSCRIPT	417
Глава 19. Использование встроенных базовых объектов	419
Знакомство со встроенными объектами	419
Базовые объекты	420
Класс <code>Array</code>	420
Интерфейсные классы: <code>Boolean</code> , <code>Number</code> и <code>String</code>	430
Класс <code>Date</code>	436
Класс <code>Function</code>	441
Класс <code>Object</code>	441
Объект <code>Math</code>	452
Возможные проблемы	464
Flash за работой: удивительные массивы	465
Глава 20. Использование встроенных объектов фильмов	467
Объекты и классы, связанные с фильмами	467
Объект <code>Accessibility</code>	469
Класс <code>Button</code>	469
Объект <code>System.capabilities</code>	470
Класс <code>Color</code>	470
Объект <code>Key</code>	473
Объект <code>Mouse</code>	476
Объект <code>Selection</code>	476
Класс <code>Sound</code>	477
Объект <code>Stage</code>	481
Объект <code>System</code>	484
Классы <code>TextField</code> и <code>TextFormat</code>	484

Ценные мультимедийные объекты: Video, Microphone, Camera	495
Сетевые соединения в режиме реального времени: объекты NetConnection и NetStream	501
Класс LocalConnection	507
Возможные проблемы	511
Flash за работой: полная автоматизация игры в крестики-нолики с помощью класса LocalConnection	512
Глава 21. Упаковка данных и функций с помощью пользовательских объектов	515
Пользовательские объекты	515
Размещение локальных свойств в иерархии наследования	516
“Потеря” объекта prototype существующего класса при использовании оператора new	517
Подмена наследуемых свойств	520
Подмена свойств в отдельных экземплярах	522
Отсутствие подмены в глобальных объектах	522
Переписывание наследуемых свойств	522
Доступ к надклассу с помощью оператора super	523
Доступ к методу надкласса с помощью оператора super	523
Получение локальных свойств надкласса в экземплярах подкласса	524
Свойство constructor	524
Создание объектов-потомков с помощью свойства constructor	525
Использование свойства constructor для преобразования элементарных типов данных в объекты	526
Использование функции-конструктора для “безопасного” хранения	527
Создание иерархии классов с помощью свойства _proto_	528
Использование свойства constructor для исполнения функций-конструкторов	529
Создание свойства _constructor_ в прототипе подкласса	531
Разработка надклассов и подклассов: связи типа “являться” и “иметь”	532
Связь типа “являться”	532
Связь типа “иметь”	532
Прояснение сложных случаев	532
Детализация иерархий классов: абстракция и конкретизация	532
Возможные проблемы	534
Глава 22. Компоненты	537
Использование встроенных компонентов	537
Компонент ComboBox	538
Компонент ListBox	539
Компонент CheckBox	540
Переключатели	541
Нажимные кнопки	542
Прокручивающееся текстовое поле	542
Полоса прокрутки	543
Использование панели Library с компонентами	544
Редактирование оболочек	545
Создание новых компонентов	545
Начало создания собственного компонента	545
Определение класса прямоугольника	545
Определение свойств компонентов	548
Добавление собственной пиктограммы	550
Задание идентификатора связи	550
Тестирование	550

Установка компонента на панели Components	551
Создание пользовательского интерфейса	553
Создание фильма текущего просмотра	556
Возможные проблемы	558
Глава 23. Использование обучающих взаимодействий	559
Взаимосвязи и шаблоны	559
Использование обучающих взаимодействий	560
Добавление параметров	562
Добавление взаимодействий	563
Добавление взаимодействия на временную шкалу	566
Использование перетаскивания	567
Конфигурирование обучающего взаимодействия Drag and Drop	567
Добавление объектов перетаскивания и целевых элементов	568
Удаление объектов перетаскивания	568
Заполнение бланка	569
Конфигурирование взаимодействия Fill in the Blank	569
Активные объекты	570
Конфигурирование обучающего взаимодействия Hot Object	570
Добавление и удаление вариантов активных объектов	570
Активные области	570
Множественный выбор	571
Добавление вариантов множественного выбора	571
Истина или ложь	571
Конфигурирование обучающего взаимодействия True or False	571
Обратная связь, отслеживание знаний и навигация	572
Задание параметров обратной связи	572
Задание параметра отслеживания знаний	572
Основная структура сценариев и компонентов обучающих взаимодействий	573
Возможные проблемы	574
Flash за работой: использование системы LMS	574
Подготовка данных при работе с системой, совместимой с AICC	575
Подготовка данных при работе с системой, совместимой со SCORM	576
Глава 24. Коллективное использование ActionScript	577
Библиотеки коллективного использования	577
Сеансовые элементы коллективного использования	578
Элементы коллективного использования, созданные на этапе разработки	579
Удаленный доступ во Flash	580
Java-классы и удаленный доступ во Flash	581
Классы ActionScript и удаленный доступ во Flash	582
Выполнение служебных обращений к серверу из Flash-фильма	582
Включение внешнего кода ActionScript	583
Возможные проблемы	584
Flash за работой: функция LocalConnection	585
Создание локальной связи	585
Отправление сообщений между доменами	587
Глава 25. Тестирование, отладка и устранение проблем	589
Как избежать наиболее распространенных проблем	589
Многоплатформенное тестирование	591
Эффективные процедуры тестирования	591
Оптимизация	592

Тестирование производительности загрузки	592
Поиск и устранение неполадок в коде ActionScript	593
Использование отладчика	594
Контрольные точки и прохождение по ним	594
Окно Output	596
Средства дистанционной отладки	598
Активизация отладчика удаленным образом	599
До сих пор остались проблемы?	599
ЧАСТЬ V. РЕАЛИЗАЦИЯ ВНЕШНИХ СВЯЗЕЙ С ПОМОЩЬЮ FLASH	601
Глава 26. Локальная связь	603
Управление браузером	603
Использование команды getURL	603
Целевые окна и фреймы	604
Создание новых окон браузера	605
Использование специальных целевых объектов	605
Вызов функций JavaScript	605
Проблемы, связанные с JavaScript	606
Изменение дескриптора OBJECT/EMBED	606
Передача сообщений из Flash в браузер	608
Передача сообщений из браузера во Flash	609
Альтернативные методики	612
Задание вопросов с помощью JavaScript	612
Создание пользовательских окон	613
Управление проектором	614
Модификация окна	614
Скрытие контекстного меню	615
Другие команды	615
Взаимосвязь с программой Director	615
Взаимосвязь Flash–Director	616
Взаимосвязь Director–Flash	617
Команды Lingo	619
Хранение информации локально	620
Взаимосвязь Flash-фильмов	622
Возможные проблемы	622
Глава 27. Взаимосвязь с сервером	625
Загрузка текстовых данных	625
Загрузка данных с помощью объекта LoadVars	625
Использование объекта LoadVars	626
Старый способ загрузки переменных	627
Внедрение данных на HTML-страницу	628
Загрузка динамических данных	630
Отправление данных	631
Возможные проблемы	633
Flash за работой: отправление электронной почты из Flash	634
Глава 28. XML-данные	637
Знакомство с XML	637
Основы XML	638
Использование XML во Flash	639
Синтаксический анализ XML-данных	639

Преобразование текста в XML	640
Исследование XML-данных	640
Облегчение обращения к XML	642
Обращение к атрибутам	642
Создание XML-данных	643
Создание и модификация атрибутов	644
Импортирование XML-документов	644
Отправление XML-документов на сервер	646
XML-сокеты	646
Возможные проблемы	647
Flash за работой: простое XML-приложение	647
ЧАСТЬ VI. ВЫХОДНЫЕ ПАРАМЕТРЫ FLASH	653
Глава 29. Печать Flash-фильма	655
Печать содержимого Flash	655
Подготовка файлов к печати	656
Назначение меток кадров	656
Отключение функции печати	657
Изменение распечатываемого фоновых цвета	658
Определение области печати	659
Создание кнопки Print	664
Возможные проблемы	667
Flash за работой: невидимые чернила	668
Глава 30. Оптимизация, публикация и экспортирование фильмов	669
Тестирование фильмов	669
Использование модуля Bandwidth Profiler	669
Использование отчета о размерах для оптимизации файлов	671
Оптимизация фильмов	671
Задание параметров публикации	673
Формат Flash	674
Формат HTML	676
Формат GIF	677
Формат JPEG	679
Формат PNG	680
Формат QuickTime	682
Экспортирование различных форматов файлов	684
Возможные проблемы	686
Flash за работой: использование именованных анкером	686
ПРИЛОЖЕНИЕ А. Улучшение доступности Flash	688
Правила доступности	688
Предметный указатель	691

ОБ АВТОРАХ

Майкл Гурвиц (Michael Hurwicz) — сотрудник компании Late Night Design — работает с методиками Flash, представляющими собой точку пересечения искусства, дизайна и технологии. С 1985 года он написал ряд статей, которые посвящены компьютерным технологиям. Майкл является президентом некоммерческой организации Irthlingz, занимающейся природоохранными вопросами. Познакомиться с деятельностью Майкла можно, обратившись к нему по электронной почте (michael@hurwicz.com) или посетив Web-узлы <http://www.latenightdesign.com>, <http://www.hurwicz.com> и <http://www.irthlingz.com>.

Лора Мак-Кейб (Laura McCabe) — независимый дизайнер. В настоящее время она проживает в Балтиморе, штат Мериленд, США. У Лоры разносторонние интересы: она прослушала базовый курс по философии, имеет образование в области прикладного искусства и дизайна и, в конце концов, освоила Flash. В течение шести лет она работала в области Internet-технологий и сотрудничала со многими клиентами, совершенствуя свои познания в области дизайна, разработки, информационной архитектуры и создания Web-продуктов. С ее разработками в области Flash можно познакомиться на коллективном узле Poems That Go, а с персональными работами — на узле <http://www.stolenglance.com>. В свободное время Лора занимается фотографией, редактированием, пишет статьи, время от времени катается по морю на байдарке и без усталости читает самые разные книги.

О СОАВТОРАХ

Лон Коули (Lon Coley) (LonColey@ariadne-webdesign.co.uk) — профессионал в сфере информационных технологий, специализирующийся в области офисных и Internet-приложений. На Web-узле ее компании (<http://www.ariadne-webdesign.co.uk>) размещено гораздо больше информации, чем можно было бы поместить в эту книгу. Опыт и наработки в данной области позволяют Лон, помимо разработки узлов для клиентов, выступать в роли консультанта и специалиста по устранению проблем, сотрудничая с крупными и мелкими компаниями, стремящимися усилить свое присутствие в Internet или имеющими узлы, которые, по их мнению, не в полной мере охватывают все необходимые возможности. Свято веря в то, что любой человек, обладая соответствующими средствами и навыками, может создать Web-узел, Лон часто сотрудничает с компаниями, которые желают разработать свои Web-узлы и в процессе внедрения новых технологий нуждаются в совете профессионала. Будучи опытным специалистом в области обучения, Лон разрабатывает и пишет обучающие материалы по бизнесу и образованию. Они охватывают весь спектр ее профессиональных знаний и всегда подготовлены так, чтобы в полной мере удовлетворить все возможные потребности клиентов.

Гэри Розенцвейг (Gary Rosenzweig) является ведущим специалистом, основателем и владельцем компании CleverMedia, расположенной в Денвере, занимающейся разработкой компьютерных игр и мультимедиа. Ему принадлежит авторство названий пакетов Macromedia Director и Macromedia Flash. Свою компанию CleverMedia Гэри основал несколько лет назад; сегодня ей принадлежат четыре крупнейших Web-узла компьютерных игр, созданных на основе технологий Shockwave и Flash. Гэри живет в Денвере, штат Колорадо, с женой Дебби, кошкой Люси и собакой Наташей. Помимо компьютеров и Internet он увлекается кемпингом, кинематографом и научной фантастикой.

ПОСВЯЩЕНИЕ

Маме и папе, которые всегда со мной.

Майкл Гурвиц

*Эту книгу я посвящаю маме и папе за их
постоянное воодушевление и поддержку.*

Лора Мак-Кейб

БЛАГОДАРНОСТИ МАЙКЛА

Огромное спасибо всем, кто принимал участие в создании этой книги, в частности Кейт Смолл (Kate Small), Сьюзен Хоббс (Susan Hobbs), Кэрол Боуэрс (Carol Bowers), Чаку Хатчинсону (Chuck Hutchinson) и Линн Баус (Lynn Baus) за редактирование текста, тщательный поиск и исправление моих ошибок, промахов и упущений. Чак является непревзойденным специалистом по редактированию и выискиванию ошибок.

Моему самому замечательному агенту Дэвиду Фугейту (David Fugate) из компании Waterside Productions за его бесценную работу по созданию и упрочению связей с Квебеком.

Ральфу Бокльбергу (Ralf Bokelberg) (<http://www.QLOD.com>), создателю фильма `охо.fl` (глава 13 "Использование операторов"), который стал основой роликов `tictactoe_lc.fl` и `player_lc.fl`, описанных в главе 20 "Использование встроенных объектов фильмов".

Адаму Холден-Баху (Adam Holden-Bache) и Марку Льюису (Mark Lewis), соответственно президенту и руководителю технического отдела компании Mass Transmit (<http://www.mass-transmit.com>), за ролик `dropdownmenu.fl`, описанный в главе 14 "Работа с данными: использование предложений".

Робину Дебреуилу (Robin Debreuil) (<http://www.debrteuil.com>) за фильм `MovieClass.fl` (глава 16 "Взаимодействие, события и установление последовательности"), являющийся аналогом "двойной панели" (глава 15 "Объединение предложений в функции"), и за то, что он помог мне разобраться в объектно-ориентированном программировании.

Гэри Гросмену (Gary Grossman), ведущему специалисту коллектива разработчиков Macromedia Flash, за функцию `makeHandler()`, описанную в главе 15 "Объединение предложений в функции".

Хелен Триоло (Helen Triolo) (<http://i-Technica.com>) за ролики `dynamic-ButtonMovie.fl` (глава 16 "Взаимодействие, события и установление последовательности") и `trigdemo.fl` (глава 19 "Использование встроенных базовых объектов").

Питеру Холлу (Peter Hall) (<http://peterjoel.com>) за ролики `scratch.fl` (глава 17 "Демонстрация мощи видеоклипов") и `saveDrawing.fl` (глава 18 "Процедура рисования с помощью ActionScript").

Энди Холлу (Andy Hall) (ahall@panache.co.uk) за ролик `spacelisten.fl` (глава 17 "Демонстрация мощи видеоклипов") и функцию `curveTo()` (глава 18 "Процедура рисования с помощью ActionScript").

Рику Иврингу (Ric Ewing) (rewing@riverdeep.net) за ролики `drawmethods.fl` и `4SegSircle.fl`, описанные в главе 18 "Процедура рисования с помощью ActionScript", а также за общую поддержку и помощь в работе над прикладным программным интерфейсом рисования.

Киту Питерсу (Keith Peters) (kp@bit-101.com, www.bit-101.com) за ролики `3Dcube.fl` и `test5.fl`, описанные в главе 18 "Процедура рисования с помощью ActionScript".

Милли Маруани (Millie Maruani) (millie@noos.fr, <http://millie.free.fr>) за ролики `api_flower.fl` и `api_cube.fl`, описанные в главе 18 "Процедура рисования с помощью ActionScript".

Фотиосу Бассайанису (Fotios Bassayannis) (<http://fotios.cc>) за великолепный ролик `Arrays.fl`, описанный в разделе "Flash за работой" главы 19 "Использование встроенных базовых объектов".

Роберту Пеннеру (Robert Penner) (robertpenner@yahoo.com, www.robert-penner.com) за функцию `superCon()` (аналог `mySuper()`), описанную в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

БЛАГОДАРНОСТИ ЛОРЫ

Огромное спасибо всем, кто помогал в работе над моей частью книги: Трейси Келли (Tracy Kelly) и Крисси Рей (Chrissy Rey), Сигрид Трампи (Sigrid Trumpy) за ролик, включенный в главу 10 "Интеграция видео", сообществу Flash за их постоянное воодушевление и поддержку, Франу (Fran) и Бобу Книсли (Bob Knisley) за их поддержку и создание надлежащих условий для работы, маме и Шарлотте МакЭнхилл (Charlotte McEnhill) за то, что они всегда были рядом, Горацио (Horatio) из компании MacWarehouse за потрясающее обслуживание при сбоях жесткого диска моего компьютера. Отдельная благодарность Кейт Смолл, Линн Баус и всей группе из Квебека, в которую входят также Сьюзен Хоббс, Кэрол Боуэрс и Чак Хатчинсон.

ВВЕДЕНИЕ

Macromedia Flash MX— последняя версия средства разработки для надстройки Flash Player, которая в настоящее время является основной надстройкой данного рода и установлена в браузерах 98% пользователей Web.

Технология Flash развивается с 1996 года как инструмент создания интерактивной векторной анимации для Web. Основной "ударной силой" Flash является возможность создания векторных анимационных файлов с небольшим временем загрузки, которые обеспечивают при этом высокую степень интерактивности.

Помимо этого, Flash (особенно после появления Flash MX) является многофункциональным средством, посредством которого можно реализовать доступ к базам данных, поддержку XML, интеграцию видео и аудио, использовать предварительно встроенные шаблоны, процедуру перетаскивания, получать доступ к серверам приложений и шлюзам, работающим в режиме реального времени. Все эти операции во Flash выполняются под управлением языка сценариев **ActionScript**, похожего на язык JavaScript.

Большинство коллективов, работающих с Flash, в настоящее время делятся на две группы: одни работают непосредственно с ActionScript, а другие — с графическим **содержимым**. Доступ к базам данных и работа с динамическим **содержимым** является отдельным направлением; точно так же, как и имеется отдельная "ветвь" технологии Flash для персональных цифровых ассистентов (PDA). В настоящее время Flash превратилась в столь широкую и разнообразную среду, что редко можно найти индивидуума, равно хорошо знакомого со всеми ее возможностями.

В настоящее время компания Macromedia предлагает целое семейство продуктов **версии MX**, которые группируются в пакет Macromedia Studio MX. В его состав входят хорошо известные обновленные версии продуктов, такие как Dreamweaver MX, Fireworks MX и ColdFusion MX, а также новая разработка для серверов потокового аудио и видео — Flash Communication Server. Macromedia стремится создать как инструментальные средства, так и платформы, предназначенные для создания следующего поколения мультимедийных Web-приложений.

Что нового во FLASH MX

Помимо простого "расширения", во Flash MX увеличена "глубина" программы. Это достигнуто с помощью множества функций, усовершенствующих и без того большие возможности программы. В данную версию внесены следующие важные усовершенствования.

- Папки, предназначенные для систематизации слоев временной шкалы.
- Новые элементы управления Stage, облегчающие редактирование символов.
- Средства коллективного использования библиотек.
- Улучшенная функция Color Mixer (Цветовой микшер).
- Редактирование на уровне пикселей.
- Команда "распределения в слои", позволяющая автоматически разнести любое число выделенных объектов по своим собственным слоям.
- Динамические маски.

■ Улучшенные элементы управления звуком.

- Инструмент Free Transform (Свободное преобразование), позволяющий выполнять несколько операций преобразования, таких как перемещение, вращение, масштабирование, наклон и искажение *огibaющей* без смены инструментальных средств.

Графический пользовательский интерфейс программы также претерпел существенные изменения. В данной версии появились сворачиваемые, плавающие панели, экономящие пространство экрана, а также новые панели инспекторов свойств, благодаря которым исчезает необходимость использования многих других диалоговых окон и панелей.

В язык **ActionScript** были внесены следующие нововведения.

- Возможность динамически загружать файлы JPEG и MP3.

Я Анкерные точки, позволяющие использовать кнопки Forward (Вперед) и Back (Назад) своих браузеров для перехода от анкера к анкеру.

- Функция Code Hints (Советы по использованию кода), позволяющая в нужный момент просматривать синтаксис команд.

И Повторное использование компонентов интерфейса перетаскивания.

а Просмотр текущих компонентов.

В Усовершенствованная функция отладки.

- Более полная модель объектов и событий.

И Расширенная поддержка текстовых полей и форматирования текста.

В Прикладной программный интерфейс рисования, расширяющий возможности инструментов рисования программы Flash.

■ Ускоренное выполнение функций XML.

ДЛЯ КОГО ПРЕДНАЗНАЧЕНА ЭТА КНИГА

С самого начала мы предполагаем, что читатель совершенно незнаком с Flash. В книге дается полное описание принципов и технологий, исходя из посылки, что читатель вообще ничего не знал о Flash до того, как открыл эту книгу.

В то же время мы предполагаем, что читатель намеревается стать опытным и профессиональным пользователем Flash, поэтому детально описываем темы средней и высокой сложности.

Начните читать книгу с самого начала и ничего не пропускайте, даже самых мельчайших подробностей, описываемых при рассмотрении сложных тем. В главах часто отсутствуют пошаговые инструкции, описывающие выполнение основных задач. Предполагается, что это побудит читателя самостоятельно работать с программой.

КАК ПОЛЬЗОВАТЬСЯ КНИГОЙ

Книгу можно читать от корки до корки или пользоваться ею от случая к случаю. Однако материал, представленный в последних главах, предполагает, что вы уже познакомились с предыдущими главами. Новичкам будет полезно немного попрактиковаться в работе с программой перед тем, как приступить к чтению книги. Скорее всего, вы будете использовать данную книгу как справочное пособие, пытаясь с ее помощью решать проблемы по мере их возникновения. С другой стороны, в программу Flash MX включено множество новых функций, с которыми могут быть незнакомы даже опытные пользователи. Если вы относитесь именно к этой категории читателей, пролистайте книгу и обратите внимание на абзацы, помеченные пиктограммой.



Если в книге приводятся примеры файлов, записанных на компакт-диске, который прилагается к этой книге, обязательно поработайте с ними и попробуйте их модифицировать. Такая практика окажет вам неоценимую услугу, и, может быть, вы будете использовать эти файлы для своих собственных проектов.

Кроме того, обязательно посетите многочисленные Web-ресурсы для разработчиков Flash. В качестве стартовой площадки можно воспользоваться Web-узлом автора этой книги, Майкла Гурвица (<http://www.flashhoop.com>). Кроме того, в книге приводятся несколько тематических списков рассылки, которые позволят получить квалифицированную помощь по всем вопросам, касающимся Flash.

Книга разделена на шесть частей.

В частях I (главы 1–5) и II (главы 6–10) основное внимание уделено вопросам использования графического пользовательского интерфейса среды разработки, а также описанию ориентированных на дизайн функций, не требующих или минимально использующих язык **ActionScript**.

Части III (главы 11–18), IV (главы 19–25) и V (главы 26–28) представляют собой полное руководство по использованию **ActionScript**. В части III подробно описывается язык **ActionScript**, начиная с самых элементарных принципов и заканчивая функциями, событиями и прикладным программным интерфейсом рисования. Часть IV представляет собой справочное руководство по работе со встроенными и пользовательскими объектами и компонентами. Здесь же описаны взаимодействия, принципы коллективного использования библиотек и средства отладки Flash. В главе 20 "Использование встроенных объектов фильмов" рассказывается о продукте **Flash Communication Server**, а также об объектах **Video**, **Microphone** и **Camera**. В части V описаны способы реализации связей как с клиентскими компьютерами, так и с серверами, в том числе и взаимодействие, основанное на **XML**.

В части VI рассматриваются выходные возможности программы, такие как печать, размещение и экспортирование Flash-файлов.

УСЛОВНЫЕ ОБОЗНАЧЕНИЯ, ПРИНЯТЫЕ В КНИГЕ

Для того чтобы отличить ключевые слова **ActionScript** от остального текста, используется моноширинный шрифт. К примеру, в книге будут встречаться следующие сочетания: метод `gotoAndPlay()`; указание компилятора `ttinitclip` и т.д.

В коде **ActionScript** слова, выделенные курсивом, представляют собой не сам код, а заполнители, которые должны быть заменены фактическим кодом. Например, в коде `Key.isDown(charCode)` слово *charCode* представляет собой заполнитель кода символа, подлежащего вставке. Код `Key.isDown(Key.Up)` является фактическим кодом **ActionScript**, в котором `Key.Up` представляет собой фактический код символа.

В книге часто используется принятое в **ActionScript** соглашение о том, что когда объект представляет собой новый экземпляр, созданный программистом, его имя начинается с символов *tu*. Например, `TextField` является встроенным классом текстовых полей, не созданных программистом, а `myTextField` — это новый экземпляр текстового поля.

В конце большинства глав имеются разделы "Возможные проблемы" и "Flash за работой". Если в процессе работы у вас возникли проблемы, не забудьте прочитать раздел "Возможные проблемы", в котором описываются пути решения наиболее распространенных из них.

Обратите особое внимание на примеры, приводимые в разделах "Flash за работой", поскольку зачастую они взяты из реальных проектов, а кроме того, в них приводится построчное объяснение принципов работы программы. Чтобы составить наиболее полное представление о том, что представляет собой та или иная функция, лучше всего посмотреть, как она работает в каком-либо реальном приложении.

СРЕДА и ИНСТРУМЕНТЫ FLASH

В ЭТОЙ ЧАСТИ...

Глава 1. Что нового во Flash MX

Глава 2. Интерфейс Flash

Глава 3. Рисование во Flash

Глава 4. Использование растровых изображений во Flash

Глава 5. Работа с текстом

Что нового во FLASH MX

В ЭТОЙ ГЛАВЕ...

Усовершенствование интерфейса	29
Компоненты	32
Поддержка видео	32
Усовершенствование сценариев	33
Совместимость с Flash 5	36
Доступ к содержимому	37
Возможные проблемы	37

УСОВЕРШЕНСТВОВАНИЕ ИНТЕРФЕЙСА

В программе Macromedia Flash MX сделан огромный шаг вперед по сравнению с Flash 5. В программу внесены усовершенствования и нововведения, такие как поддержка видео, компоненты пользовательского интерфейса, программируемое рисование и маски по сценарию.

Запустив программу, вы сразу же увидите, что изменились очень многие ее элементы. Внесены существенные изменения в интерфейс, благодаря чему он стал гораздо более удобным и интуитивно понятным.

ГРУППЫ ПАНЕЛЕЙ

Новые группы панелей версии MX предназначены одновременно для экономии пространства и организации среды проектирования. Новые панели можно сворачивать, делать плавающими или привязывать к определенной области экрана, что позволяет настроить их в соответствии с предпочтениями пользователя. Большинство панелей можно держать на рабочем столе свернутыми до тех пор, пока они не понадобятся. Даже панели Library (Библиотека) и Movie Explorer (Проводник фильма) можно использовать в сочетании с другими панелями.

ИНСПЕКТОР СВОЙСТВ

Наиболее эффективной новой функцией интерфейса является панель Property Inspector (Инспектор свойств). На этой панели, позаимствованной из программы Dreamweaver, сосредоточены основные свойства объекта (рис. 1.1). При выделении какого-либо объекта или инструмента открывается панель инспектора свойств, на которой приведены свойства и параметры данного объекта. Панель инспектора свойств заменила панели Stroke (Штрих), Fill (Заливка), Text (Текст), Paragraph (Абзац), Character (Символ), Instance (Экземпляр), Frame (Кадр), Effect (Эффект) и Sound (Звук), которые использовались в программе Flash 5. Например, при выделении объекта в окне Stage (Рабочее поле) откроется панель инспектора свойств с полями имени экземпляра, символьных поведений, координат, размеров, цветовых эффектов и подкачки символов. Работая с панелью инспектора свойств, нет необходимости открывать многочисленные окна и панели, требовавшиеся при работе с Flash 5. Естественно, теперь работа будет выполняться гораздо быстрее и эффективнее.

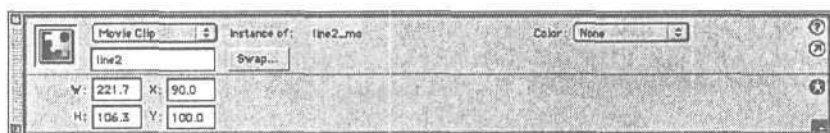


Рис. 1.1. Новая панель Property Inspector является контекстно-зависимой и отображает только свойства и параметры выделенного объекта или инструмента

НОВЫЕ ИНСТРУМЕНТЫ И ЭЛЕМЕНТЫ МЕНЮ

В программу Flash MX также внесены новые инструментальные средства и элементы меню, позволяющие усовершенствовать процесс дизайна и анимации. Новые средства дизайна включают инструмент Free Transform (Свободное преобразование) на панели Tools, команды Break Apart (Разделить) и Distribute to Layers (Распределить по слоям) в меню Modify (Изменить) и команду Snap to Pixels (Привязать к пикселям) в меню View (Вид). Папки слоя, выполняющие функции, аналогичные папкам библиотеки, позволяют систематизировать временную шкалу и сэкономить место на экране. Встроенные шаблоны являются руководством по созданию содержимого для рекламных баннеров, презентаций, анкет, слайд-шоу и даже графики для портативных устройств.

ИНСТРУМЕНТ FREE TRANSFORM

Новый инструмент Free Transform (Свободное преобразование) позволяет в большей степени управлять выделенным объектом. Модификаторы Edit Envelope (Редактирование огибающей) и Distort (Искажение) делают возможным реализацию сложных эффектов преобразования, в том числе наклон, проецирование и искажение.

КОМАНДЫ BREAK APART и DISTRIBUTE TO LAYERS

Во Flash 5 с помощью команды Break Apart (Разделить) выполнялось преобразование текстовых блоков в кривые. В программе Flash MX с помощью данной команды (выполняемой путем выбора пунктов меню **Modify** ⇒ **Break Apart**) стал доступен очень важный промежуточный шаг: разбивка текстового блока на отдельные буквы, как показано на рис. 1.2. Команду Break Apart можно использовать и для преобразования текстовых блоков в кривые, но для этого ее придется выполнить дважды. Сначала текстовый блок будет разбит на отдельные буквы, а затем выполняется преобразование в кривые.

Чтобы каждый из объектов можно было анимировать индивидуально, их необходимо расположить в отдельных слоях. Команда **Distribute to Layers** (Распределить по слоям) берет несколько объектов слоя и разносит их по отдельным слоям. Кроме того, новым слоям автоматически присваиваются имена в соответствии с именами разнесенных объектов. Выделите несколько объектов в окне **Stage** (Рабочее поле) и выполните команду **Modify**⇒**Distribute to Layers**. Скомбинировав команды **Break Apart** и **Distribute to Layers**, с помощью выполнения всего двух операций можно разбить текстовые блоки на буквы и разнести их по своим собственным слоям так, что их можно будет анимировать индивидуально. Данные команды являются одними из основных при создании анимации.

WE THE PEOPLE. IN ORDER TO FORM

Рис. 1.2. Теперь команда **Break Apart** выполняет разбивку текстового блока на отдельные буквы

КОМАНДА SNAP TO PIXELS

С помощью команды **Snap to Pixels** (Привязать к пикселям) можно управлять расположением объекта на уровне пикселей. Этой команды позиционирования уже давно с нетерпением ожидали все пользователи **Flash**. Головной болью дизайнеров, работающих с **Flash**, был способ, используя который **Flash** помещала объекты (и, в частности, текст) не только на целое, но и на дробное количество пикселей. Для обеспечения плавной визуализации векторной графики во **Flash** используются операции с плавающими точками, которые являются математическими средствами, но ориентированными на работу с пиксельной графикой. Тем не менее плавающие точки являются очень полезным средством и позволяют сглаживать тонкие линии (а также текст). Растровые изображения и шрифты, привязанные к целым пикселям, при размещении с использованием плавающих точек отображались некорректно. Работая в программе **Flash 5**, необходимо было все время проверять, чтобы объекты были помещены с привязкой только к целым значениям. Благодаря команде **Snap to Pixels** появилась возможность выполнять точное выравнивание по целым числам, как показано на рис. 1.3. Для включения функции привязки к пикселям выполните команду **View**⇒**Snap to Pixels**.

Совет

Подробная информация о векторном формате **Flash** приведена в главе 3 "Рисование во **Flash**". Растровые изображения подробно описаны в главе 4 "Использование растровых изображений во **Flash**", а растровые шрифты — в главе 5 "Работа с текстом".

ПАПКИ СЛОЕВ

Еще одним блоком программы, позволяющим оптимизировать и систематизировать рабочий процесс, являются папки панели **Timeline** (Временная шкала). Дело в том, что в процессе работы над проектом количество применяемых слоев увеличивается в угрожающей прогрессии. Например, при поиске видеоклипа может возникнуть необходимость просмотреть около 50 слоев. Теперь слои можно группировать в папки и сворачивать и разворачивать их по мере необходимости. Это позволяет в значительной степени оптимизировать рабочее пространство экрана, максимально увеличить размеры рабочего поля и в гораздо большей степени (по сравнению с возможностями, доступными в предыдущих версиях программы) систематизировать работу над проектом. К примеру, слои с кнопками можно сгруппировать в папку "Кнопки". Рабочий процесс оптимизируется за счет того, что теперь любой пользователь, даже совершенно незнакомый с проектом, может легко найти нужные объекты и код на временной шкале.

ШАБЛОНЫ

В программе Flash MX имеется несколько предварительно заданных шаблонов для работы со специальным содержимым, например, рекламных объявлений, презентаций или графики для мобильных устройств. Пример шаблонов, предназначенных для создания презентаций, приведен на рис. 1.4. Пользователю необходимо только добавить свой текст и графику, а о функциональности презентации позаботится сама программа.

Шаблоны, предназначенные для создания рекламных объявлений, комплектуются направляющим слоем и поясняющими инструкциями (рис. 1.5). Можно создать шаблоны с заблокированным содержимым, чтобы впоследствии заказчик мог внести в созданный файл минимальные изменения и обновления. Для доступа к шаблонам выполните команду **File⇒New from Template** (Файл⇒Создать на основе шаблона).

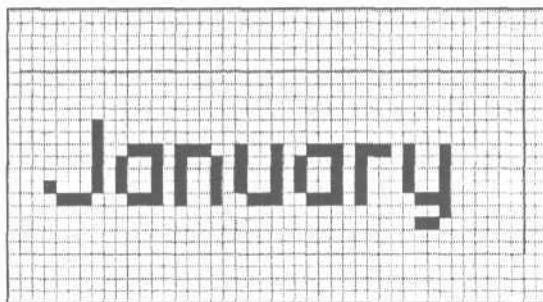


Рис. 1.3. Команда **Snap to Pixels** позволяет точно выровнять объект по заданным координатам пикселей

КОМПОНЕНТЫ

В программе Flash 5 была введена концепция Smart Clips (Интеллектуальные клипы), позволяющая повторно использовать видеоклипы со встроенными функциями и заданными параметрами. Эта концепция была усовершенствована и интегрирована в версию MX под названием *компоненты*. Панель Components (Компоненты), изображенная на рис. 1.6, представляет собой библиотеку наиболее распространенных элементов интерфейса, таких как линейки прокрутки и флаговые поля. Благодаря тому, что такие элементы всегда находятся под рукой, пользователь может просто перетянуть, например, полосу прокрутки на текстовое поле, как показано на рис. 1.6. Добавив к фильму нужный элемент, можно настроить предварительно заданные параметры сценария. После выхода программы Flash MX компания Macromedia выпустила два набора новых компонентов, предоставляющих доступ к таким элементам интерфейса, как меню с древовидной структурой, индикаторы выполнения задач и календари. В настоящее время в разработке находится много новых компонентов, что позволяет надеяться на дальнейшее усовершенствование процесса авторских разработок.

ПОДДЕРЖКА ВИДЕО

Во Flash 5 не было настоящей поддержки видео, его можно было только импортировать как серию статических изображений или экспортировать в QuickTime. Версия MX полностью поддерживает внедренное видео, позволяя работать без необходимости обращения к внешним плеерам. Видео можно импортировать в нескольких форматах и с помощью кодека (устройства кодирования и декодирования видео) Sorenson Spark, изображенного на рис. 1.7, внедрять непосредственно в SWF-файлы. Импортированным видео можно управлять (вращать, масштабировать, наклонять, маскировать и анимировать) точно так же, как и другими элементами. Им можно управлять и с помощью **ActionScript**. Кодека Sorenson Spark обеспечивает существенное сжатие больших видеофайлов. Для импортирования видео и доступа к кодеку Sorenson Spark выполните команду **File⇒Import** (Файл⇒Импортировать) и найдите нужный видеофайл на своем компьютере. Процедура сжатия выполняется при экспортировании Flash-фильма.

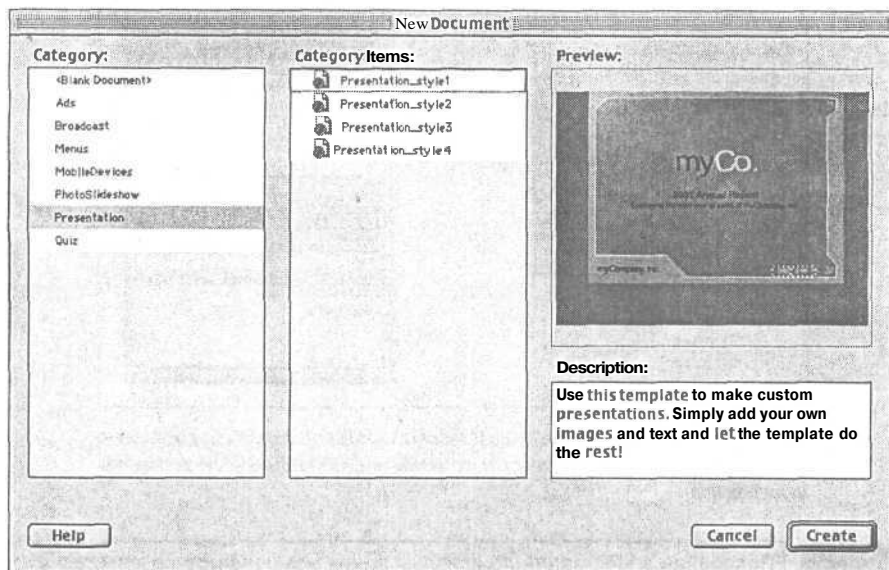


Рис. 1.4. Во Flash MX имеются предварительно заданные шаблоны для работы с определенным содержанием, а также наиболее распространенные навигационные кнопки

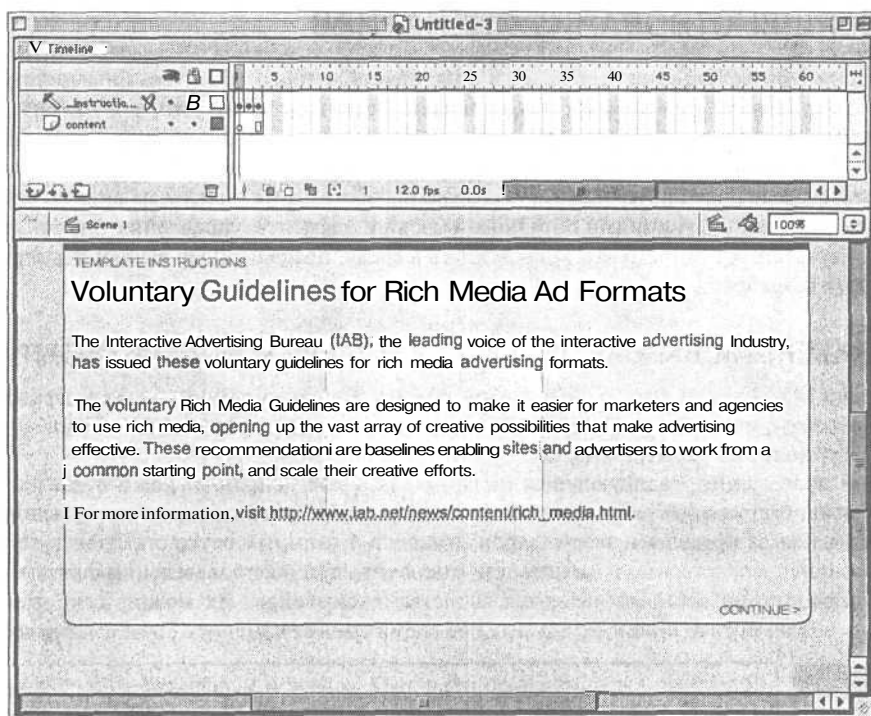


Рис. 1.5. Рекламные шаблоны включают инструкции по использованию наиболее распространенных рекламных форматов

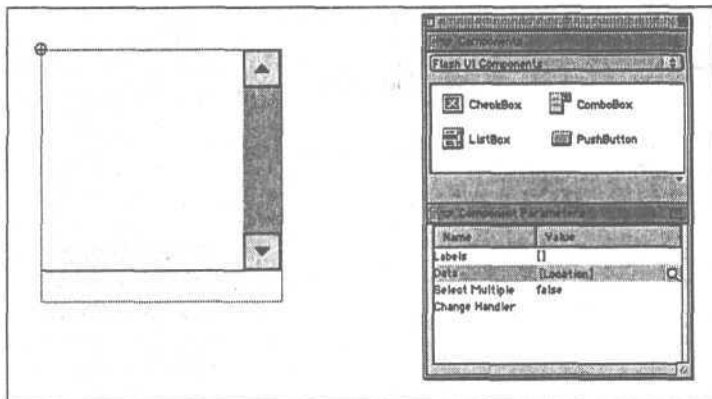


Рис. 1.6. Панель *Components* пользовательского интерфейса позволяет перетаскивать наиболее распространенные функциональные элементы интерфейса

Совет

Подробная информация об использовании компонентов содержится в главе 9 "Кнопки, меню и поля ввода данных", а процедура создания пользовательских компонентов описывается в главе 22 "Компоненты".

УСОВЕРШЕНСТВОВАНИЕ СЦЕНАРИЕВ

В версии **MX ActionScript** выступает в роли обособленного объектно-ориентированного языка. Усовершенствованная объектная модель позволяет интегрировать видеоклипы, кнопки и текстовые поля, а расширенная модель событий обеспечивает гораздо более широкие возможности взаимодействия пользователя с программой. Благодаря этим усовершенствованиям пользователь получил возможность создавать собственные подклассы видеоклипов, обработчики событий с функциями обратного вызова и элементы управления текстовыми полями. В результате этого **Rash MX** превратилась в среду, позволяющую выполнять серьезные прикладные разработки.

УСОВЕРШЕНСТВОВАННАЯ ОБЪЕКТНАЯ МОДЕЛЬ и МОДЕЛЬ СОБЫТИЙ

В версии **MX** объекты интегрированы более тесно. Теперь кнопки представляют собой истинные объекты, свойствами которых можно управлять посредством **ActionScript**, а видеоклипы могут получать события кнопок.

Будучи программой, базирующейся на событиях, **Rash** исполняет код в ответ на какие-либо события. Во **Flash 5** обработчики событий **ActionScript**, сигнализирующие о выполнении кода, оставались за пределами экземпляров символов, к которым осуществлялся доступ, и их нельзя было непосредственно изменить или отключить при воспроизведении фильма. В версии **MX** обработчики событий являются свойствами символов. Их можно даже поместить внутрь символов, так что пользователь в любое время сможет изменить обработчики событий.

Совет

Подробная информация об объектной модели и модели событий приводится в главе 11 "Знакомство с ActionScript", в главе 16 "Взаимодействие, события и установление последовательности" и в главе 17 "Демонстрация мощи видеоклипов".

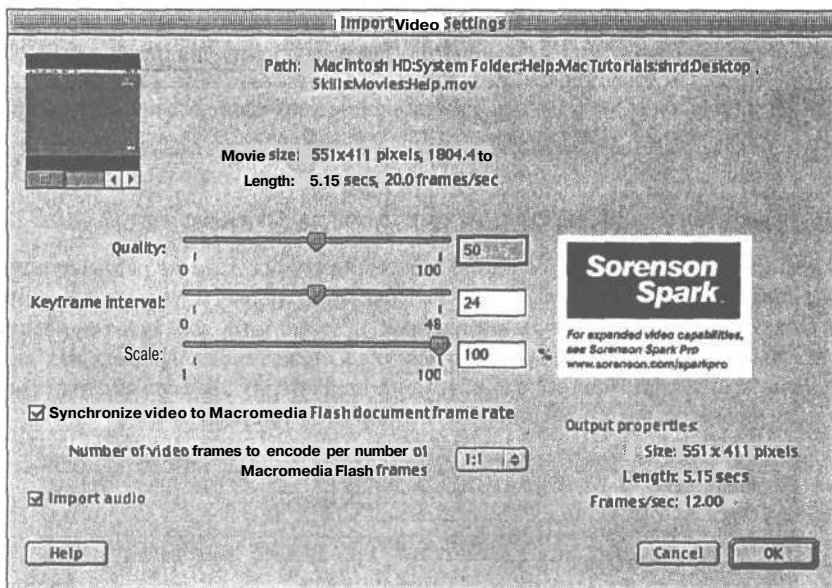


Рис. 1.7. Новый кодек Sorenson Spark позволяет получать высококачественное видео малого объема и внедрять его непосредственно в SWF-файлы

МАСКИРОВАНИЕ по СЦЕНАРИЮ

Во Flash 5 маски были недоступными для внесения изменений. В версии MX видеоклипы могут быть маскированы, а это означает, что для масок можно писать сценарии и они могут откликаться на действия пользователя. Эта функция открывает совершенно новые возможности в области дизайна. Маска может откликаться на перемещение мыши пользователем, отображая содержимое при наведении указателя мыши на объект.

Совет

Подробно о масках по сценарию рассказывается в главе 17, "Демонстрация мощи видеоклипов".

ПРОГРАММНОЕ РИСОВАНИЕ

Новый прикладной программный интерфейс рисования открывает новые возможности рисования изображений в объекте видеоклипа. Посредством **ActionScript** новые видеоклипы можно создавать с нуля без привлечения графических средств. С помощью ActionScript можно нарисовать весь графический интерфейс, и при этом предварительно созданные изображения не потребуются.

Совет

О программном рисовании вы узнаете в главе 18 "Процедура рисования с помощью ActionScript".

ЗАГРУЗКА ВНЕШНИХ ИЗОБРАЖЕНИЙ и ЗВУКА

До появления Flash MX внешние изображения и звук нельзя было загружать динамически. Благодаря новым функциям программы MX, `loadMovie()` и `loadSound()`, файлы JPEG и MP3 можно динамически загружать при помощи URL. Это означает, что размер

Flash-фильмов можно уменьшить с помощью средств, загружаемых в процессе их выполнения. Можно также обновить содержимое фильма без редактирования исходных Flash-файлов.

Совет

Подробная информация о загрузке внешних изображений и звука приводится в главе 20 "Использование встроенных объектов фильмов".

УСОВЕРШЕНСТВОВАННЫЙ РЕДАКТОР ACTIONSCRIPT

Усовершенствования интерфейса Flash MX распространяются и на редактор ActionScript. Теперь при вводе кода можно открыть панель **Code Hints** (Советы по использованию кода), настроить отображение кода различными цветами, а кроме того, добавлена нумерация строк кода, как показано на рис. 1.8. Для быстрого доступа к справочной информации с редактором связана панель ActionScript Reference (Справка по ActionScript). Все это облегчает написание кода лицам, не являющимся профессиональными программистами.

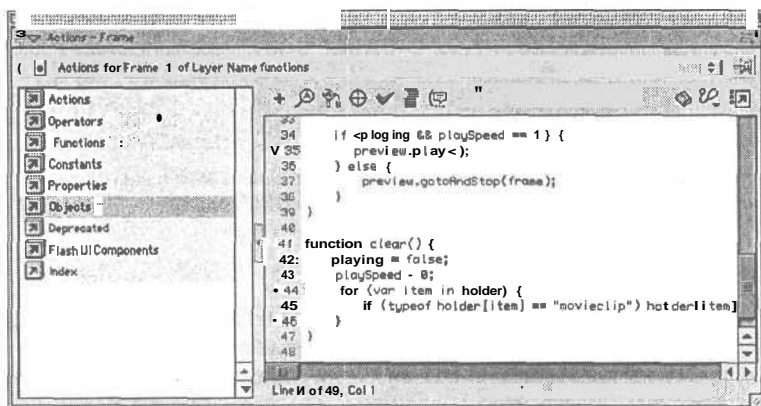


Рис. 1.8. Усовершенствованный редактор ActionScript введена функция нумерации строк и советов по использованию кода

Совет

О том, как использовать редактор ActionScript, рассказывается в главе 11 "Знакомство с ActionScript".

СОВМЕСТИМОСТЬ С FLASH 5

Предыдущим версиям Flash не доставало обратной совместимости. В версии MX файлы .fla можно сохранять как документы Flash 5 (рис. 1.9), чтобы их мог открыть любой пользователь, работающий с программой версии 5 или MX. Это позволяет сделать содержимое доступным для пользователей, на компьютерах которых установлены предыдущие версии Flash-плеера. При сохранении файла как документа Flash 5 на экране появится предупреждение о том, что в сохраненном файле будет опущено содержимое, специфичное для версии MX.

ДОСТУП К СОДЕРЖИМОМУ

Ранее содержимое Flash было недоступно для программ считывания экрана, которыми пользуются люди с физическими ограничениями. В версии MX имеется панель Accessibility (Доступность), изображенная на рис. 1.10, позволяющая добавить к содержимому Flash над-

ПИСИ и описания (в HTML эту функцию выполняет дескриптор <alt>), чтобы сущность Flash-содержимого могла быть описана программами считывания и другими вспомогательными средствами.

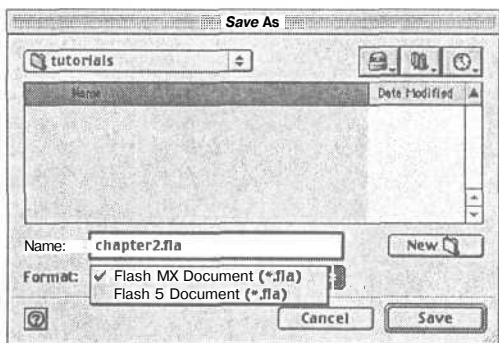


Рис. 1.9. Flash-файлы можно сохранять в формате Flash MX или Flash 5

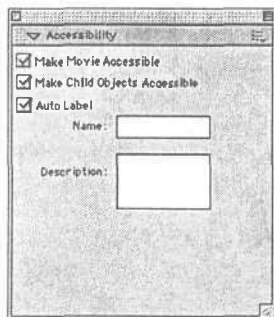


Рис. 1.10. В окне Accessibility можно ввести надписи и описание содержимого так, чтобы оно стало доступным для программ считывания информации с экрана

Совет

Подробная информация о параметрах доступности Flash приведена в приложении А.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Куда делась программа Generator?

Разработчики Flash MX отказались от использования Generator. Теперь большинство функций этой программы, например взаимосвязь с базами данных посредством XML, встроено непосредственно во Flash MX. Кроме того, в ближайшем будущем планируется интеграция программы с ColdFusion MX и другими инструментальными платформами.

Что случилось с опциями вращения и масштабирования инструмента Arrow (Стрелка) программы Flash 5?

Все параметры преобразования, в том числе вращение и масштабирование, теперь входят в состав инструмента Free Transform (Свободное преобразование). Выберите этот инструмент и щелкните на объекте. Для масштабирования объекта перетащите его угловую точку, а для вращения объекта щелкните мышью и удерживайте ее нажатой вне объекта до тех пор, пока не появится круг со стрелкой (пиктограмма вращения). Затем начинайте вращение объекта.

ИНТЕРФЕЙС FLASH

В ЭТОЙ ГЛАВЕ...

Наборы панелей	39
Задание атрибутов документа	41
Временная шкала	42
Панель инструментов	45
Клавиши быстрого доступа	50
Библиотека	52
Проводник фильма	53
Задание параметров Flash	53
Возможные проблемы	58
Flash за работой: создание идеального интерфейса Flash MX	59

НАБОРЫ ПАНЕЛЕЙ

При первом же запуске Flash MX вы обнаружите существенные изменения, внесенные в ее интерфейс. Теперь дизайнеры и разработчики имеют гораздо больше возможностей для настройки среды проектирования и оптимизации рабочего процесса.

Интерфейс Flash состоит из окна фильма, панели инструментов и других многочисленных панелей. Размещение всех этих элементов можно настроить в соответствии с собственными предпочтениями.

Пользовательский интерфейс в версии MX значительно образом модернизирован. При работе с Flash пользователю всегда не хватает места на экране (хотя в настоящее время большинство мониторов работает с разрешением 1024x768, а не 800x600). В связи с этим новые наборы па-

нелей и инспекторы свойств были спроектированы так, чтобы максимально увеличить рабочую область экрана. Все панели, за исключением панелей Timeline (Временная шкала), Stage (Рабочее поле) и инспектора свойств для Macintosh, можно сделать плавающими, сворачивать или разворачивать по мере необходимости. Пользователи Macintosh не могут привязать никакие другие панели к панелям Timeline и Stage, а панель инспектора свойств всегда является плавающей. Можно привязать другие панели к панели Actions (Действия), но на компьютерах Macintosh последняя всегда является плавающей. Быстро скрыть и отобразить все панели можно с помощью клавиши <Tab>. Разумеется, что наборы панелей в значительной степени влияют на удобство и простоту использования интерфейса Flash MX в целом.

В программе Flash MX имеются следующие панели:

- Toolbox (Панель инструментов);
- Stage (Рабочее поле);
- Align (Выравнивание);
- Color Swatches (Каталоги цветов);
- Transform (Преобразование);
- Debugger (Отладчик);
- Reference (Справка);
- Accessibility (Доступность);
- Component Parameters (Параметры компонентов);
- Answers (Ответы);
- Timeline (Временная шкала);
- Property Inspector (Инспектор свойств);
- Color Mixer (Цветовой микшер);
- Info (Информация);
- Actions (Действия);
- Movie Explorer (Проводник фильма);
- Output (Вывод данных);
- Components (Компоненты);
- Library (Библиотека).

ИНСПЕКТОР СВОЙСТВ

Параметры, доступные на панели Property Inspector (Инспектор свойств), являются контекстно-зависимыми и открываются в зависимости от того, какой объект был выделен. Панель Property Inspector позволяет чрезвычайно быстро редактировать объекты. Панели Stroke (Штрих), Fill (Заливка), Text (Текст), Paragraph (Абзац), Character (Знак), Instance (Экземпляр), Frame (Кадр), Effect (Эффект) и Sound (Звук), применявшиеся в программе Flash 5, полностью заменены панелью Property Inspector (рис. 2.1). К сожалению, мечта реализована не полностью, и доступ к некоторым инструментам (например, Round Rectangle Radius, с помощью которого можно нарисовать прямоугольник со скругленными углами) пока что можно получить только в подразделе Options (Параметры) панели инструментов либо на других панелях. Некоторые параметры повторяются на разных панелях. В частности, параметры заливки и штрихования доступны на панели инструментов, а также на панели Property Inspector и Color Mixer. Тем не менее панель Property Inspector является существенным усовершенствованием интерфейса и практически полностью избавляет от необходимости открывать многочисленные диалоговые окна.

КОМПОНОВКА ПАНЕЛЕЙ

Чтобы связать несколько панелей, перетащите панель за левый угол к другой панели. При наведении указателя мыши в нужное место (под кнопкой закрытия панели, рядом со строкой заголовка) появится пиктограмма с изображением руки (рис. 2.2). Чтобы скрыть привязанную панель, перетащите ее за левый угол в сторону от остальных связанных с ней панелей, а затем щелкните на кнопке закрытия, расположенной в верхнем левом углу. Если связано несколько панелей, кнопку закрытия имеет только самая верхняя из них и после щелчка на ней будут закрыты *все* связанные панели.

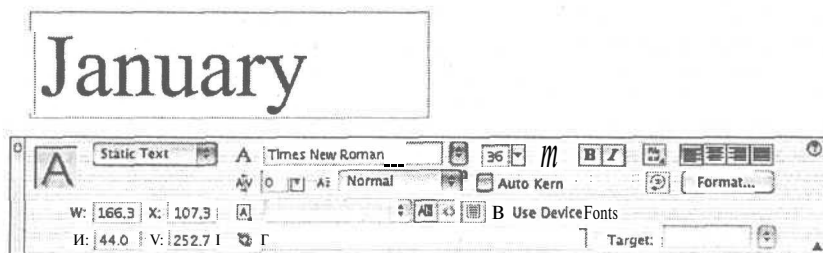


Рис. 2.1. Параметры новой панели Property Inspector являются контекстно-зависимыми

Во Flash MX имеются заданные по умолчанию наборы панелей, доступ к которым можно получить с помощью команды **Window**⇒**Panel Sets** (Окно⇒Наборы панелей). В открывшемся подменю будут доступны наборы Default (По умолчанию), Designer (Дизайнер) и Developer (Разработчик) с различными разрешениями. Очень важным является и то, что пользователь может сохранять свои собственные наборы панелей. Для этого нужно создать нужный набор и выполнить команду **Window**⇒**Save Panel Layout** (Окно⇒Сохранить компоновку панелей).

ЗАДАНИЕ АТТРИБУТОВ ДОКУМЕНТА

Для доступа к атрибутам документа, формально именуемым свойствами фильма, предназначена панель Property Inspector либо команда **Modify**⇒**Document** (Изменить⇒Документ). На панели инспектора свойств показаны атрибуты документа, задаваемые по умолчанию при создании фильма, доступ к которым можно также получить, щелкнув в области фильма, не выделяя никаких объектов. После щелчка на кнопке Size (Размер) панели инспектора свойств откроется диалоговое окно Document Attributes (Атрибуты документа), в котором можно задать размеры рабочей области, параметры согласования, фоновый цвет, частоту смены кадров и **единицы** измерения линеек. После щелчка на кнопке Publish (Публикация) откроется диалоговое окно Publish Settings (Параметры публикации), а после щелчка в поле Background (Фон) откроется каталог цветов для выбора фонового цвета рабочей области. Скорость воспроизведения фильма изменяется в единицах кадров в секунду (fps). Это значение можно ввести прямо в поле Frame Rate (Частота смены кадров).

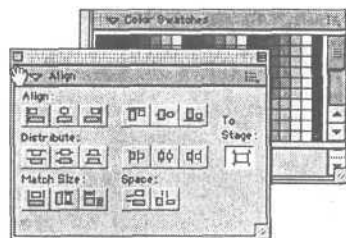


Рис. 2.2. Чтобы связать панели, щелкните рядом со строкой заголовка, в результате чего появится пиктограмма с изображением руки. Перетащите панель, расположив ее поверх другой, и отпустите кнопку мыши

ВРЕМЕННАЯ ШКАЛА

Как и традиционный фильм, видеоклип Flash разворачивается во времени. Центром управления содержимым Flash-фильма является панель Timeline (Временная шкала), позволяющая управлять содержимым, отображаемым в течение определенного времени в различных кадрах и ключевых кадрах. На панели Timeline (рис. 2.3) "расквартированы" все кадры и слои, составляющие фильм, а также считывающая головка, являющаяся индикатором текущего кадра.

КАДРЫ

Кадры представляют собой отдельные статические кванты, или моменты времени, которые комбинируются друг с другом для создания эффекта движения. Вспомните книжку-раскладку, с помощью которой создавался примитивный эффект анимации. На каждой странице был изображен отдельный статический рисунок, но при быстром перелистывании страниц создавалась иллюзия движения. Кадры позволяют устанавливать последовательность и управлять скоростью воспроизведения фильма, а также задают его длину в целом. Как и в случае с фильмом, анимация, происходящая в меньшем количестве кадров, в конечном видеоролике будет казаться более быстрой, чем анимация с большим количеством кадров. Для вставки кадров выполните команду **Insert⇒Frame (Вставка⇒Кадр)** или нажмите клавишу <F5>.

КЛЮЧЕВЫЕ КАДРЫ

Ключевые кадры сигнализируют об изменениях в анимации. В кадрах хранится содержимое предшествующих им ключевых кадров. И только после прохождения нового ключевого кадра может измениться содержимое области фильма в пределах одного слоя.

Для вставки ключевого кадра выполните команду **Insert⇒Keyframe (Вставка⇒Ключевой кадр)** или нажмите клавишу <F6>. В любом ключевом кадре, следующем за другим ключевым кадром, отображается содержимое первого ключевого кадра, что позволяет пользователю вносить изменения в исходное содержимое. Для вставки ключевого кадра с пустым содержанием выполните команду **Insert⇒Blank Keyframe (Вставка⇒Пустой ключевой кадр)** или нажмите клавишу <F7>. На временной шкале ключевые кадры изображаются черными кружками, а пустые ключевые кадры — пустыми кружками.

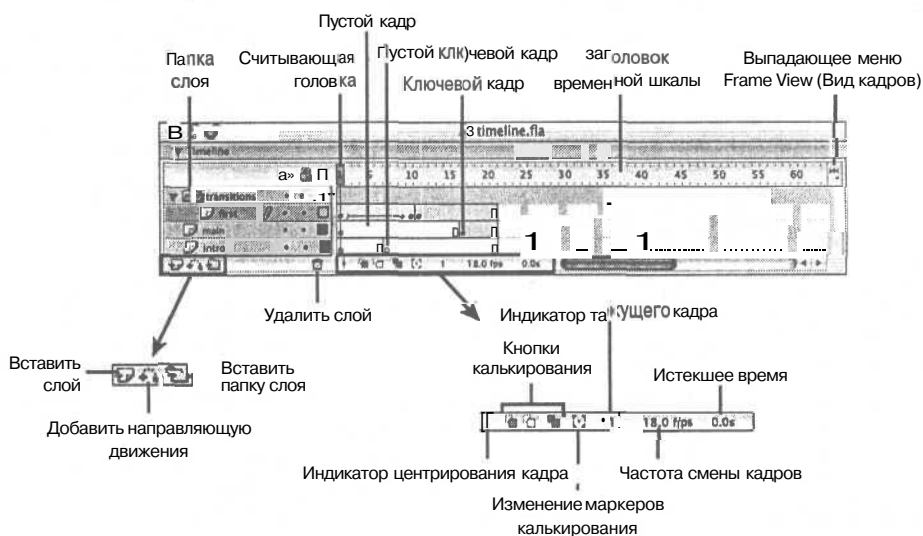


Рис. 2.3. На панели *Timeline* осуществляется управление покадровой анимацией

СЛОИ И ПАПКИ СЛОЕВ

Слои позволяют компоновать содержимое фильма. Каждый слой можно считать отдельной полосой фильма на куске чистой ацетатной ткани. Если на каком-то участке слоя содержимое отсутствует, то сквозь данный слой будет "просвечивать" содержимое слоя, находящегося под ним. Если на каком-то участке верхнего слоя имеется содержимое, оно будет закрывать нижние слои.

Слои позволяют систематизировать и разделить содержимое. Содержимым легче управлять, если оно разнесено по слоям. Особенностью Flash является возможность слияния простых несгруппированных изображений, существующих в пределах одного слоя, если они перекрываются или соприкасаются. На рис. 2.4 показано, что если два красных круга, расположенных в одном слое, соприкасаются, они становятся единым объектом. Разнесение содержимого в отдельные слои позволяет анимировать объекты индивидуально, с достижением весьма сложных эффектов. Добавление слоев не приводит к увеличению размера файла фильма, поэтому слой можно смело использовать при необходимости организации содержимого.

По мере создания слоев Flash автоматически нумерует их и присваивает им имена. Для переименования слоя дважды щелкните на его имени (например, Layer 1), расположенном в левой части панели Timeline (Временная шкала). Уделите побольше внимания присвоению имен слоям, делая их достаточно короткими, но при этом несущими смысловую нагрузку. В дальнейшем слои будут служить указателем расположения объектов. По мере увеличения проекта и количества слоев в них можно будет легко запутаться, но этого никогда не произойдет, если слои будут иметь четкие описательные имена.

Пиктограммы, расположенные справа от каждого слоя, позволяют скрывать и блокировать слои, а также отображать их в виде контуров, как показано на рис. 2.5. Слои, с которыми вы не работаете в данный момент, стоит заблокировать, что позволит избежать непреднамеренного изменения содержимого. Удерживая клавишу <Option> (для Macintosh) или <Alt> (для Windows), щелкните на кружке, расположенном под пиктограммой с изображением замка. В результате этой операции будут заблокированы все остальные слои. Чтобы заблокировать или снять блокировку отдельных слоев, щелкните на кружках, расположенных в столбце блокировки. О блокировании слоев свидетельствует пиктограмма с изображением замка. Чтобы скрыть слой и увидеть содержимое, расположенное под ним, воспользуйтесь кружками, которые находятся под пиктограммой с изображением глаза. Аналогичным образом, щелчок на кружке видимости при нажатой клавише <Option> или <Alt> в строке одного из слоев приведет к скрытию всех остальных слоев. О скрытии слоев свидетельствует значок X красного цвета, расположенный в левом столбце под пиктограммой с изображением глаза. Слои можно отображать в виде контуров, что может оказаться полезным при наличии большого количества перекрывающихся элементов на различных слоях. При отображении слоя в виде контура цветной квадрат, расположенный в правом столбце, станет незаполненным.

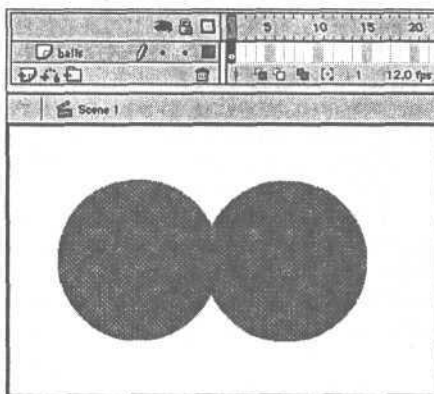


Рис. 2.4. В случае соприкосновения простых несгруппированных объектов, расположенных в одном слое, они будут объединены, в результате чего образуется единый объект

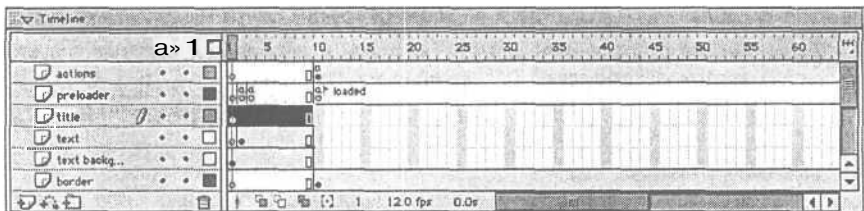
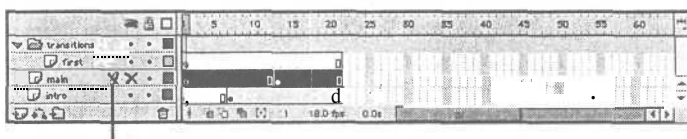


Рис. 2.5. Пиктограммы слоя позволяют скрыть или заблокировать его, а также отобразить в виде контура

Пиктограмма с изображением карандаша, расположенная между именем слоя и столбцами пиктограмм, указывает на слой, выделенный в настоящий момент и доступный для редактирования. Имейте в виду, что скрытый слой редактировать нельзя. При попытке отредактировать скрытый слой появится пиктограмма с изображением перечеркнутого карандаша, как показано на рис. 2.6.



Перечеркнутая пиктограмма карандаша

Рис. 2.6. Скрытый слой является недоступным для редактирования, о чем свидетельствует перечеркнутая пиктограмма с изображением карандаша

Чтобы изменить расположение слоев, щелкните на имени слоя и перетащите его на новое место. Чтобы одновременно выделить несколько слоев, при щелчке на слое удерживайте нажатой клавишу <Shift>.

Пиктограммы, расположенные в нижней части области слоев в окне Timeline, позволяют добавить новый слой, направляющий слой или вставить папку слоя. Направляющие слои позволяют выравнивать объекты. Их можно использовать для статического выравнивания элементов (например, для выравнивания объектов на других слоях относительно объектов или сетки, созданных на направляющем слое) или в качестве направляющих движения, позволяющих выравнивать объекты вдоль пути их перемещения. Чтобы превратить любой слой в направляющий, щелкните на пиктограмме Add Motion Guide (Добавить направляющую движения). В опубликованных Flash-фильмах направляющие слои не отображаются.



Превосходной новой функцией Flash MX являются папки слоев. Теперь слои можно группировать и систематизировать в папках. Это позволяет не только легко найти нужные элементы фильма, но и не загромождать рабочую область экрана, открывая только папки, содержащие слои, с которыми в данный момент необходимо работать. Чтобы поместить слой в папку, достаточно просто перетащить его на пиктограмму папки. Чтобы развернуть структуру папки слоя, щелкните на стрелке, расположенной слева от пиктограммы папки.

КАЛЬКИРОВАНИЕ И РЕДАКТИРОВАНИЕ НЕСКОЛЬКИХ КАДРОВ

В области фильма одновременно отображается только один кадр. Для облегчения процесса выравнивания, особенно при работе с покадровой анимацией, с целью одновременного отображения ряда кадров в области фильма применяется функция калькирования. На трех кнопках, расположенных в нижней части панели Timeline, изображены по-разному наложен-

ные друг на друга квадраты. После щелчка на кнопке Onion Skin (Калькирование) одновременно будет отображено несколько кадров, как показано на рис. 2.7.

Текущий кадр (под головкой воспроизведения) является полноцветным, а окружающие его кадры имеют серый цвет и недоступны для редактирования. Изменить количество видимых кадров можно путем перетаскивания маркеров Start Onion Skin (Начало калькирования) и End Onion Skin (Конец калькирования). При перемещении воспроизводящей головки изменяется текущий кадр и место, в котором начинается и завершается диапазон калькируемых кадров.

После щелчка на кнопке Onion Skin Outlines (Контурное калькирование) производится отображение объектов, находящихся в пределах диапазона калькируемых кадров, в виде контуров. Для редактирования всех калькируемых кадров щелкните на кнопке Edit Multiple Frames (Редактирование всех кадров). Рядом с тремя описанными выше кнопками располагается и четвертая, Modify Onion Markers (Изменение маркеров калькирования). После щелчка на ней откроется список опций, позволяющих закрепить маркеры так, что они не будут двигаться при изменении текущего кадра. Здесь же можно выбрать число калькируемых кадров: Onion 2, Onion 5 или Onion All.

ПАНЕЛЬ ИНСТРУМЕНТОВ

На этой панели (рис. 2.8) размещаются инструменты, предназначенные для работы во Flash. Данная панель подразделяется на четыре области: Tools (Инструменты), View (Вид), Colors (Цвета) и Options (Параметры). Некоторые из инструментов имеют модификаторы, отображаемые в области Options при их выборе.

ПОДРАЗДЕЛ TOOLS

В верхней части панели инструментов в подразделе Tools располагаются инструменты выбора, рисования, раскрашивания и изменения объектов.

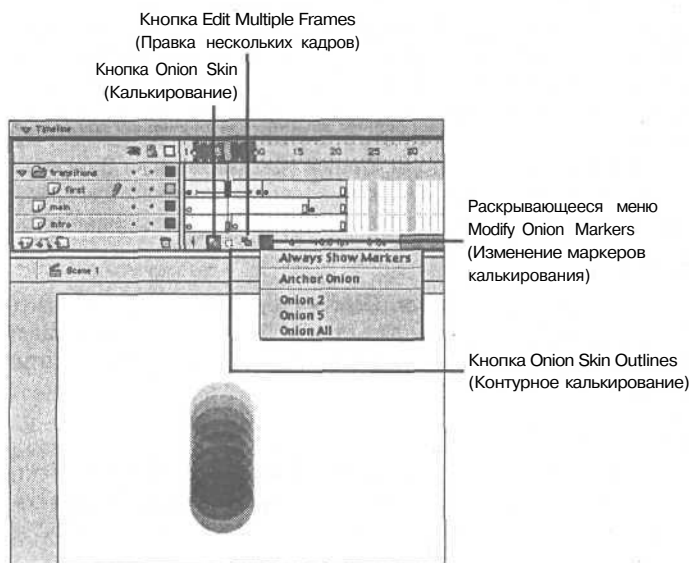


Рис. 2.7. Для одновременного отображения нескольких кадров перетаскивайте маркеры калькирования

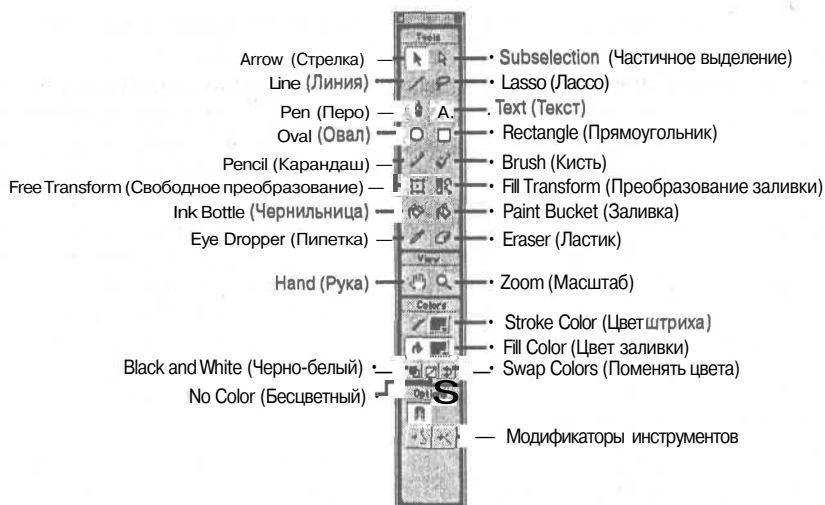


Рис. 2.8. Панель инструментов Flash MX

ИНСТРУМЕНТ ARROW

Инструмент Arrow (Стрелка) используется для выделения объектов в рабочей области. С помощью этого инструмента можно одним щелчком выделить объект, нарисовать область для выбора нескольких объектов или дважды щелкнуть им, чтобы выделить области заливки и штрихования объекта. Правда, после щелчка на изображении, содержащем и штрихи, и заливку, будет выделена только та область, на которой был произведен щелчок. Поскольку инструментом Arrow вы будете пользоваться чаще других, запомните, что ему соответствует клавиша быстрого доступа <V>.

При выборе инструмента Arrow обратите внимание на то, что в разделе Options панели инструментов будут отображены три модификатора. Первый из них, имеющий вид кнопки с изображением магнита, называется Snap to Objects (Привязка к объектам). При его выборе (по умолчанию) на конце указателя мыши будет отображаться кольцо, увеличивающееся по мере перетаскивания какого-либо объекта к другому, к которому он может быть привязан. Когда будет выделен объект или его часть, станут доступными модификаторы Smooth (Сглаживание) и Straighten (Выпрямление). Чтобы добавить точки для **скругления** изображения или удалить точки для его спрямления, выделите линию или часть контура изображения и щелкните на нужном модификаторе.

ИНСТРУМЕНТ SUBSELECTION

С помощью инструмента Subselection (Частичное выделение) производится выделение отрезков прямых линий или контуров изображения. Его также можно использовать для отображения точек на линиях и контурах и последующего перетаскивания этих точек с целью изменения изображения.

ИНСТРУМЕНТ LINE

Щелкнув и перетащив инструмент Line (Линия), вы создадите прямую линию. Удерживая нажатой клавишу <Shift>, с помощью данного инструмента можно рисовать линии по горизонтали, вертикали или по диагонали.

ИНСТРУМЕНТ LASSO

С помощью инструмента Lasso (Лассо) можно выделить области неправильной формы. Он используется для "захвата" неудобных областей. Щелкните на инструменте и нарисуйте

область произвольной формы. Инструмент будет "привязан" к указателю мыши до тех пор, пока вы не отпустите кнопку, создавая тем самым конечную точку.

Инструмент Lasso имеет три модификатора. Два верхних называются Magic Wand (Волшебная палочка) и Magic Wand Settings (Параметры волшебной палочки). Они позволяют выделять отдельные участки растрового изображения. После щелчка на растровом изображении будут выделены пиксели в соответствии с выбранным цветовым диапазоном. Продолжайте щелкать для добавления выделенных участков. Одним из параметров данного модификатора является пороговое значение, определяющее, насколько чувствительной является "волшебная палочка" к цветовому контрасту соседних пикселей. Чем ниже значение, тем более точным будет выделение. Большие значения соответствуют более широкому диапазону цветов. Параметр сглаживания **определяет**, насколько сильно будут скруглены границы выделенного фрагмента. Последним модификатором является Polygon Mode (Режим многоугольника), при использовании которого выделенная область будет иметь вид многоугольника.

ИНСТРУМЕНТ PEN

Инструмент Реп (Перо) является наиболее мощным инструментом рисования. С его помощью можно точно нарисовать траекторию, состоящую из прямых и криволинейных участков. Чтобы нарисовать участок в виде прямой линии, просто щелкните на инструменте Реп, а для создания криволинейных участков щелкните на инструменте и перетащите его.

ИНСТРУМЕНТ TEXT

После щелчка на инструменте Text создается место для начала ввода текста. Атрибуты текста назначаются на панели инспектора свойств.

ИНСТРУМЕНТ OVAL

С помощью инструмента Oval (Овал) можно нарисовать овалы и идеальные круги. Чтобы нарисовать овал, щелкните и перетащите инструмент; для создания круга при щелчке и перетаскивании удерживайте нажатой клавишу <Shift>.

ИНСТРУМЕНТ RECTANGLE

Инструмент Rectangle (Прямоугольник) позволяет рисовать прямоугольники и квадраты. Чтобы нарисовать прямоугольник, щелкните и перетащите инструмент; для создания квадрата при щелчке и перетаскивании удерживайте нажатой клавишу <Shift>. Инструмент Rectangle имеет модификатор Round Rectangle Radius (Радиус скругления углов прямоугольника), позволяющий задать радиус скругления углов.

ИНСТРУМЕНТ PENCIL

Инструмент Pencil (Карандаш) позволяет рисовать неровные линии. В отличие от инструментов Реп или Line, с помощью которых можно рисовать только от точки до точки, инструмент Pencil движется вслед за мышью и является цифровым эквивалентом рисования карандашом. Модификатор Pencil Mode (Режим карандаша) позволяет выпрямлять или скруглять траектории при рисовании.

ИНСТРУМЕНТ BRUSH

Инструмент Brush (Кисть) имитирует мазок кистью при рисовании. Это лучший инструмент окрашивания, имеющийся во Flash. Его модификаторы позволяют выбрать режим окрашивания, размер и форму кисти, а также заблокировать цвет заливки. Модификатор Brush Mode (Режим кисти) позволяет рисовать поверх штрихов или заливки либо защищать части объекта, подобно использованию трафарета. В режиме Paint Fills (Закрашивание заливки) окрашивается та часть рисунка, где имеется заливка или пустые участки, а штрихи и контуры остаются нетронутыми. В режиме Paint Behind (Закрашивание позади) объекты рисунка защищены и окрашивать разрешается только пустые участки. В режиме Paint Selection

(Закрашивание выделенного фрагмента) окрашивается только заливка выделенного объекта. Однако использование данного режима является излишне сложным способом изменения заливки, поэтому я настоятельно рекомендую просто выделить объект и применить к нему новую заливку посредством поля Fill (Заливка) на панели инспектора свойств либо в разделе Color панели инструментов. Последним режимом модификатора Brush Mode является Paint Inside (Закрашивание внутри контура), который позволяет применять цвет к области заливки, где был начат мазок, но не к его контурам.

Если у вас есть чувствительный к нажиму графический планшет, то будет доступен еще один модификатор — Pressure (Нажим). С его помощью можно изменять размер мазка кисти в соответствии с силой нажима при рисовании. Чем сильнее нажим, тем толще будет мазок.

ИНСТРУМЕНТ FREE TRANSFORM



Инструмент Free Transform (Свободное преобразование) является нововведением Flash MX. Он позволяет выполнять преобразования как индивидуально, так и в комбинации с объектами, группами, экземплярами и текстовыми блоками. Для выполнения преобразования выделите объект в рабочем поле и щелкните на инструменте Free Transform. Вокруг выделенного объекта появится ограничивающий контур, а по мере перемещения указателя мыши вдоль границ объекта он будет изменять свой внешний вид, что свидетельствует о возможности выполнения тех или иных преобразований.

Данный инструмент имеет модификаторы Rotate and Skew (Вращение и наклон), Scale (Масштабирование) и два новых: Distort (Искажение) и Envelope (Огибающая). Модификатор Distort позволяет промоделировать построение перспективы объекта. Для этого нужно щелкнуть и перетащить угловые или боковые маркеры ограничивающего контура. При этом будут преобразованы соседние края, как показано на рис. 2.9.

Как видно на рис. 2.10, модификатор Envelope позволяет деформировать и искажать выделенные объекты. Выделенный фрагмент находится в пределах так называемой огибающей, а для деформации ее содержимого достаточно перетащить маркеры Безье. Деформацию можно даже преобразовать в отдельные ключевые кадры, так что она будет выполняться пошагово в течение определенного временного интервала.



Рис. 2.9. Модификатор Distort изменяет форму и выравнивание объекта, моделируя его ракурс

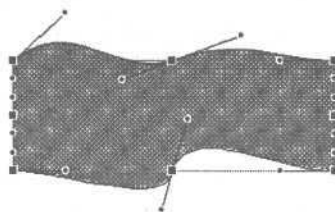


Рис. 2.10. Модификатор Envelope искажает и деформирует объекты

ИНСТРУМЕНТ FILL TRANSFORM

Помимо чистых цветов, в качестве заливки можно использовать градиентное и даже растровое заполнение. Инструмент Fill Transform (Преобразование заливки) позволяет отрегулировать градиентную и растровую заливки. При щелчке на градиентной или растровой заливке ее обрамление и середина помечаются круглыми и квадратными маркерами, которые можно перетаскивать. Если навести на один из этих маркеров указатель мыши, он изменит свой вид, указывая на опции редактирования. Пользователь может изменить форму, повернуть, масштабиро-

вать и изменить угол наклона градиентного заполнения или заливки растровым изображением. Можно даже создать мозаичное заполнение растровым изображением.

ИНСТРУМЕНТ INK BOTTLE

Инструмент Ink Bottle (Чернильница) используется для изменения существующего контура. При его использовании в сочетании с панелью инспектора свойств можно редактировать цвет, толщину и стиль штриха. Можно изменить любой из этих атрибутов, а для применения внесенных изменений достаточно щелкнуть на нужном штрихе.

ИНСТРУМЕНТ PAINT BUCKET

Инструмент Paint Bucket (Заливка) используется для изменения существующей заливки. Аналогично инструменту Ink Bottle, его можно использовать для редактирования заливки в сочетании с панелью инспектора свойств. Изменить заливку можно либо посредством панели инспектора свойств, либо поля Fill Color (Цвет заливки) на панели инструментов, а чтобы внести изменения, достаточно щелкнуть на инструменте Paint Bucket.

ИНСТРУМЕНТ EYEDROPPER

Инструмент Eyedropper (Пипетка) позволяет копировать атрибуты штриха или заливки из одного объекта и сразу же применять их к другому. Его можно использовать для выбора растрового изображения, которое будет использоваться в качестве заливки. Щелкните инструментом Eyedropper на штрихе или заливке, подлежащих копированию. В зависимости от того, был ли выделен штрих либо заливка, инструмент Eyedropper превратится или в Ink Bottle, или в Paint Bucket. При щелчке на другом объекте его штрих или заливка станут такими же, как и у первого.

ИНСТРУМЕНТ ERASER

С помощью инструмента Eraser (Ластик) можно стереть объекты во всем рабочем поле или в его части. При перетаскивании этого инструмента будет вытерто все, что окажется на его пути. Параметры модификатора Erase Mode (Режим ластика) позволяют указать элементы, подлежащие удалению, например, штрихи и заливку в пределах одного слоя (Erase Normal — Обычное стирание), только заливку (Erase Fills), только штрихи (Erase Strokes), только выделенную заливку (Erase Selected Fills) или только заливку, которую уже начали стирать (Erase Inside — Стирание внутри объектов). Модификатор Faucet (Кран) позволяет целиком удалить штрихи или заливку одним щелчком мыши. Модификатор Eraser Shape (Форма ластика) позволяет указать форму и размер ластика.

На заметку

Модификатор Faucet имеет неправильную форму. Выделение выполняет капля воды, а не сам кран. Поэтому при работе с маленькими объектами поместите каплю воды точно над фрагментом, который подлежит выделению.

ПОДРАЗДЕЛ VIEW

Просматривать рабочее поле и перемещаться в его пределах можно с помощью двух инструментов: Hand (Рука) и Zoom (Масштаб). Фактический размер фильма может составлять лишь небольшой участок рабочей области. Инструмент Zoom позволяет увеличивать и уменьшать участки рабочего поля. Щелкните на инструменте Zoom либо воспользуйтесь клавишей <M> или <Z>, а затем щелкните в рабочем поле или перетащите контур выделения над фрагментом, подлежащим увеличению. Для уменьшения выбранного фрагмента щелкните инструментом Zoom, удерживая нажатой клавишу <Option> (для Macintosh) или <Alt> (для Windows).

Инструмент Hand (Рука) позволяет перемещаться в пределах рабочего поля. Чтобы воспользоваться инструментом Hand при работе с каким-то другим инструментом, нажмите и удерживайте клавишу пробела. Обычно при масштабировании объекта возникает необходи-

мость переместить рабочую область так, чтобы она находилась над нужным фрагментом объекта. Для этого при выбранном инструменте Zoom нажмите клавишу пробела и сместите изображение в нужное место рабочей области. После того как клавиша пробела будет отпущена, вы вернетесь к тому инструменту, с которым работали ранее.

ПОДРАЗДЕЛ COLORS

В подразделе Colors панели инструментов задают цвета штрихов и заливки. Если в рабочем поле не выделен никакой фрагмент, то после щелчка в поле Stroke (Штрих) или Fill (Заливка) будут заданы цвета изображения, которое будет рисоваться в дальнейшем. Если в рабочем поле выделены какие-то объекты, то изменение цвета штриха или заливки на панели инструментов приведет к немедленному изменению соответствующих атрибутов этих объектов.

Под полями Fill и Stroke располагаются три кнопки. Кнопка Black and White (Черно-белый) изменяет цветовые параметры на заданные по умолчанию черный штрих и белую заливку. Щелчок на кнопке No Color (Бесцветный) отключает либо штрих, либо заливку, в зависимости от того, какой из этих атрибутов был выбран. Кнопка Swap Colors (Поменять цвета) изменяет цвет выделенного контура на цвет заливки и наоборот.

КЛАВИШИ БЫСТРОГО ДОСТУПА

В программе Rash MX имеются клавиши быстрого доступа ко многим панелям и командам меню. Они помогут сэкономить в процессе работы массу времени, поэтому уделите немного внимания тому, чтобы запомнить те клавиши и их комбинации, которыми вам придется пользоваться чаще всего.

В табл. 2.1 указаны клавиши быстрого доступа к различным инструментам. Поскольку инструментами приходится пользоваться достаточно часто, то для быстрого доступа к ним используются одиночные клавиши.

ТАБЛИЦА 2.1. КЛАВИШИ БЫСТРОГО ДОСТУПА к ИНСТРУМЕНТАМ

ИНСТРУМЕНТ	КЛАВИША БЫСТРОГО ДОСТУПА
Arrow (Стрелка)	<V>
Subselection (Частичное выделение)	<A>
Line (Линия)	<N>
Lasso (Лассо)	<L>
Pen (Перо)	<P>
Text (Текст)	<T>
Free Transform (Свободное преобразование)	<Q>
Fill Transform (Преобразование заливки)	<F>
Ink Bottle (Чернильница)	<S>
Paint Bucket (Заливка)	<K>
Eyedropper (Пипетка)	<I>
Eraser (Ластик)	<E>
Hand (Рука)	<H>
Zoom (Масштаб)	<M>, <Z>

В табл. 2.2 перечислены комбинации клавиш быстрого доступа к панелям, которые работают как выключатели, т.е. один раз их можно использовать для открытия панели, а второй раз — для скрытия.

ТАБЛИЦА 2.2. КЛАВИШИ БЫСТРОГО ДОСТУПА к ПАНЕЛЯМ

ПАНЕЛЬ	MACINTOSH	WINDOWS
Tools (Инструменты)	<Cmd+F2>	<Ctrl+F2>
Timeline (Временная шкала)	<Option+Cmd+T>	<Alt+Ctrl+F2>
Property Inspector (Инспектор свойств)	<Cmd+F3>	<Ctrl+F3>
Answers (Ответы)	<Option+F1>	<Alt+F1>
Align (Выравнивание)	<Cmd+K>	<Ctrl+K>
Color Mixer (Цветовой микшер)	<Shift+F9>	<Shift+F9>
Color Swatches (Каталоги цветов)	<Cmd+F9>	<Ctrl+F9>
Info (Информация)	<Cmd+I>	<Ctrl+I>
Scene (Сцена)	<Shift+F2>	<Shift+F2>
Transform (Преобразование)	<Cmd+T>	<Ctrl+T>
Actions (Действия)	<F9>	<F9>
Debugger (Отладчик)	<Shift+F4>	<Shift+F4>
Movie Explorer (Проводник фильма)	<Option+F3>	<Alt+F3>
Reference (Справка)	<Shift+F1>	<Shift+F1>
Output (Вывод данных)	<F2>	<F2>
Accessibility (Доступность)	<Option+F2>	<Alt+F2>
Components (Компоненты)	<Cmd+F7>	<Ctrl+F7>
Component Parameters (Параметры компонентов)	<Option+F7>	<Alt+F7>
Library (Библиотека)	<F11>	<F11>

НАСТРОЙКА КЛАВИШ БЫСТРОГО ДОСТУПА

Во Flash MX клавиши быстрого доступа заданы по умолчанию, но их можно откорректировать в соответствии с собственными предпочтениями. На рис. 2.11 видно, что в программе имеется несколько наборов клавиш быстрого доступа, использующихся в наиболее популярных приложениях. Можете воспользоваться одним из этих наборов или создать свой собственный, выполнив команду **Edit⇒Keyboard Shortcuts (Правка⇒Клавиши быстрого доступа)**.

В программе предварительно заданы следующие наборы клавиш быстрого доступа: Macromedia Standard (Macromedia по умолчанию), Fireworks 3, Fireworks 4, Flash 5, FreeHand 10, FreeHand 9, Illustrator 10, Illustrator 9, Photoshop 5 и Photoshop 6. Если вы никогда раньше не работали с Flash MX, но являетесь опытным пользователем любого из этих приложений, то вам будет удобнее воспользоваться хорошо знакомым набором клавиш быстрого доступа.

Чтобы создать новый набор клавиш быстрого доступа, выберите один из уже имеющихся, наиболее похожий на тот, что предстоит создать. Затем щелкните на кнопке Duplicate Set (Дублировать набор), расположенной справа от раскрывающегося меню Current Set (Текущий набор), и задайте уникальное имя нового набора. Затем выберите один из элементов раскрывающегося меню Commands (Команды). При выборе элемента данного меню команды будут отображаться индивидуально, и их можно будет выбирать. Щелкните на команде, подлежащей изменению, в результате чего на экране будет отображена соответствующая комбинация клавиш быстрого доступа. Исходную комбинацию можно изменить полностью или просто добавить к ней какие-то клавиши. После редактирования комбинации клавиш щелкните на кнопке Change (Изменить). После того как новый набор будет полностью составлен, щелкните на кнопке OK.

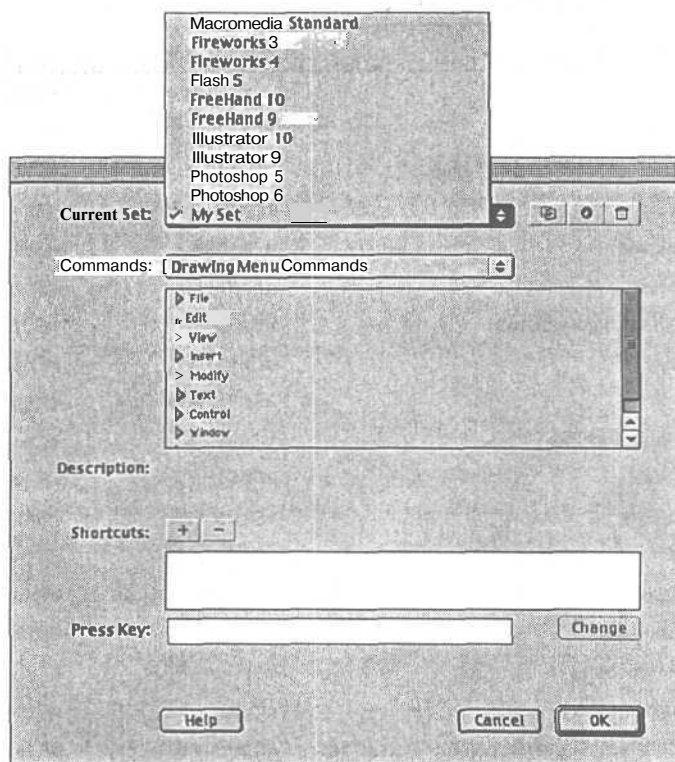


Рис. 2.11. Воспользуйтесь одним из имеющихся наборов клавиш быстрого доступа или создайте свой собственный

БИБЛИОТЕКА

Библиотека является местом размещения символов и импортированных элементов фильма. Библиотека представляет собой список содержимого, где перед каждым элементом размещена пиктограмма, указывающая на его тип. Элементы можно группировать в папки, каждую из которых можно разворачивать отдельно, что позволит минимизировать прокрутку экрана при просмотре содержимого библиотеки. Как и в случае слоев, количество элементов библиотеки быстро увеличивается по мере разработки проекта, и если их не систематизировать, наступит полная неразбериха.

СОЗДАНИЕ ОБЩИХ БИБЛИОТЕК

В программе Flash имеется возможность сохранения постоянных наборов библиотек посредством команды **Common Libraries** (Общие библиотеки). Создав общую библиотеку, предназначенную для работы над определенным проектом, можно быть уверенным в том, что все разработчики будут использовать одни и те же элементы.

Во Flash MX имеется несколько встроенных общих библиотек, содержащих образцы кнопок и звуковых файлов. Доступ к этим библиотекам можно получить посредством команды **Window ⇒ Common Libraries** (Окно ⇒ Общие библиотеки).

ПРОВОДНИК ФИЛЬМА

На панели Movie Explorer (Проводник фильма) представлена иерархическая структура каждого элемента Flash-фильма. Здесь можно отобразить все содержимое фильма покадрово, по каждому слою или сцене, как показано на рис. 2.12. Это позволяет, например, найти и заменить любой символ, имя экземпляра, строку кода или текста. Если вы не можете найти какой-то элемент, обязательно воспользуйтесь панелью Movie Explorer, и все проблемы будут быстро решены. Можете даже сделать распечатку, чтобы иметь наглядное представление о структуре фильма и его элементах.

Для открытия панели Movie Explorer выполните команду **Window⇒Movie Explorer**. Воспользовавшись кнопками-фильтрами, расположенными в верхней части окна, можно установить, какие элементы следует отображать на панели. С помощью первой кнопки можно указать, следует ли отображать текст, затем следуют кнопки для символов и сценариев, звуковых и видео-элементов, растровых изображений и кадров. Затем следует кнопка, позволяющая выполнять настройку отображаемых элементов. Сразу под этими кнопками располагается поле поиска.

Используйте описанные выше кнопки фильтрации для навигации по элементам фильма. Для выделения элемента в списке нужно дважды щелкнуть на нем. Если дважды щелкнуть на тексте, его можно будет отредактировать прямо на панели Movie Explorer. Меню параметров (его кнопка располагается в правом верхнем углу панели) позволяет переименовывать экземпляры и символы, редактировать символы прямо в рабочем поле или в новом окне, переходить к кадру, слою или сцене, в которых расположен выбранный элемент. После двойного щелчка на строке кода последний откроется в окне редактора ActionScript Editor.

ЗАДАНИЕ ПАРАМЕТРОВ FLASH

Для задания глобальных параметров среды проектирования предназначено диалоговое окно Preferences (Параметры), показанное на рис. 2.13, которое открывается после выполнения команды **Edit⇒Preferences (Правка⇒Параметры)**. Большинство задаваемых парамет-

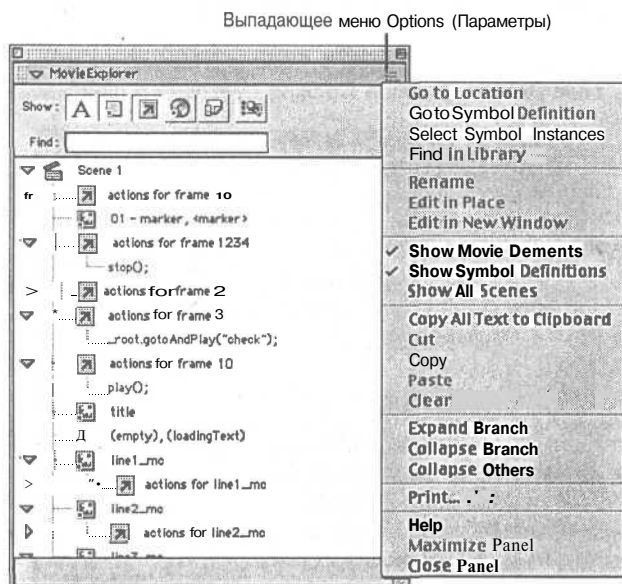


Рис. 2.12. На панели Movie Explorer отображено полное содержимое фильма

ров зависит исключительно от пристрастий и привычек самого пользователя. Возможно, задаваемые параметры будут зеркальным отображением настроек другого, хорошо знакомого вам приложения. Откорректируйте настройки программы так, чтобы создать наиболее удобную для себя среду проектирования.

ОБЩИЕ НАСТРОЙКИ

Первым параметром вкладки General Preferences (Общие настройки) (см. рис. 2.13) является Undo Levels (Уровни отмены). Он определяет число отмен и повтора команд, хранящихся в памяти во время каждого сеанса работы. Диапазон значений может быть от 0 до 200; по умолчанию задано значение 100. Чем больше число, тем больше памяти будет задействовано для хранения данных, необходимых для отмены или повтора выполненных команд. Поэтому необходимо поддерживать разумный баланс между числом отменяемых команд и ресурсами памяти, имеющимися в системе.

Далее пользователи Windows увидят параметр Printing Options (Параметры печати) с опцией Disable PostScript (Отключить язык PostScript), которая по умолчанию отключена. Если при печати файла на принтере, поддерживающем PostScript, у вас возникли проблемы, установите флажок в поле данной опции, но знайте, что процесс печати будет происходить медленнее. На компьютерах Macintosh параметр Printing Options отсутствует.

Затем следуют поля опции Selection Options (Параметры выделения): Shift Select (Выделение с помощью клавиши <Shift>) и Show Tooltips (Показывать всплывающие подсказки об инструментах). По умолчанию флажки установлены в полях обеих опций. Если установлен флажок в поле опции Shift Select, то после щелчка в рабочем поле можно будет выделить только один объект. Чтобы к выделенному фрагменту можно было добавить дополнительные объекты, при выделении необходимо удерживать нажатой клавишу <Shift>. Если снять флажок с поля опции Shift Select, то после щелчка в рабочей области к выделенному фрагменту будут автоматически добавляться другие объекты.

Если установлен флажок в поле опции Show Tooltips, то при наведении указателя мыши на тот или иной инструмент на экране будет отображаться контекстная информация о данном инструменте, в том числе и клавиши быстрого доступа к нему (если таковые имеются). Эта функция особенно полезна для тех, кто только начинает осваивать Flash MX.

Теперь рассмотрим опции группы Timeline Options (Параметры временной шкалы). Установление флажка в поле Disable Timeline Docking (Отключить привязку временной шкалы)

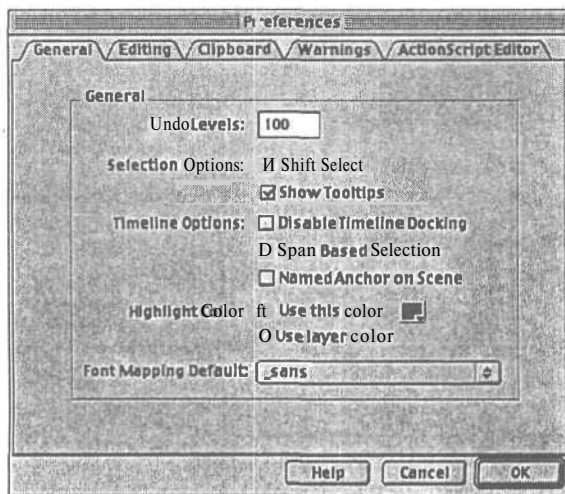


Рис. 2.13. Диалоговое окно Preferences предназначено для настройки глобальных параметров среды проектирования

препятствует привязке панели Timeline к рабочей области, в результате чего она всегда будет плавающей. Данной функцией можно воспользоваться для максимально возможного увеличения рабочего поля приограниченном пространстве экрана. По умолчанию флажок в поле опции Disable Timeline Docking не установлен.



Параметр Span Based Selection (Выделение в пределах диапазона) является новым способом выделения кадров на панели Timeline (Временная шкала) (рис. 2.14). Когда установлен флажок в поле данной опции, после щелчка на каком-либо кадре на панели Timeline будет выделен целый диапазон кадров, находящихся либо между ключевыми кадрами, либо между начальным и конечным кадрами (если ключевые кадры отсутствуют). Если ключевые кадры следуют друг за другом, после щелчка будет выделен только отдельный ключевой кадр. Для выделения отдельного кадра, не являющегося ключевым, необходимо щелкнуть на нем правой кнопкой мыши либо при щелчке удерживать нажатой клавишу <Cmd> (для Macintosh) или <Ctrl> (для Windows). Если флажок в поле опции Span Based Selection не установлен (по умолчанию), то после щелчка на кадре будет выделен только этот кадр. Таким образом производилось выделение кадров в программе Flash 5.



Опция Named Anchor on Scene (Именованный анкер на сцене) также является нововведением Rash MX. Теперь можно добавлять именованные анкеры так, чтобы после щелчка на кнопке Forward (Вперед) или Back (Назад) браузера можно было переходить от одной сцены фильма к другой. Анкеры можно разместить и для перехода от одного кадра к другому, но делать это придется вручную. Если в поле опции Named Anchor on Scene установлен флажок, программа автоматически помещает именованный анкер в первый кадр каждой сцены фильма. Мне кажется, что ценность данного параметра не слишком велика, поскольку многие дизайнеры и разработчики вообще не используют сцены, а те, кто это делают, преследуют чисто организационные цели.

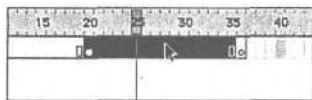


Рис. 2.14. Новый способ выделения кадров позволяет одним щелчком мышью выделить целый диапазон кадров

Параметр Highlight Color Options (Параметры цвета выделения) определяет цвет ограничительного контура, обрамляющего объекты и элементы при их выделении. Установка переключателя в поле опции Use This Color (Использовать данный цвет) позволяет выбрать любой Web-безопасный цвет из имеющейся цветовой палитры. Если установлен переключатель в поле опции Use Layer Color (Использовать цвет слоя), то при работе с программой цвет будет назначаться в соответствии со слоем, на котором располагается данный объект. При создании каждого слоя для него назначается цвет. Этот же цвет используется для заполнения цветного квадрата, который расположен справа от имени слоя. Установка переключателя в поле опции Use Layer Color позволяет быстро определить, в каком слое располагается объект.



Новой функцией Flash MX является автоматическая подсказка о замене недостающих шрифтов при открытии фильма, содержащего шрифты, не установленные в системе. Параметр Font Mapping Default (Отображение шрифтов по умолчанию) позволяет выбрать шрифт по умолчанию, который будет использоваться при замене недостающих шрифтов.

ПАРАМЕТРЫ РЕДАКТИРОВАНИЯ

Во вкладке Editing Preferences (Параметры редактирования) сосредоточены параметры настройки рисования, инструмента Реп (Перо), а также параметры расположения текста по вертикали. Все параметры данной вкладки по умолчанию отключены.

В группу опций Pen Tools (Инструменты пера) входит параметр Show Pen Preview (Предварительный просмотр пера), позволяющий предварительно просматривать сегменты линий, созданные с помощью инструмента Реп, в процессе их рисования, до того, как будет создана конечная точка. Если в поле данной опции флажок не установлен, то создаваемые сегменты нельзя будет увидеть до тех пор, пока не будет создана конечная точка.

При установке флажка в поле опции Show Solid Points (Показывать сплошные точки) невыделенные анкерные точки будут отображаться сплошными, а выделенные анкерные точки — пустыми. Если флажок в поле данной опции не установлен (по умолчанию), выделенные анкерные точки будут отображаться сплошными, а невыделенные точки — пустыми.

На заметку

Чтобы получить возможность работать с именованными анкерами, откройте диалоговое окно Publish Settings (Параметры публикации), перейдите во вкладку HTML и из раскрывающегося меню Template (Шаблон) выберите пункт Flash with Named Anchors (Flash с именованными анкерами).

Если установлен флажок в поле опции Show Precise Cursors (Показывать точные курсоры), то в рабочем поле инструмент Реп будет иметь вид перекрестия, что облегчает процедуру точного размещения анкерных точек. По умолчанию при использовании инструмента Реп отображается его пиктограмма. Чтобы в процессе рисования переключаться между этими настройками, воспользуйтесь клавишей <Caps Lock>

В программе Flash MX текст можно располагать как по горизонтали, так и по вертикали. Для этого достаточно щелкнуть на кнопке Text Direction (Направление текста), расположенной на панели инспектора свойств. Пользователи, работающие с азиатскими языками, могут вздохнуть с облегчением. Теперь по умолчанию текст можно располагать по вертикали, установив флажок в поле опции Default Text Orientation (Ориентация текста по умолчанию), расположенной в группе Vertical Text (Расположение текста по вертикали). Помимо этого, можно воспользоваться опциями Right to Left Text Flow (Написание текста справа налево) и No Kerning (Без кернинга). Даже если установлен флажок в поле опции No Kerning, вы все равно сможете задавать межбуквенный интервал горизонтального текста на панели инспектора свойств.

Параметры категории Drawing Settings (Параметры рисования) позволяют установить допуск и указать уровень привязки, сглаживания и выпрямления объектов. По умолчанию для всех параметров выбрано значение Normal (Обычное).

Параметр Connect Lines (Соединять линии) определяет, насколько близко может располагаться нарисованная конечная точка к другой линии, чтобы еще не быть привязанной к ней. Кроме того, он определяет чувствительность распознавания горизонтальной и вертикальной линий, т.е. насколько точно по вертикали или по горизонтали должна быть нарисована линия для того, чтобы было выполнено соответствующее выравнивание. При включенной опции Snap to Objects (Привязка к объектам) данный параметр определяет, насколько близко друг от друга могут находиться объекты, чтобы еще не быть привязанными друг к другу. Данный параметр может принимать значения Must Be Close (Должны быть рядом), Normal (Обычно) и Can Be Distant (Могут отстоять).

При работе в режиме Straighten (Спрямление) или Smooth (Сглаживание) параметр Smooth Curves (Гладкие кривые) указывает степень применения сглаживания или выпрямления к кривым или к прямым линиям. При сглаживании из кривых удаляются точки, что приводит к созданию более плавных дуг, а в режиме спрямления кривые выравниваются. Данный параметр может иметь значения Off (Отключить), Strict (Строго), Normal (Обычно), Tolerant (Допустимо).

Параметр Recognize Lines (Распознавание линий) определяет пороговое значение, при котором программа распознает как линии неровные сегменты, нарисованные с помощью инструмента Pencil (Карандаш), и преобразует их в строгие прямые. Данный параметр может иметь значения Off (Отключить), Strict (Строго), Normal (Обычно), Tolerant (Допустимо).

Параметр Recognize Shapes (Распознавание форм) определяет, насколько точно должна быть нарисована форма, чтобы ее можно было распознать как геометрическую фигуру, такую как круг, овал, квадрат, прямоугольник или дуга. Данный параметр может иметь значения Off (Отключить), Strict (Строго), Normal (Обычно), Tolerant (Допустимо).

Параметр Click Accuracy (Точность щелчка) определяет пороговое значение того, насколько близко к объекту должна располагаться мышь, чтобы объект мог быть выделен. Данный параметр может иметь значения Strict (Строго), Normal (Обычно) и Tolerant (Допустимо).

ПАРАМЕТРЫ БУФЕРА ОБМЕНА

Параметры вкладки Clipboard Preferences (Параметры буфера обмена) определяют, каким образом объекты копируются в буфер обмена. Параметры этой вкладки, за исключением FreeHand Text (Текст FreeHand), зависят от используемой платформы.

В распоряжении пользователей Windows имеются группы опций Bitmaps (Растровые изображения) и Gradients (Градиентное заполнение). В группе Bitmaps можно указать глубину цвета (Color Depth) и разрешение (Resolution), применить сглаживание контурных неровностей (установив флажок в поле опции Smooth) и указать предельный объем памяти, допустимый при копировании растровых изображений в буфер обмена (опция Size Limit). Если объем памяти ограничен, можно уменьшить значение, по умолчанию установленное в поле Size Limit (Ограничить объем).

Параметр Gradients (Градиентное заполнение) позволяет указать качество градиентного заполнения, копируемого в буфер обмена и вставляемого в другие приложения. Данный параметр не влияет на качество градиентного заполнения, вставляемого в программу Flash, поскольку оно всегда является максимальным.

Пользователи Macintosh могут воспользоваться группой параметров PICT Settings (Параметры изображения), в состав которой входят опции Type (Тип), Resolution (Разрешение) и Gradients (Градиентное заполнение). Параметр Type определяет формат копирования графики. Данный параметр может принимать значения Objects (для работы с векторной графикой) или Bitmaps (Растровые изображения) с указанием диапазона битовой глубины. Параметр Resolution (Разрешение) определяет число точек на дюйм (dpi) в копируемых изображениях. Здесь же имеется поле опции, предназначенное для включения данных PostScript. Параметр Gradients указывает качество градиентного заполнения графики, копируемой и вставляемой не в программу Flash. По умолчанию для данного параметра задано значение Normal (Обычное), но можно выбрать и значения None (Нет), Fast (Быстрое) и Best (Наилучшее). Чем выше параметр качества, тем дольше будет копироваться графика. Как и аналогичный параметр для Windows, данный параметр не влияет на качество градиентного заполнения, вставляемого во Flash, поскольку здесь всегда поддерживается максимальное качество.

Для параметра FreeHand Text имеется только поле опции Maintain Text as Blocks (Поддерживать текст в виде блоков). Установив флажок в поле данной опции, вы сможете вставить текст из программы FreeHand и сохранить его доступным для редактирования.

НАСТРОЙКА ПРЕДУПРЕЖДАЮЩИХ СООБЩЕНИЙ

Во вкладке Warning Preferences (Параметры предупреждающих сообщений) можно указать, когда следует выводить на экран те или иные предупреждающие сообщения. По умолчанию включены все опции.

- Warn on Save for Macromedia Flash 5 Compatibility (Предупреждать о совместимости при сохранении в формате Macromedia Flash 5). Предупреждающее сообщение будет выводиться при попытке сохранить файлы с содержимым Flash MX в формате Flash 5.
- Warn on Missing Fonts (Предупреждать об отсутствующих шрифтах). Предупреждающее сообщение будет выводиться при попытке открытия файла, содержащего шрифты, не установленные в вашей системе.

- Warn on Loss of Expert Mode Formatting (Предупреждать о потере форматирования режима Expert). Предупреждающее сообщение будет выводиться при попытке перейти из режима Expert в режим Normal на панели Actions с потерей форматирования, примененного в режиме Expert.
- Warn on Reading Generator Content (Предупреждать при считывании содержимого программы Generator). Поверх любого содержимого программы Generator будет отображен символ X красного цвета. Помните, что объекты программы Generator больше не поддерживаются.
- Warn on Inserting Frames When Importing Content (Предупреждать о вставке кадров при импортировании содержимого). Предупреждающее сообщение будет выводиться в случае необходимости расширения панели Timeline, чтобы она могла вместить импортируемое содержимое.

ПАРАМЕТРЫ РЕДАКТОРА ACTIONSCRIPT

Доступ к параметрам редактора ActionScript можно также получить с панели Actions (Действия), выбрав из всплывающего меню пункт Preferences (Параметры).

Группа опций Editing Options (Параметры редактирования) позволяет настраивать параметры форматирования при работе с панелью Actions. При работе в режиме Expert установка флажка в поле опции Automatic Indentation (Автоматический отступ) устанавливает отступ кода. Для задания величины отступа в поле Tab Size (Величина отступа) вводится число от 1 до 4 (задано по умолчанию). При установке флажка в поле опции Code Hints (Советы по использованию кода) по мере ввода кода как в режиме Normal, так и в режиме Expert на экран будет выводиться панель с подсказками. Перемещение ползунка Delay (Задержка) позволяет отрегулировать временной промежуток, по истечении которого будет открываться панель Code Hints. По умолчанию ползунок установлен в значение 0.

Параметры группы Text (Текст) позволяют указать тип и размер шрифта, отображаемого на панели Actions.

Параметры группы Syntax Coloring (Выделение синтаксиса цветом) предназначены для задания цвета текста синтаксических элементов, таких как ключевые слова (Keywords), идентификаторы (Identifiers), комментарии (Comments) и строки (Strings), переднего (Foreground) и заднего плана (Background).

Все параметры редактора ActionScript имеют значения, заданные по умолчанию. Чтобы вернуться к этим настройкам, щелкните на кнопке Reset to Defaults (Восстановить значения по умолчанию), расположенной в нижней части окна Preferences.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Я пробовал работать с параметрами и расположением панелей и случайно скрыл нужную панель. Помогите!

Не паникуйте. Панели легко восстановить на экране. Для этого есть несколько способов.

- Откройте меню Window (Окно) и выберите панель, которую следует отобразить.
- Воспользуйтесь комбинацией клавиш, предназначенной для отображения нужной панели.
- Если неожиданно исчезли все панели, скорее всего, вы случайно нажали клавишу <Tab>. Нажмите ее еще раз, и панели снова появятся на экране.
- Набор панелей, отображаемый по умолчанию, всегда можно восстановить, выполнив команду Window⇒Panel Sets⇒Default Layout (Окно⇒Наборы панелей⇒Размещение по умолчанию).

FLASH ЗА РАБОТОЙ: СОЗДАНИЕ ИДЕАЛЬНОГО ИНТЕРФЕЙСА FLASH MX

После того как вы немного поработаете с Flash, вы будете знать, какие наборы панелей наилучшим образом подходят для вашей работы. Запомните, какие комбинации панелей вы используете для выполнения тех или иных задач, и сохраните их как отдельные наборы. Потратив совсем немного времени на создание нужных наборов панелей, вы в дальнейшем существенно увеличите эффективность своей работы.

При работе с Flash я пользуюсь набором панелей для рисования и создания сценариев. В набор для рисования, изображенный на рис. 2.15, входят панели Timeline (Временная шкала), Property Inspector (Инспектор свойств), Align (Выравнивание), Color Mixer (Цветовой микшер), Color Swatches (Каталоги цветов), Components (Компоненты) и Library (Библиотека). Имейте в виду, что размер рабочего поля можно увеличить, а привязанные панели можно разворачивать по мере необходимости, что позволит увеличить размер рабочей области.

В набор для написания сценариев входят панели Timeline (Временная шкала), ActionScript Editor (Редактор ActionScript), Property Inspector (Инспектор свойств), Components (Компоненты), Component Parameters (Параметры компонентов) и Movie Explorer (Проводник фильма).

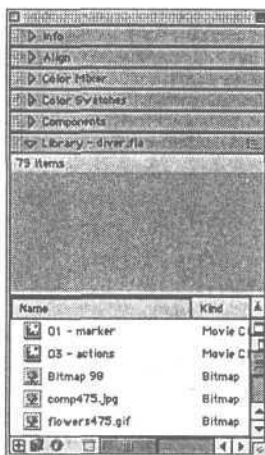


Рис. 2.15. Пользовательский набор панелей для рисования

РИСОВАНИЕ во FLASH

В ЭТОЙ ГЛАВЕ...

Знакомство с векторной графикой	61
Использование инструментов рисования	62
Редактирование и корректирование изображений	68
Использование заливки	72
Стирание	75
Использование вспомогательных элементов компоновки	76
Создание масок	78
Возможные проблемы	79
Flash за работой: насколько большим получится изображение	79

ЗНАКОМСТВО с ВЕКТОРНОЙ ГРАФИКОЙ

Технология Flash изначально была разработана как анимационное приложение, предназначенное для преобразования старой, пошаговой анимации в формат, пригодный для размещения в Web. В основе Flash всегда лежала процедура рисования. Несмотря на то что в настоящее время появились гораздо более современные технологии, такие как сценарии и комплексные взаимодействия, именно рисование является той базовой точкой, с которой начинается знакомство с Flash.

Компьютеры отображают графику в двух форматах: векторном и растровом. Растровые изображения состоят из дискретных элементов, называемых *пикселями*. Каждому пикселю соответствует определенное значение цвета. Будучи объединенными, разные цвета пикселей создают форму, являющуюся изображением. В растровом изображении представлена инфор-

мация о его размерах и цвете каждого пикселя, содержащемся в нем. Передача изображения осуществляется попиксельно. Растровые изображения превосходно подходят для отображения мельчайших цветовых оттенков, например в фотографиях. Но как быть, если возникнет необходимость изменить размер растрового изображения? Этого можно достичь двумя способами: либо полностью переделать изображение и воссоздать его с нужными размерами, либо растянуть его, но это обычно приводит к нежелательным результатам.

Наряду с растровой графикой существует еще и векторная, в которой изображения создаются с помощью линий и кривых посредством ряда сохраненных команд. Эти команды, именуемые *векторами*, задают атрибуты линий и кривых, в том числе их толщину, направление, цвет и расположение, которые затем обрабатываются компьютером. Векторная графика позволяет гораздо сильнее детализировать изображение и изменять его размеры без потери четкости. Процедура редактирования векторной графики заключается в изменении атрибутов линий и кривых. Векторная графика лучше всего подходит для отображения простых изображений со сплошными цветовыми участками, таких как пиктограммы, логотипы и рисованная анимация. И векторные, и растровые изображения создаются по заказу клиента, но файлы векторных изображений имеют меньшие размеры, и создать их можно быстрее. Растровые изображения передаются попиксельно, т.е. размер файла и время его загрузки непосредственно привязаны к размерам изображения. В векторной графике осуществляется передача инструкций, которые затем обрабатываются процессором, т.е. размер файла и скорость отображения определяются сложностью переданных инструкций, а не размером графического файла.

Поскольку Rash была создана как анимационная программа, нет ничего удивительного в том, что она работает с векторными изображениями. В программе могут быть отображены и растровые изображения, но все же она предназначена для работы именно с векторной графикой.

ИСПОЛЬЗОВАНИЕ ИНСТРУМЕНТОВ РИСОВАНИЯ

Существует несколько основных концепций понимания принципов создания графики в программе Flash. Нарисованные во Flash изображения состоят из штрихов и заливки (или того и другого). Штрихи являются контуром, внутри которого располагается заливка. Даже в изображениях, содержащих оба указанных компонента, штрихи и заливка не зависят друг от друга. Это позволяет задавать их атрибуты индивидуально и даже перемещать их независимо друг от друга, в результате чего создаются достаточно необычные эффекты.

Изображения, наложенные друг на друга в пределах одного слоя, взаимодействуют. Изображение, нарисованное поверх другого, заменяет собой те участки исходного изображения, которые оно закрывает. При соприкосновении изображения одного цвета сливаются друг с другом, а изображения разных цветов остаются отдельными, даже несмотря на то, что их наложенные участки заменяются. Линии, нарисованные с помощью инструментов Pencil (Карандаш), Line (Линия), Brush (Кисть), Oval (Овал) или Rectangle (Прямоугольник), разбиваются на сегменты, которые могут пересекать другие изображения или разрезать изображения, расположенные под ними.

Совет

Подробная информация обо всех инструментах представлена в главе 2 "Интерфейс Flash".

Этим необычным поведением можно воспользоваться для создания интересных негативных изображений. Чтобы запретить взаимодействие изображений, сгруппируйте их (выделив и выполнив команду **Modify⇒Group** (Изменить⇒Группировать)) или поместите их в отдельные слои.

При рисовании цвета штриха и заливки будут такими, как указаны в полях Stroke (Штрих) и Fill (Заливка) на панели инструментов, или на панели инспектора свойств при работе с определенным инструментом, или на панели Color Mixer (Цветовой микшер). Цвет, за-

данный на одной из этих панелей, будет указан и на остальных панелях. Любое нарисованное изображение имеет контур, но заливка может применяться только к изображениям, имеющим замкнутые внутренние области.

ИНСТРУМЕНТ LINE

Line (Линия) является одним из основных инструментов рисования. С его помощью можно нарисовать прямые линии или штрихи. Выберите инструмент Line, а затем щелкните в рабочем поле и перетащите курсор. В результате будет нарисована прямая линия. Удерживая при перетаскивании клавишу <Shift>, можно расположить линии строго по горизонтали, по вертикали или по диагонали.

Инструмент Line можно использовать и для создания замкнутых фигур, таких как квадраты или прямоугольники, к которым можно применить заливку. Однако в этом случае заливку придется добавлять вручную, с помощью инструмента Paint Bucket (Заливка), после того, как форма будет замкнута.

ИНСТРУМЕНТ PENCIL

Инструмент Pencil (Карандаш) позволяет создавать линии и изображения неправильной формы. Выберите инструмент Pencil, а затем в подразделе Options (Параметры) панели инструментов выберите один из режимов карандаша, позволяющих в процессе рисования спрямлять линии, сглаживать или оставлять их шероховатыми (с этой целью применяется модификатор Ink (Чернила)). После выбора инструмента щелкните в рабочем поле и перетащите курсор. В результате будет нарисована линия. Удерживая клавишу <Shift>, можно располагать линии строго по горизонтали или по вертикали. Как и при работе с инструментом Line, с помощью инструмента Pencil можно создавать замкнутые фигуры, но для их заливки придется отдельно применять инструмент Paint Bucket.

ИНСТРУМЕНТЫ OVAL и RECTANGLE

Инструменты Oval (Овал) и Rectangle (Прямоугольник) позволяют легко нарисовать простые фигуры. Для создания овалов и прямоугольников щелкните соответствующим инструментом и перетащите курсор. Для создания идеальных квадратов и окружностей при перетаскивании удерживайте нажатой клавишу <Shift>.

С помощью данных инструментов можно создать прямоугольники со скругленными углами. Задача эта весьма кропотливая, но при наличии терпения можно добиться впечатляющих результатов. Сначала выберите инструмент Rectangle, а затем в нижней части панели инструментов выберите модификатор Round Rectangle Radius (Радиус закругления углов прямоугольника). В поле Corner Radius (Радиус закругления) введите числовое значение. Нулевое значение соответствует прямым углам. Оптимальный радиус закругления обычно выбирается методом проб и ошибок.

ИНСТРУМЕНТ PEN

Инструмент Pen (Перо) является самым мощным средством рисования. Тем, кто раньше никогда не работал с графическими программами, надо привыкнуть к этому инструменту, особенно к тому, как он рисует кривые. Вам придется потратить некоторое время, чтобы научиться использовать всю мощь инструмента, но в результате вы будете рисовать такие кривые и изображения неправильной формы, которые невозможно создать никаким другим инструментом.

Процедура рисования инструментом Pen заключается в создании анкерных точек и их последующем соединении. Выберите инструмент Pen и перетащите указатель мыши по рабочему полю. Обратите внимание на то, что справа от пера располагается маленький символ х (рис. 3.1), сигнализирующий о размещении первой анкерной точки.



Рис. 3.1. При использовании инструмента Реп справа от пера располагается символ х, свидетельствующий о задании первой анкерной точки изображения

СОЗДАНИЕ ПРЯМЫХ ОТРЕЗКОВ

Для создания отрезков выберите инструмент Реп и разместите первую анкерную точку. Она будет иметь вид кружочка, который превратится в синий квадрат после создания других анкерных точек. Если при работе с инструментом Реп удерживать клавишу <Shift>, будут нарисованы горизонтальные, вертикальные или расположенные по диагонали (т.е. под углом 45°) линии. Процесс задания анкерных точек необходимо закончить, сообщив тем самым Раш о том, что траектория замкнутой или незамкнутой формы завершена. Для завершения разомкнутой траектории либо дважды щелкните в конечной анкерной точке, либо щелкните вне траектории, удерживая нажатой клавишу <Cmd> (Macintosh) или <Ctrl> (Windows), либо щелкните на инструменте Реп панели инструментов. Для завершения замкнутой траектории удерживайте инструмент Реп над первой анкерной точкой. На рис. 3.2 показано, что справа от пера появится маленький кружок, свидетельствующий о том, что вы замыкаете траекторию.

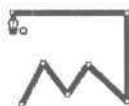


Рис. 3.2. Если инструмент Реп правильно размещен при замыкании траектории, то справа от него появится маленький кружок

СОЗДАНИЕ КРИВОЛИНЕЙНЫХ СЕГМЕНТОВ

Истинная мощь инструмента Реп заключается в возможности создания математически точных криволинейных сегментов, которые называются *кривыми Безье*. Чтобы точно описать дугу, необходимо задать четыре основных элемента: две анкерных точки и два управляющих маркера (рис. 3.3). Чтобы нарисовать кривую с помощью инструмента Реп, щелкните им в рабочем поле и перетащите анкерные точки в том направлении, в котором хотите нарисовать кривую. После первого щелчка и перетаскивания анкерной точки вы увидите, что она имеет управляющий маркер. Щелкните инструментом для создания второй анкерной точки и перетащите курсор в противоположном направлении, в результате чего будет нарисована дуга.



Рис. 3.3. Две анкерных точки и два управляющих маркера

Если перетащить второй анкер в том же направлении, что и первый, будет нарисована зигзагообразная кривая (рис. 3.4).



Рис. 3.4. Перетаскивание анкерных точек в одном направлении приводит к созданию зигзагообразной кривой

Длина и угол управляющих маркеров определяют форму кривой. Лучше всего завершить создание траектории перед корректировкой кривых. Чтобы мастерски рисовать кривые Безье, необходимо затратить очень много времени, и только высочайшие профессионалы могут сразу в процессе рисования получить именно такую кривую, как им нужно. Если вы никогда не занимались этим, сначала нарисуйте приблизительную форму кривой, а затем займитесь ее корректировкой. Чтобы замкнуть криволинейную траекторию, щелкните в начальной анкерной точке и перетащите курсор в сторону от кривой, как показано на рис. 3.5.

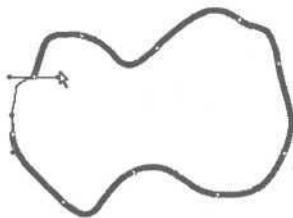


Рис. 3.5. Чтобы замкнуть криволинейную траекторию, щелкните в начальной анкерной точке и перетащите курсор в сторону от кривой

КОРРЕКТИРОВКА АНКЕРНЫХ ТОЧЕК

Самый быстрый способ рисования инструментом Реп заключается в создании траектории и последующей корректировке анкерных точек. Анкерные точки криволинейных траекторий имеют управляющие маркеры, а анкерные точки прямых отрезков являются угловыми точками. После создания формы вы можете добавить или удалить анкерные и угловые точки, преобразовать анкерные точки в угловые и наоборот, а также переместить любые существующие точки.

Чтобы откорректировать анкерные точки, сначала их нужно выделить. Место размещения анкерных точек в криволинейных и неоднородных контурах может быть далеко не очевидным. Для отображения анкерных точек выберите инструмент Subselection (Частичное выделение) и щелкните на контуре (рис. 3.6).

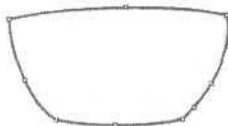


Рис. 3.6. Для отображения анкерных точек щелкните на контуре инструментом Subselection

Если вы щелкнете прямо на анкерной точке, то выделите ее и одновременно отобразите все остальные анкерные точки контура. Щелчок на контуре вне анкерной точки приведет к отображению анкерных точек, а для выделения точки нужно будет щелкнуть непосредственно на ней. Чтобы добавить к выделенному контуру дополнительные анкерные точки, при щелчке удерживайте нажатой клавишу <Shift>. Для перемещения анкерных точек щелкните и перетащите курсор, как показано на рис. 3.7. Чтобы лишь слегка передвинуть анкерную точку, выделите ее и воспользуйтесь клавишами со стрелками.

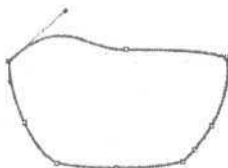


Рис. 3.7. Для изменения формы кривой щелкните и перетащите управляющий маркер

При редактировании контура может возникнуть необходимость преобразовать угловую точку в точку кривой в целях добавления кривых или выполнить обратное преобразование для выпрямления контура. Чтобы преобразовать угловую точку в точку кривой, создайте управляющий маркер, удерживая при перетаскивании нажатой клавишу <Option> (Macintosh) или <Alt> (Windows). Перетащите управляющий маркер в направлении необходимого искривления. Чтобы преобразовать точку кривой в угловую, сначала отобразите анкерные точки с помощью инструмента Subselection. Затем выберите инструмент **Pen** и щелкните на точке кривой, в результате чего она будет преобразована в угловую точку.

В процессе редактирования контура может возникнуть необходимость его дальнейшей детализации или, наоборот, упрощения. Дополнительные точки позволяют детализировать контур, добавляя к нему кривые или углы. Для добавления анкерной точки щелкните на контуре инструментом **Pen**. Для упрощения контура следует удалить анкерные точки. Для этого щелкните на точке инструментом Subselection и нажмите клавишу <Delete>. Кроме того, можно выбрать инструмент **Pen** и щелкнуть либо один раз для удаления угловой точки, либо дважды — для удаления точки кривой.

КОРРЕКТИРОВКА СЕГМЕНТОВ

Сегменты контура можно редактировать независимо от их анкерных точек. К примеру, может возникнуть необходимость откорректировать небольшой участок кривой. Чтобы откорректировать размеры и угол кривизны, щелкните и перетащите управляющие маркеры. Если пересекаются два криволинейных сегмента, два управляющих маркера будут направлены в стороны от общей анкерной точки. Каждый маркер управляет искривлением со своей стороны относительно общей анкерной точки. Перетаскивание одного конца двунаправленного маркера изменит обе кривые с обеих сторон от анкерной точки, которая останется на своем месте, как показано на рис. 3.8.



Рис. 3.8. Перетаскивание одного конца двунаправленного маркера изменит обе кривые с обеих сторон от общей анкерной точки

Чтобы откорректировать дугу криволинейного участка между двумя анкерными точками, перетащите один конец двунаправленного маркера, удерживая при этом нажатой клавишу <Option> (Macintosh) или <Alt> (Windows).

Если криволинейный отрезок пересекается с прямолинейным, управляющий маркер будет расположен с криволинейной стороны анкера. Для изменения дуги щелкните на маркере инструментом Subselection и перетащите его, а для **перемещения** кривой щелкните и перетащите анкерную точку.

Для перемещения сегментов можно воспользоваться инструментом Arrow (Стрелка). Если инструмент Arrow расположить над контуром, то справа от стрелки появится маленький уголок или кривая, **указывающие**, какой тип сегмента может быть выделен. При выделении криволинейного сегмента справа от стрелки появится маленькая кривая линия (рис. 3.9).



Рис. 3.9. Если инструмент Arrow расположить над криволинейным сегментом, то рядом с указателем появится маленькая дуга

При выделении отрезка прямой рядом со стрелкой появится правый уголок, как показано на рис. 3.10. Чтобы переместить сегмент, щелкните и перетащите его.



Рис. 3.10. Если инструмент Arrow расположить над отрезком прямой, то рядом с указателем появится маленький уголок

ИНСТРУМЕНТ BRUSH

Инструмент Brush (Кисть) заслуживает особого **внимания**, поскольку он является единственным средством, позволяющим имитировать мазки кистью, как показано на рис. 3.11. Чтобы получить наиболее реалистичные мазки, необходим специальный чувствительный к нажиму графический планшет, при работе с которым вместо мыши используется графическое перо. Попытка рисовать с помощью мыши сродни попытке подписать квитанцию куском мыла. Планшеты реагируют на изменение давления при рисовании и соответствующим образом изменяют толщину штриха.

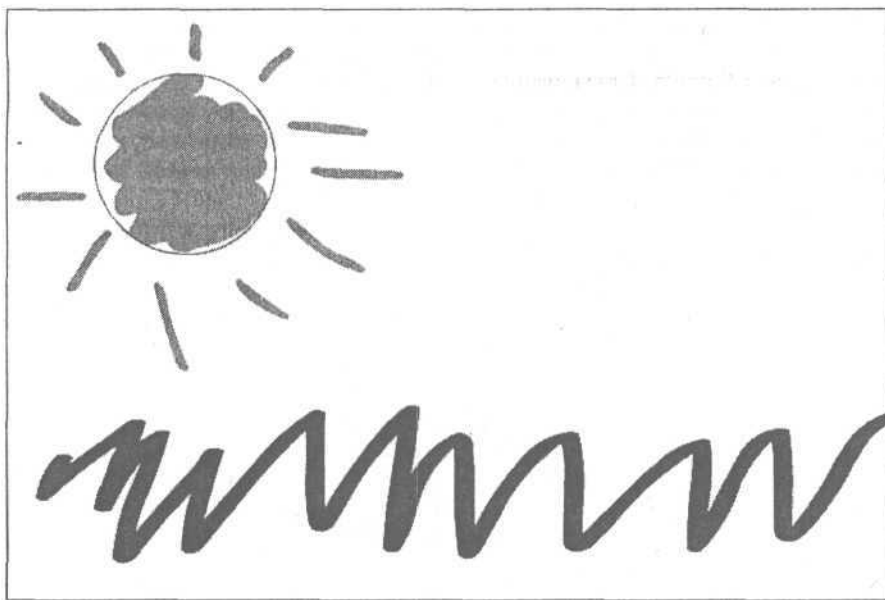


Рис. 3.11. С помощью инструмента *Brush* создаются штрихи неправильной формы, похожие на мазки кистью

Для корректировки размера и формы кисти выберите один из модификаторов, расположенных в разделе *Options* панели инструментов. Выбрав нужный режим кисти, можно выполнить окраску обычным образом, создать только заливку, рисовать на заднем плане относительно объектов, только внутри объектов или внутри выделенного контура. Комбинируя мазки кистью с точно нарисованными формами, можно создать очень интересную текстуру изображения.

РЕДАКТИРОВАНИЕ и КОРРЕКТИРОВАНИЕ ИЗОБРАЖЕНИЙ

Как уже говорилось, в процессе рисования, особенно с помощью мыши, тяжело сразу же создать идеальное изображение. Лучше всего (особенно для новичков) создать приближительную форму, а затем довести ее до совершенства путем редактирования. Редактировать существующие формы гораздо легче, чем создавать их с нуля.

ИСПОЛЬЗОВАНИЕ ИНСТРУМЕНТОВ *ARROW* и *SUBSELECTION*

Вы уже знаете, что инструменты *Arrow* и *Subselection* можно использовать для редактирования контуров. Просто щелкните и перетяните нужный сегмент или точку. При перетаскивании анкерной точки изменится место ее расположения, а также размер и форма прилегающих сегментов. При перетаскивании сегмента изменится его форма, а анкерные точки по обе стороны от сегмента останутся на своих местах.

ВЫПРЯМЛЕНИЕ и СГЛАЖИВАНИЕ

Контуры и изображения, имеющие неправильную форму, можно как выпрямлять, так и сглаживать. Предположим, что с помощью инструмента *Pencil* вы нарисовали несколько кривых произвольной формы, но некоторые из них оказались слишком сильно изогнутыми

или, наоборот, слишком пологими. Чтобы откорректировать кривую, выделите ее с помощью инструмента Arrow, а затем на панели инструментов выберите модификатор Straighten (Выпрямление) или Smooth (Сглаживание). Чтобы добиться наилучших результатов, попробуйте поочередно применить модификаторы. Нет ничего удивительного в том, что модификатор Straighten не оказывает влияния на прямые линии или фигуры с прямыми сторонами. Аналогичным образом, модификатор Smooth нельзя применить к овалам и окружностям, хотя с его помощью все-таки можно немного изменить кривые линии.

ОПТИМИЗАЦИЯ КРИВЫХ

Чем больше анкерных точек содержат нарисованные кривые, тем больше вычислений необходимо выполнить для их создания. В предельных случаях это может привести к увеличению размеров файлов и замедлению отображения графики. Чтобы уменьшить число анкерных точек, кривые можно оптимизировать. Это приведет к тому, что кривые станут более сглаженными и изящными. Делается это для того, чтобы для создания нужного изображения использовалось необходимое и достаточное количество анкерных точек.

Для оптимизации кривой выделите контур и выполните команду **Modify⇒Optimize** (Изменить⇒Оптимизировать), откроется диалоговое окно Optimize Curves (Оптимизировать кривые), изображенное на рис. 3.12. В этом окне имеются ползунок Smoothing (Сглаживание), а также параметры использования нескольких проходов и отображения итогового сообщения. Задайте необходимые параметры и щелкните на кнопке ОК. Сглаживание уменьшает по сравнению с исходной кривой число анкерных точек. Чтобы значительно уменьшить число анкерных точек, установите флажок в поле опции Use multiple passes (Пройти несколько раз). Это приведет к выполнению нескольких операций сглаживания, при каждой из которых будут удаляться анкерные точки. Операции сглаживания будут выполняться до тех пор, пока дальнейшая оптимизация будет возможна. При установлении флажка в поле опции Show totals message (Показать итоговое сообщение) сравниваются исходное и оптимизированное число кривых с указанием степени оптимизации.

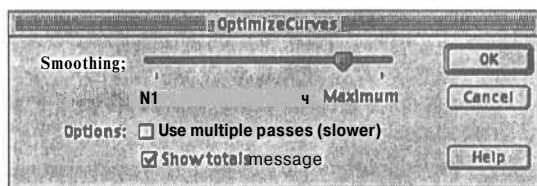


Рис. 3.12. Кривые можно сгладить в диалоговом окне Optimize Curves

РАБОТА со ШТРИХАМИ

Штрихом называется линия или контур изображения. Штрихи имеют три атрибута: цвет, толщину и стиль. Цвет штриха можно задать в поле Stroke (Штрих), на панели инспектора свойств или на панели Color Mixer. Толщину и стиль штриха можно задать только на панели инспектора свойств. Чтобы задать все атрибуты сразу, лучше всего воспользоваться панелью инспектора свойств, однако на ней можно выбрать только безопасные Web-цвета.

Палитра безопасных Web-цветов была разработана четыре или пять лет назад при попытке создания **цветов**, которые бы одинаково отображались на всех компьютерах. Одним из факторов несовместимости содержимого Web является то, что различные операционные системы и браузеры отображают конечные наборы цветов. Проблема заключается в том, что каждый браузер и операционная система работают с набором цветов, несколько отличающимся от того, который

поддерживается другой системой. Поэтому была разработана палитра безопасных Web-цветов, содержащая 216 цветов, общих для двух основных браузеров (Internet Explorer и Netscape Navigator) и двух основных операционных систем, Windows и Macintosh. Эти цвета имеются в каталоге, поддерживаемом программой Flash. Однако некоторые компьютеры имеют расширенные возможности отображения цветов, поэтому было принято соглашение о том, что можно пользоваться и цветами, не входящими в палитру безопасных Web-цветов. Кроме того, 216 цветов — это очень ограниченная палитра для дизайнерских разработок. В программе Flash имеется панель Color Mixer (Цветовой микшер), позволяющая создавать свои собственные цвета и оттенки, а также изменять коэффициент прозрачности цветов.

Панель инспектора свойств используется для назначения и редактирования атрибутов штриха, в том числе и цвета. Для создания штриха выполните следующие действия.

1. Выберите подходящий инструмент рисования.
2. На контекстно-зависимой панели инспектора свойств будут отображены соответствующие атрибуты (в данном случае — это атрибуты штриха, как показано на рис. 3.13).

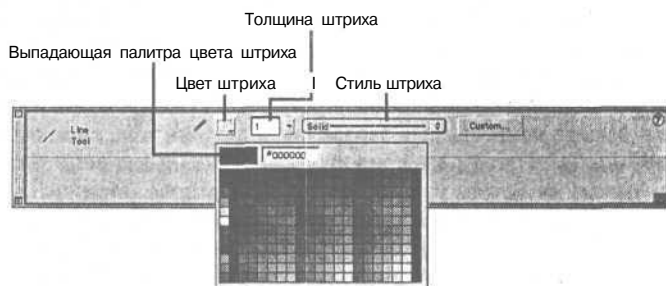


Рис. 3.13. Атрибуты штриха на панели инспектора свойств

3. Щелкните в каталоге цветов и из открывшейся палитры выберите нужный цвет.
4. Чтобы задать толщину штриха, в поле толщины штриха введите число либо воспользуйтесь ползунком.
5. Толщина штриха умножается с шагом 0,25 и может лежать в диапазоне от 0,25 до 10. Чем больше число, тем толще штрих.
6. Задайте стиль штриха. Рядом с полем толщины штриха расположено раскрывающееся меню, из которого можно выбрать один из наиболее распространенных стилей штриха (рис. 3.14). Стиль Hairline (Волосая линия) соответствует самому тонкому штриху, толщиной 0,25. При этом ранее заданная толщина штриха не принимается во внимание. Штрихи всех остальных стилей могут быть любой толщины.

Для создания собственного стиля штриха щелкните на кнопке Custom (Настройка), в результате чего откроется диалоговое окно Stroke Style (Стиль штриха), изображенное на рис. 3.15. Подходите к созданию стиля с умом. Помните, что сложные линии, особенно не-

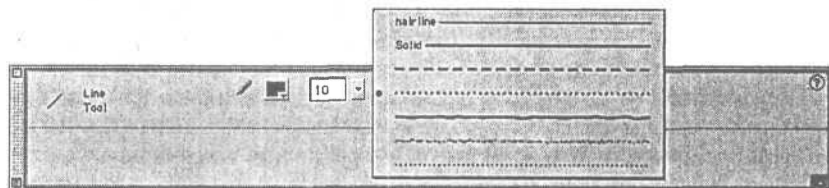


Рис. 3.14. Для задания стиля штриха воспользуйтесь панелью инспектора свойств

правильной формы, требуют сложных векторных расчетов, что отрицательным образом сказывается на размере файла и скорости отображении линий.

На заметку

Использование любого из стилей пунктирной линии существенно влияет на размер файла и скорость воспроизведения изображения. Пользуйтесь такими стилями очень осторожно.

Чтобы отредактировать существующий стиль, сначала убедитесь, что в рабочем поле ничего не выделено. Штрихи можно редактировать с помощью инструмента Ink Bottle (Чернильница), изображенного на рис. 3.16.

Задайте атрибуты штриха на панели инспектора свойств, а затем щелкните на штрихе инструментом Ink Bottle для применения этих атрибутов (рис. 3.17).

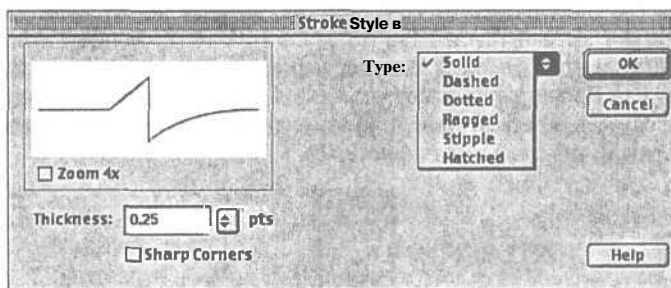


Рис. 3.15. С помощью диалогового окна *Stroke Style* можно создать собственный стиль штриха



Рис. 3.16. Для редактирования штрихов воспользуйтесь инструментом *Ink Bottle*

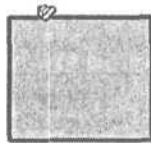


Рис. 3.17. При редактировании штриха измените его атрибуты на панели инспектора свойств, а затем щелкните на штрихе инструментом Ink Bottle

ИСПОЛЬЗОВАНИЕ ЗАЛИВКИ

Заливка — это внутренняя область изображения, которая может представлять собой чистый цвет, градиентное заполнение и даже растровый рисунок. Заливку можно применить к любому замкнутому контуру, а изображение, нарисованное с помощью инструмента Oval (Прямоугольник) или Rectangle (Прямоугольник), будет автоматически заполнено заливкой (за исключением случаев, когда на панели инструментов в подразделе Colors выбрана опция No Color (Бесцветный), как показано на рис. 3.18).



Рис. 3.18. Чтобы нарисовать замкнутый контур без заливки, щелкните в поле Fill на панели инструментов и выберите опцию No Color

Чтобы отредактировать заливку, измените ее параметры на панели Color Mixer, Property Inspector или на панели инструментов, а затем щелкните на заливке инструментом Paint Bucket (рис. 3.19).

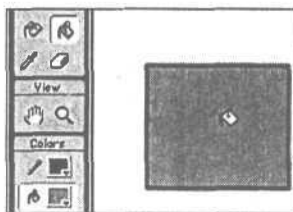


Рис. 3.19. Чтобы отредактировать заливку, измените ее атрибуты, а затем щелкните на заливке инструментом Paint Bucket

ЗАЛИВКА ЧИСТЫМ ЦВЕТОМ

Как и при работе со штрихами, цвет заливки можно задать в трех местах: в поле Fill (Заливка) панели инструментов, на панели инспектора свойств и на панели Color Mixer. Чтобы задать чистый цвет, просто выберите его на любой из этих панелей. Чтобы задать цвет, не относящийся к палитре безопасных Web-цветов, воспользуйтесь панелью Color Mixer (рис. 3.20).

На панели Color Mixer можно выбрать одну из двух моделей цветопередачи: RGB (красный-зеленый-синий) или HSB (оттенок-насыщенность-яркость). Щелкните в правом верхнем углу панели Color Mixer (см. рис. 3.20), из раскрывшегося всплывающего меню Mixer

выберите цветовой режим. RGB и HSB являются разными моделями воспроизведения цвета. RGB — это стандартная аддитивная цветовая модель, созданная для просмотра графики на мониторах компьютеров. В модели RGB цвета создаются путем смешивания основных компьютерных цветов: красного, синего и зеленого. RGB является цветовой моделью по умолчанию. HSB означает "оттенок-насыщенность-яркость". В данной модели цвета определяются в соответствии с этими тремя значениями. Если вы не привыкли к использованию модели HSB, оставьте заданный по умолчанию режим RGB.

Затем щелкните в раскрывающемся меню Fill (Заливка) и выберите значение Solid (Чистый). Для выбора цвета щелкните в цветовом поле. Если хотите детализировать цвет, воспользуйтесь ползунком Brightness (Яркость). Чтобы сделать цвет частично прозрачным, откорректируйте процентное значение в поле Alpha (Прозрачность).

На заметку

При работе с панелью Color Mixer для доступа к элементам управления обязательно щелкните на пиктограмме Stroke или Fills, а не в цветовом поле. После щелчка в цветовых полях будут открываться соответствующие палитры.

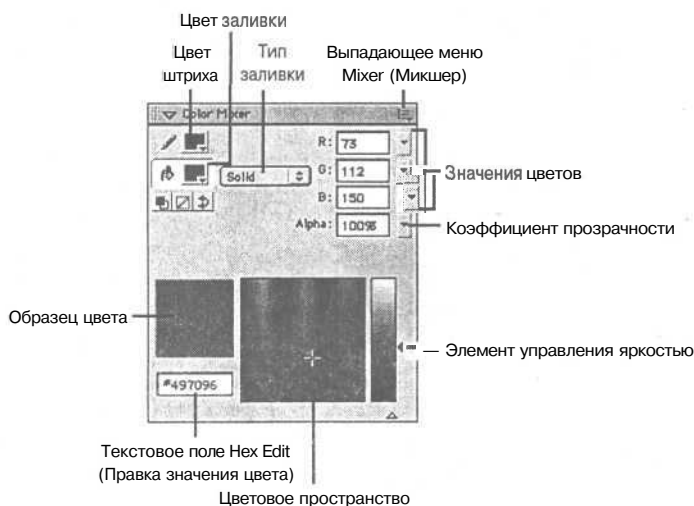


Рис. 3.20. Воспользовавшись панелью Color Mixer, в качестве заливки можно выбрать пользовательский цвет, не относящийся к палитре безопасных Web-цветов, а также градиентное заполнение

ГРАДИЕНТНОЕ ЗАПОЛНЕНИЕ: ЛИНЕЙНОЕ и РАДИАЛЬНОЕ

В качестве заливки часто используют градиентное заполнение, представляющее собой плавный переход от одного цвета к другому. Линейное градиентное заполнение представляет собой непрерывный переход от одного цвета к другому в диапазоне между этими цветами. Радиальное градиентное заполнение представляет собой цветовой диапазон в круговой диаграмме с направленностью от центра окружности. Радиальное градиентное заполнение добавляет глубину круговым объектам, визуализируя их трехмерными. После щелчка в поле Fill панели инструментов, а также на панели инспектора свойств или Color Mixer в нижней части цветовой палитры вы увидите образцы градиентного заполнения (рис. 3.21).

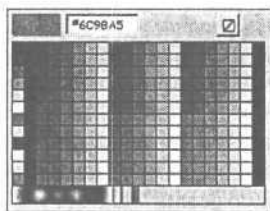


Рис. 3.21. Образцы градиентного заполнения располагаются в левом нижнем углу всплывающего окна цветовой палитры

Пользователь может создать собственное градиентное заполнение и указать используемые в нем цвета. Для создания пользовательского градиентного заполнения воспользуйтесь панелью Color Mixer. Щелкните в поле Fill и выберите тип градиентного заполнения (линейное или радиальное). При этом посередине панели Color Mixer появится панель задания градиентного заполнения вместе с соответствующими указателями (рис. 3.22).



Рис. 3.22. Воспользовавшись соответствующими указателями на панели Color Mixer, можно задать пользовательское градиентное заполнение

Чтобы изменить цвет градиентного заполнения, укажите, какой цвет подлежит изменению, щелкнув на одном из указателей, расположенных по краям линейки задания градиентного заполнения. Затем щелкните в цветовом поле, расположенном в левом верхнем углу панели Color Mixer, и выберите безопасный Web-цвет либо щелкните в цветовом пространстве и выберите пользовательский цвет. Новый цвет отобразится на панели задания градиентного заполнения со стороны выбранного указателя. Для более детальной настройки градиентного заполнения можно добавить дополнительные указатели, щелкнув на панели градиентного заполнения между двумя исходными указателями. Щелкните на указателе и перетащите его на новое место. Для сохранения градиентного заполнения щелкните в правом верхнем углу панели Color Mixer (см. рис. 3.20), в результате чего откроется всплывающее меню Mixer, из которого следует выбрать пункт Add Swatch (Добавить палитру). Новое градиентное заполнение будет добавлено на панели Color Swatches (Каталоги цветов) текущего документа.

РАСТРОВОЕ ИЗОБРАЖЕНИЕ

Панель Color Mixer также позволяет использовать в качестве заливки растровые изображения, являющиеся альтернативой традиционной заливке чистым цветом или градиентным заполнением. В качестве заливки можно назначить любое растровое изображение, которое к тому же можно расположить мозаичным образом, чтобы оно охватывало всю область заливки объекта. При этом создается достаточно необычная заливка, из-за чего она используется не слишком часто. Для создания растровой заливки щелкните на панели Color Mixer в поле Fill.

Затем из раскрывающегося меню Fill выберите пункт Bitmap (Растровое изображение). Откроется диалоговое окно Import to Library (Импортировать в библиотеку), изображенное на рис. 3.23, в котором следует выбрать нужное растровое изображение.

Растровое изображение отображается как текущий цвет в левом верхнем углу панели Color Mixer и применяется в качестве заливки выделенного контура (рис. 3.24).

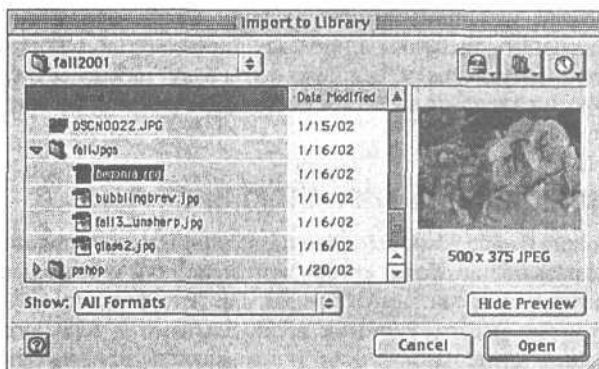


Рис. 3.23. Диалоговое окно *Import to Library* открывается после выбора в качестве типа заливки растрового изображения

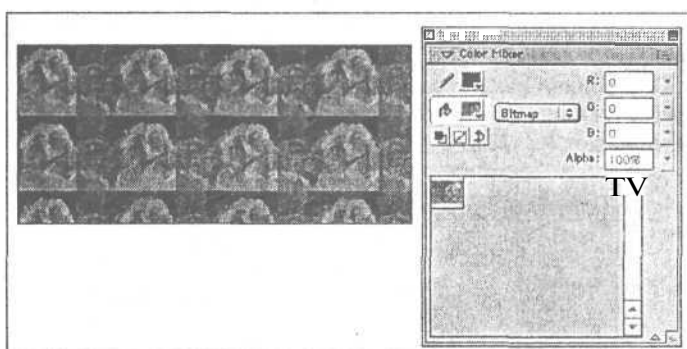


Рис. 3.24. Применить растровое изображение в качестве заливки можно с помощью панели *Color Mixer*

СТИРАНИЕ

Инструмент Eraser (Ластик) позволяет стирать по отдельности или все штрихи и заливку в рабочем поле.

Совет

Инструмент Eraser и его модификаторы подробно описаны в главе 2 "Интерфейс Flash".

Чтобы быстро удалить содержимое рабочего поля, дважды щелкните на инструменте Eraser. Для выборочного удаления заливки, линий, выделенной заливки или областей внутри заливки выберите соответствующий модификатор. Чтобы удалить сегменты штрихов или областей заливки, выберите модификатор Faucet (Кран).

Для удаления элементов можно также выделить их инструментом Arrow (Стрелка) и нажать клавишу <Delete>.

ИСПОЛЬЗОВАНИЕ ВСПОМОГАТЕЛЬНЫХ ЭЛЕМЕНТОВ КОМПОНОВКИ

Во Flash существует несколько вспомогательных элементов, позволяющих повысить точность рисования и создать унифицированные рисунки. Сетки, направляющие и направляющие слои помогут выравнивать элементы в пределах рисунка, что создаст общее впечатление упорядоченности и **концептуальности**. Если элементы будут размещены случайным образом, без взаимосвязи с другими элементами, то документ будет казаться бессистемным и ему будет недоставать гармоничности.

СЕТКИ, НАПРАВЛЯЮЩИЕ И НАПРАВЛЯЮЩИЕ СЛОИ

Сетки, направляющие и направляющие слои позволяют точно расположить объекты в рабочем поле. Сетки и направляющие образуют горизонтальные или вертикальные линии, по которым можно выравнивать или привязывать объекты. При отображении сетки все рабочее поле будет покрыто равномерно расположенными вертикальными и горизонтальными линиями, размещенными на заднем плане изображения. Направляющие представляют собой отдельные горизонтальные или вертикальные линии, которые помещают в рабочее поле вручную. Направляющие слои позволяют создавать собственные визуальные элементы выравнивания.

Сетка представляет собой визуальный каркас для выравнивания нарисованных или помещаемых в документ элементов. Сетка не экспортируется, т.е. в конечном фильме она видна не будет. Для отображения сетки выполните команду **View⇒Grid⇒Show Grid** (Вид⇒Сетка⇒Показать сетку). Для редактирования сетки выполните команду **View⇒Grid⇒Edit Grid** (Вид⇒Сетка⇒Редактировать сетку), в результате чего откроется диалоговое окно Grid (Сетка), изображенное на рис. 3.25. В нем можно указать шаг сетки в пикселях, выбирая значения от 7 до 288.

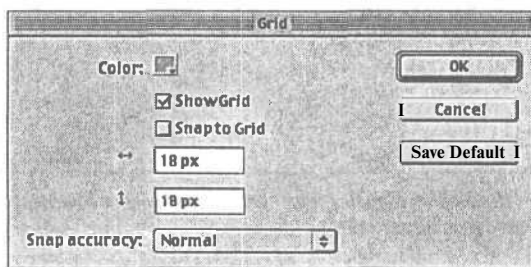


Рис. 3.25. В диалоговом окне Grid можно изменить параметры размера, цвета и привязки к сетке

Здесь же можно включить опцию привязки так, чтобы объекты были выровнены точно вдоль сетки.

При включении сетки ее линии автоматически охватывают все рабочее поле, тогда как направляющие позволяют расположить в рабочем поле вертикальные и горизонтальные линии там, где это необходимо. Направляющие являются наиболее полезным инструментом выравнивания повторяющихся элементов, задающих структуру документа, таких как панели меню или столбцы текста.

Чтобы воспользоваться направляющими, необходимо посредством команды **View⇒Rulers** (Вид⇒Линейки) включить линейки, которые будут отображены вдоль верхней и левой сторон рабочего поля. Для добавления направляющей щелкните на горизонтальной линейке вверху рабочего поля и перетащите курсор вниз от нее, в результате чего будет создана горизонталь-

ная направляющая. Для создания вертикальной направляющей щелкните на вертикальной линейке и перетащите курсор в нужное место рабочего поля. Чтобы переместить направляющую, щелкните на ней инструментом Arrow и перетащите в нужное место. Для привязки объектов к направляющим выполните команду **View⇒Guides⇒Snap to Guides** (Вид⇒Направляющие⇒Привязать к направляющим). Чтобы удалить направляющую с рабочего поля, перетащите ее с помощью инструмента Arrow на исходную линейку.

Для редактирования направляющих выполните команду **View⇒Guides⇒Edit Guides** (Вид⇒Направляющие⇒Редактировать направляющие), в результате чего откроется диалоговое окно Guides (рис. 3.26). Его параметры аналогичны параметрам диалогового окна Grid, а кроме того, в нем есть дополнительная кнопка **Clear All** (Удалить все), после щелчка на которой будут удалены все направляющие.

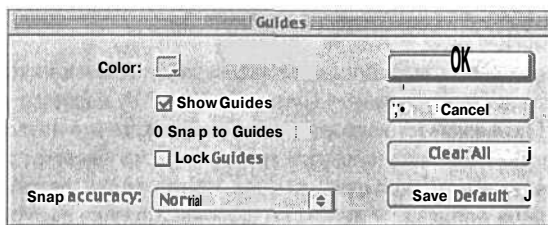


Рис. 3.26. В диалоговом окне *Guides* можно изменить параметры цвета и привязки к направляющим

Направляющие слои позволяют создавать направляющие неправильной формы. С помощью сетки и направляющих можно создать горизонтальные и вертикальные прямые линии, проходящие с одной стороны рабочего поля к другой. Направляющие слои специфичны тем, что позволяют использовать для создания направляющих любой инструмент рисования. При использовании направляющих слоев пользователь не ограничен направляющими, представляющими собой перпендикулярные линии или даже вообще линии. В направляющем слое можно нарисовать диагональную линию или шестиугольник, который будет использоваться для выравнивания объектов. Для создания направляющего слоя щелкните на пиктограмме **Guide** (Направляющий слой), расположенной в нижней части панели слоев. Нарисуйте направляющий контур, а для выравнивания объектов, расположенных на других слоях относительно объектов направляющего слоя, воспользуйтесь панелью **Align** (Выравнивание).

ПАНЕЛЬ ALIGN

С помощью панели **Align** (Выравнивание) объекты в рабочем поле можно выравнивать относительно друг друга. Для выравнивания объектов относительно рабочего поля щелкните на кнопке **To Stage** (К рабочему полю), расположенной в правой части панели (рис. 3.27). Например, этой кнопкой можно воспользоваться для расположения объектов по центру или вдоль левого края рабочего поля. Для выравнивания объектов относительно друг друга щелкните на кнопке **To Stage**. Выделите два или большее число объектов и воспользуйтесь кнопками группы **Align** для выравнивания правых или левых краев объектов, для их центрирования по горизонтали или по вертикали либо для выравнивания верхних или нижних краев. Кнопки группы **Distribute** (Распределение) используются для поочередного распределения ряда объектов: по их верхним или нижним краям, по горизонтальному или вертикальному центру, либо по левым или правым краям.

На заметку

При выравнивании объектов относительно друг друга программа фиксирует, по какому краю вы хотите выполнить выравнивание, а затем использует край самого дальнего в этом направлении объекта для выравнивания остальных объектов.

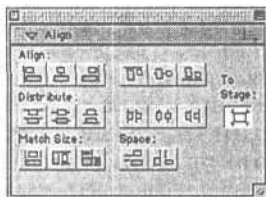


Рис. 3.27. Панель Align позволяет выравнивать объекты в рабочем поле относительно друг друга

ПРИВЯЗКА



Еще одним способом выравнивания элементов является их привязка. Одной из самых простых, но очень нужных функций Flash MX является привязка к пикселям. Поскольку Flash является векторной программой, то это позволяет легко и не всегда преднамеренно помещать объекты между целыми частями пикселей. Кажущиеся незначительными дробные части пикселей, тем не менее, могут разрушить выравнивание и создать проблемы. Функция привязки к пикселям включена только в версию Flash MX. Привязка к пикселям реализуется путем создания сетки с шагом в один пиксель между направляющими и последующей привязки объектов рабочего поля к линиям этой сетки. Для включения привязки к пикселям выполните команду **View⇒Snap to Pixels** (Привязать к пикселям). Теперь любой объект, создаваемый или помещаемый в рабочее поле, будет привязан к целочисленной координате пикселя. Подавляющее большинство проблем выравнивания теперь решено. Во избежание проблем выравнивания изображения рекомендуется всегда включать функцию привязки к пикселям. К сожалению, при открытии нового документа или новом запуске Flash функция привязки к пикселям отключается, поэтому вам придется снова включать ее.

В процессе работы функцию привязки к пикселям можно временно **отключить**, нажав клавишу <C>. При повторном нажатии этой клавиши функция снова будет включена.

На заметку

Если задать увеличение рабочего поля, равное 400%, будет видна сетка привязки к пикселям. Чтобы временно скрыть ее, нажмите и удерживайте клавишу <X>. Если отпустить ее, сетка появится снова.

СОЗДАНИЕ МАСОК

Маски являются аналогами трафаретов: они позволяют отображать части изображений, расположенных на заднем плане. Например, текст можно использовать в качестве маски для отображения частей изображения, расположенных в пределах контуров букв, как показано на рис. 3.28. Создать маску очень просто. Нарисуйте простые контуры на слое, который расположен над слоем, подлежащим маскированию. Вопреки ожиданиям, маски во Flash используются не для скрытия, а для отображения. Поэтому запомните, что маска создается для отображения находящегося под ней содержимого. Поместив контур маски на свое место, щелкните на слое правой кнопкой мыши (либо щелкните, удерживая клавишу <Ctrl>) и из контекстного меню выберите пункт **Mask** (Маска). Таким образом, маска будет применена, а ее слой и все слои, расположенные под ним, блокируются.

Совет

Подробная информация об использовании масок приведена в главе 17 "Демонстрация мощи видеоклипов".



Рис. 3.28. В качестве маски можно использовать текст

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Когда я щелкаю и перетаскиваю контур, штрихи почему-то остаются на своем месте.

После щелчка на контуре, содержащем и штрихи, и заливку, будет выделен только тот из этих элементов, на котором был выполнен щелчок. Чтобы выделить оба элемента, на контуре нужно щелкнуть дважды. Чтобы перетаскать отмеченную область, можно щелкнуть и перетаскать ее с помощью инструмента Arrow.

Можно ли импортировать векторную графику, созданную в других приложениях?

Да, при выполнении команды **File⇒Import** (Файл⇒Импортировать) можно импортировать файлы форматов .ai и .eps программы Illustrator, но, к сожалению, только те, которые были сохранены в формате версии 8 или предыдущей. Импортировать можно и файлы, созданные в программе **FreeHand** версий 7-10. При импортировании во Flash можно сохранить текстовые блоки, слои, символы библиотеки и страницы программы FreeHand. Эта программа входит в линейку продуктов Macromedia, поэтому неудивительно, что Flash поддерживает ее гораздо лучше.

FLASH ЗА РАБОТОЙ: НАСКОЛЬКО БОЛЬШИМ ПОЛУЧИТСЯ ИЗОБРАЖЕНИЕ

Несмотря на все функции точного рисования, имеющиеся во Flash, не существует способа узнать, насколько большим получится нарисованное изображение. Может показаться, что все-таки есть какой-то способ указать, что вы хотите нарисовать квадрат 15x15 пикселей, но, к сожалению, это не так. Вот какие возможности имеются.

- Нарисуйте контур, максимально приближенный к нужным размерам. Затем выделите и отредактируйте его, введя правильные значения ширины и высоты на панели инспектора свойств или Info. Этот подход несколько утомителен, но его можно использовать для квадратов, прямоугольников и кругов. Однако, если попытаться применить его к овалам или прямоугольникам с закругленными углами, то кривые будут искажены.
- Создайте направляющие и расположите их так, чтобы они образовывали нужные размеры создаваемого объекта. Затем нарисуйте объект при включенной функции привязки к направляющим. В процессе рисования объект будет привязан к направляю-

шим, что позволит добиться нужных размеров. Этот способ можно использовать при создании квадратов и прямоугольников, но его сложно применить к кругам, овалам и прямоугольникам с закругленными углами, поскольку выравнивание выполняется по прямоугольной сетке.

- При работе с прямоугольниками, имеющими закругленные углы, начните с создания закругленных углов в прямоугольнике нужных размеров. Затем выделите анкерные точки двух соседних углов, как показано на рис. 3.29, и слегка подвиньте их с помощью клавиш со стрелками до тех пор, пока прямоугольник не приобретет правильные размеры. Этот способ может быть достаточно утомительным, но он поможет сохранить нужный радиус закругления при изменении размеров.



*Рис. 3.29. С помощью инструмента **Subselection** можно изменить размеры прямоугольника с закругленными углами, сохранив при этом нужный радиус закругления*

ИСПОЛЬЗОВАНИЕ РАСТРОВЫХ ИЗОБРАЖЕНИЙ ВО FLASH

В ЭТОЙ ГЛАВЕ...

Импортирование растровых изображений	81
Трассировка растровых изображений	87
Разбивка растровых изображений	92
Сжатие растровых изображений	93
Возможные проблемы	94
Flash за работой: анимация растровых изображений	95

ИМПОРТИРОВАНИЕ РАСТРОВЫХ ИЗОБРАЖЕНИЙ

Несмотря на то что *Flash* является векторной программой, вы не ограничены работой только с нарисованными в ней изображениями. В программу можно легко импортировать файлы других форматов, в том числе и растровые изображения. Векторная графика не может конкурировать с фотореалистичностью растровых изображений. Хотя растровые изображения необходимо редактировать в других программах, *Flash* позволяет импортировать и преобразовывать их.

Растровые изображения во *Flash* можно импортировать во множестве различных форматов. Очень многое зависит от того, установлена ли в вашей системе программа *QuickTime 4* или последующей версии. В табл. 4.1 перечислены форматы файлов, которые можно импортировать во *Flash MX* без установки *QuickTime*.

Плейер *QuickTime* расширяет поддержку некоторых форматов файлов и особенно полезен при коллективном использовании файлов на компьютерах обеих платформ. В табл. 4.2 перечислены импортируемые форматы файлов, доступные при наличии *QuickTime*.

ТАБЛИЦА 4.1. ИМПОРТИРУЕМЫЕ во FLASH MX ФОРМАТЫ ФАЙЛОВ РАСТРОВЫХ ИЗОБРАЖЕНИЙ, НЕ ТРЕБУЮЩИЕ НАЛИЧИЯ QUICKTIME

ФОРМАТ	РАСШИРЕНИЕ ФАЙЛА	ПЛАТФОРМА
Растровое изображение	.bmp	Windows
Расширенный метафайл Windows	.emf	Windows
FutureSplash	.spl	Обе
GIF, анимированный GIF	.gif	Обе
JPEG	.jpg	Обе
PICT	.pct, .pic	Macintosh
PNG	.png	Обе
Flash Player 6	.swf	Обе
Метафайл Windows	.wmf	Windows

ТАБЛИЦА 4.2. ИМПОРТИРУЕМЫЕ во FLASH MX ФОРМАТЫ ФАЙЛОВ, ДОСТУПНЫЕ ПРИ НАЛИЧИИ QUICKTIME

ФОРМАТ	РАСШИРЕНИЕ ФАЙЛА	ПЛАТФОРМА
Photoshop	.psd	Обе
PICT	.pct, .pic	Обе
QuickTime Image	.ptif	Обе
Silicon Graphics Image	.sgi	Обе
TGA	.tga	Обе
TIFF	.tif	Обе

Несмотря на то что с помощью QuickTime можно импортировать файлы Photoshop, имеющие слои при этом выравниваются. В большинстве случаев формат импортируемых растровых изображений определяется приложением, в которых эти изображения редактируются.

Процедура импортирования растровых изображений очень проста. Импортируемые элементы хранятся в библиотеке, поэтому лучше всего импортировать растровые изображения прямо в библиотеку. Это будет особенно полезно при импортировании большого количества файлов. Импортировать можно одновременно несколько элементов, а потом работать с каждым из них индивидуально. Элементы можно также импортировать прямо в рабочее поле. При этом импортируемый экземпляр будет помещен в рабочее поле, а импортируемый файл — в библиотеку.

Чтобы импортировать элемент прямо в рабочее поле, выполните команду **File⇒Import (Файл⇒Импортировать)**, как показано на рис. 4.1, а для импортирования элемента в библиотеку выполните команду **File⇒Import to Library (Файл⇒Импортировать в библиотеку)**.

При выполнении любой из указанных команд откроется диалоговое окно Import (рис. 4.2), в котором следует выбрать импортируемый файл. Из раскрывающегося меню Show (на компьютерах Macintosh) или Type of File (Тип файлов) (на компьютерах с Windows) выберите либо тип импортируемого файла, либо пункт All Formats (Все форматы). В диалоговом окне имеется панель предварительного просмотра выбранного файла. Для начала процедуры импортирования щелкните на кнопке Open (Открыть).

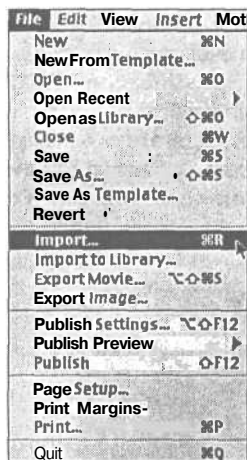


Рис. 4.1. Чтобы импортировать файл в рабочее поле, изменю File выберите пункт Import, а для импортирования в библиотеку — пункт Import to Library

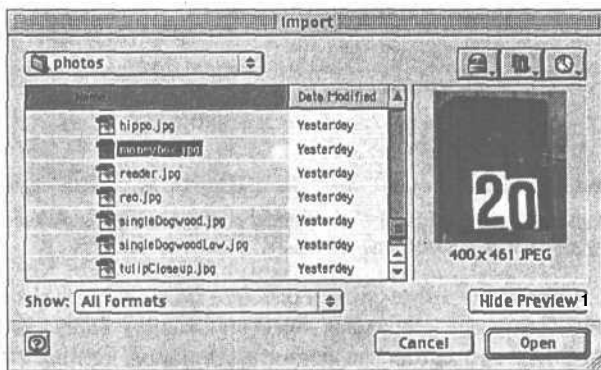


Рис. 4.2. В диалоговом окне Import можно выбрать и предварительно просмотреть файлы, подлежащие импортированию

После выбора команды File⇒Import выбранный файл будет помещен и в рабочее поле, и в библиотеку (рис. 4.3).

ПОДГОТОВКА РАСТРОВЫХ ИЗОБРАЖЕНИЙ К ИМПОРТИРОВАНИЮ

Несмотря на то что процедура импортирования растровых изображений очень проста, редактировать их во Flash так же, как векторную графику, нельзя. Следовательно, перед импортированием во Flash растровые изображения следует подготовить так, чтобы были выполнены все операции их редактирования, с которыми не сможет справиться Flash. Самое главное, чтобы перед импортированием растровые изображения были обрезаны так, чтобы их размеры стали такими, какие будут использоваться во Flash.

Очень важно понимать, каким образом векторная природа Flash влияет на обработку растровых изображений. При работе с векторной графикой для ее создания используются хранимые инструкции, тогда как при работе с растровыми изображениями используются хранимые данные для воспроизведения изображения попиксельно. Несмотря на возможность импортирования растровых изображений во Flash, в распоряжении пользователя не будет всех

тех возможностей, которые предусмотрены для работы с векторной графикой, а некоторые из имеющихся функций реализуются за счет увеличения размеров файла. Характерным примером является изменение размеров растрового изображения.

Размер импортированного во Flash изображения (см. рис. 4.3) слишком велик. Протестируем текущее растровое изображение. Для этого сначала выполните команду **Control⇒Test Movie** (Управление⇒Тестировать фильм), а затем в режиме Test — команду **View⇒Bandwidth Profiler** (Вид⇒Профайлер пропускной способности). Текущий размер фильма составляет 60 Кбайт, как показано на рис. 4.4.

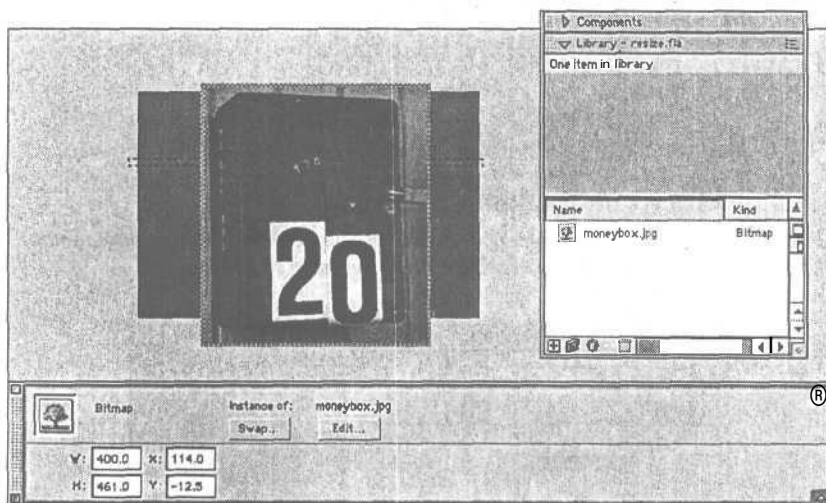


Рис. 4.3. При выполнении команды **File⇒Import** выбранный файл будет помещен в библиотеку, а его экземпляр — в рабочее поле

Изменим размер растрового изображения в рабочем поле, воспользовавшись инструментом Transform (Преобразование) и удерживая нажатой клавишу <Shift> для сохранения пропорций изображения. Снова протестировав фильм, мы увидим, что, несмотря на то, что размеры растрового изображения были уменьшены на 50%, размер файла все равно составляет 60 Кбайт, как показано на рис. 4.5.

Как такое могло произойти? Ведь размер файла растрового изображения определяется размерами самого изображения: чем больше изображение, тем большее количество пикселей нужно отобразить. Дело в том, что при работе во Flash вступают в силу ограничения, характерные для векторной графики. Flash не может уменьшить файл за счет уменьшения изображения. Кроме того, при изменении размеров растровых изображений, особенно при их увеличении, понижается качество файла. Растровые изображения содержат конечное количество пикселей. При увеличении изображения возникает недостаток данных о цвете, необходимого для заполнения дополнительных пикселей, поэтому для них используется цвет, представляющий собой промежуточный оттенок соседних точек. Результатом являются странные искажения, свойственные многим некачественным изображениям, размещенным в Web.

Из всего сказанного можно сделать следующий вывод: заранее планируйте размещение растровых изображений в своих фильмах и импортируйте их именно с такими размерами, которые в дальнейшем не нужно будет изменять. Например, создайте изображение в программе Photoshop и импортируйте его во Flash, заранее уменьшив на 50%. На рис. 4.6 видно, что размер файла при этом будет на 40% меньше по сравнению с тем, что можно было бы получить при аналогичном изменении размеров изображения во Flash.

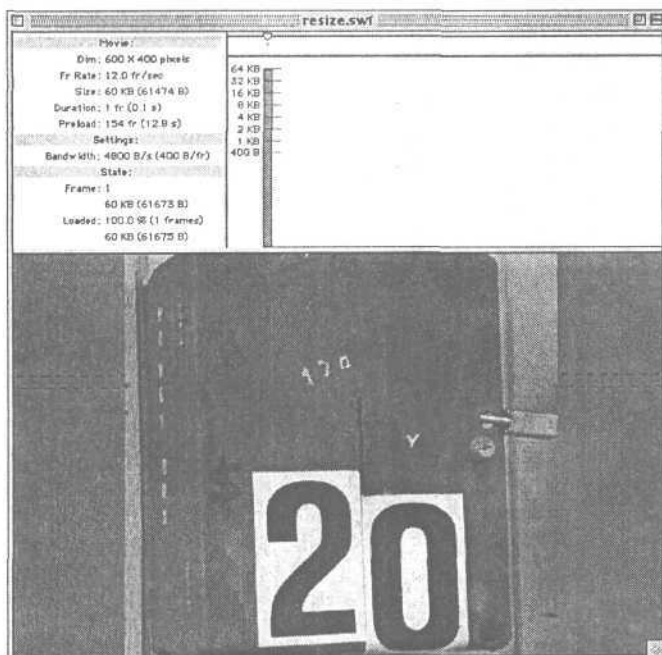


Рис. 4.4. Текущий размер импортированного растрового изображения составляет 60 Кбайт

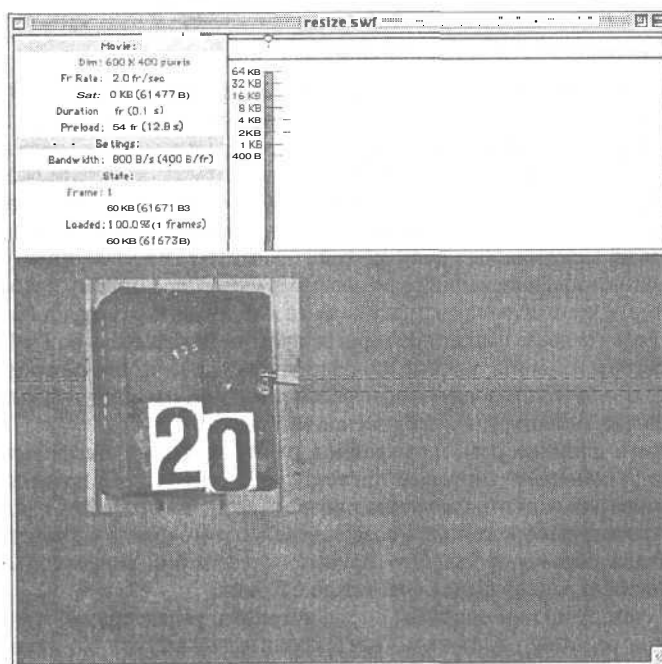


Рис. 4.5. Даже после сжатия растрового изображения на 50% общий размер файла все равно остается равным 60 Кбайт

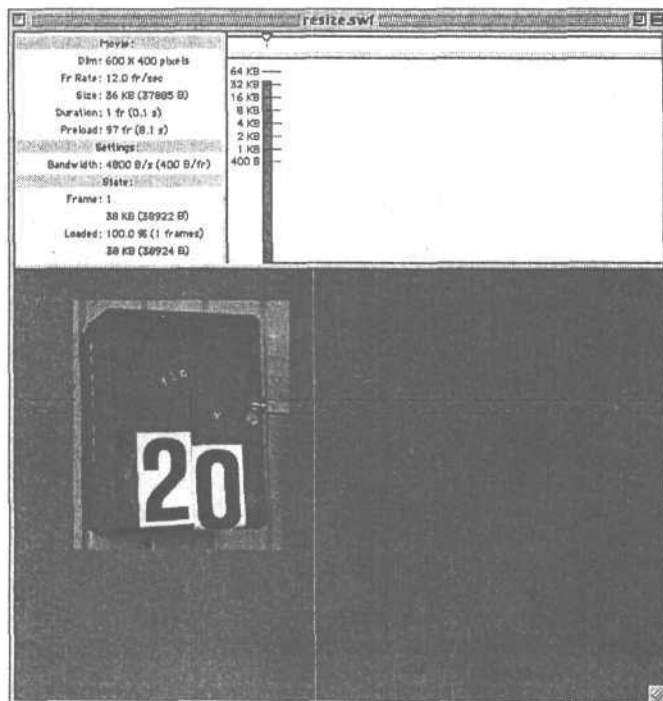


Рис. 4.6. Импортирование растрового изображения с уменьшенными размерами приведет к тому, что размер файла фильма уменьшится до 36 Кбайт

Во Flash MX на панели инспектора свойств, под экземпляром раstra (см. рис. 4.3), расположена кнопка Edit (Редактировать). После щелчка на ней запустится внешняя программа, в которой можно отредактировать данное изображение. Для этого выделите импортированное растровое изображение и щелкните на кнопке Edit. В результате растровое изображение будет открыто во внешней программе редактирования. Отредактируйте изображение и сохраните его. При этом изображение во Flash будет обновлено. Однако, работая с описанным выше изображением, я обнаружил, что при изменении размеров с помощью такой методики был получен файл большего размера (48 Кбайт) по сравнению с тем, который был получен при изменении размеров перед импортированием (36 Кбайт). Это еще раз говорит о том, что лучше всего спланировать **все** заранее и уменьшить размер изображения перед его импортированием. Если же после импортирования растрового изображения во Flash вы обнаружите, что его размеры требуется изменить, удалите его из рабочего поля и из библиотеки, вернитесь к исходному изображению в соответствующем графическом редакторе, измените нужным образом размеры и только потом заново импортируйте изображение во Flash.

Чтобы уменьшить влияние размеров файлов импортируемых растровых изображений, воспользуйтесь альфа-каналами, которые представляют собой слои, добавляющиеся к растровым изображениям для создания эффекта прозрачности. Возвращаясь к примеру рассмотренного выше растрового изображения, я создал версию этого файла в Photoshop и использовал альфа-каналы для обрезки фона. Это привело к созданию силуэтной версии рисунка (рис. 4.7), что уменьшило размер файла фильма до 6 Кбайт.

Короче говоря, чем лучше вы знакомы с графическими **редакторами**, тем лучших результатов достигнете при работе с растровыми изображениями во Flash. Но даже если вы не являетесь большим специалистом в программах редактирования изображений, то все равно сможете добиться уменьшения размеров файлов за счет планирования, обрезки и изменения размеров растровых изображений перед их импортированием.

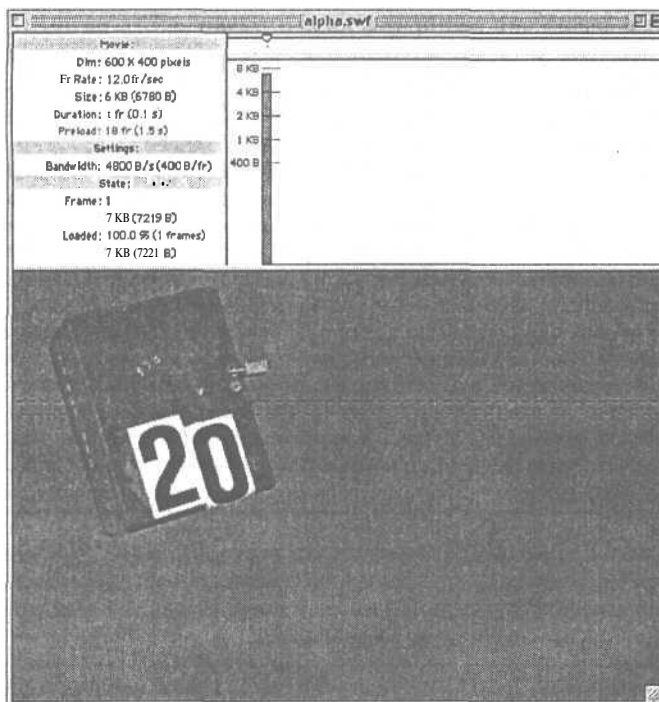


Рис. 4.7. Импортирование того же самого растрового изображения, но к которому для обрезки фона были применены альфа-каналы, привело к уменьшению размера фильма до 6 Кбайт

ТРАССИРОВКА РАСТРОВЫХ ИЗОБРАЖЕНИЙ

Обычно растровые изображения имеют больший размер файла по сравнению с векторными изображениями, что может оказать существенное влияние на размер фильма. Однако в целом сравнение растровой и векторной графики сходно сравнению яблок и слонов. Растровые и векторные изображения имеют совершенно разное предназначение. Растровые изображения лучше всего подходят для фотографий, а векторная графика — для нарисованных изображений, содержащих участки сплошного цвета.

Совет

Подробная информация о векторной графике приведена в главе 3 "Рисование во Flash".

Во Flash существует возможность преобразования растровых изображений в векторные, что при преобразовании подходящего растрового изображения приводит к уменьшению размера файла. Этот процесс получил название *трассировки растрового изображения*.

Не все растровые изображения подходят для трассировки. Трассировка очень детализированного фотографического изображения приведет к тому, что файл векторного изображения будет больше исходного растрового. Помните, что векторный формат лучше всего подходит для простой графики, содержащей однородные цвета, которые легко описать несколькими инструкциями визуализации. Представьте себе, какие инструкции нужно задать для создания представления обычной спальни. Сложные инструкции визуализации сводят на нет то уменьшение размера файла, которого стремятся достичь при трассировке растрового изображения.

Растровые изображения против векторных: цена графики

Каждый формат, и растровый, и векторный, имеет свою стоимость, которая определяется способом передачи графической информации. Поскольку отображение растрового изображения осуществляется попиксельно, то стоимость определяется скоростью загрузки. Большие размеры файлов, медленная скорость соединения, конкуренция загружаемых элементов фильма из-за недостаточной полосы пропускания, общая перегрузка *Internet* — все это приводит к замедлению загрузки растровых изображений и, следовательно, к высокой стоимости содержимого. Что касается векторной графики, то здесь выполняется передача ряда инструкций, необходимых для создания изображения, а инструкции выполняются на компьютере пользователя. Обычно инструкции быстрее загружаются и визуализируются, но при этом нагрузке подвергается процессор пользователя. Пользователи, работающие на *старых*, медленных компьютерах, дольше ожидают отображения сложной векторной графики. Стоимость векторной графики может в два раза превосходить стоимость растровых изображений. Дело в том, что сложные инструкции приводят к созданию файлов больших размеров, которые медленно загружаются. Кроме того, инструкции должны быть выполнены на компьютере пользователя. В экстремальных ситуациях (например, когда речь идет о бесконечных, случайных векторах или векторах со сценариями, таких как на узле www.playstation.com) медленные компьютеры могут "погрязнуть" в вычислениях и даже "зависнуть", если передаваемые инструкции окажутся слишком сложными для исполнения.

На рис. 4.8 приведен пример изображения, хорошо подходящего для трассировки. Лучше всего для выполнения этой операции подходят простые изображения с большими участками цвета.

Изображения со сложной структурой, как, например, на рис. 4.9, или с большим количеством цветовых оттенков лучше оставить растровыми.

Совет

Несмотря на то что в векторный формат проще всего преобразовывать простые изображения, эту операцию можно применить и к сложным растровым изображениям. Применяя менее точные параметры трассировки, сложное изображение можно упростить и создать абстрактное стилизованное векторное изображение.

Для трассировки растрового изображения выделите его экземпляр в рабочем поле, а затем выполните команду **Modify**⇒**Trace Bitmap** (**Изменить**⇒**Трассировать растровое изображение**). При этом откроется диалоговое окно **Trace Bitmap**, как на рис. 4.10.

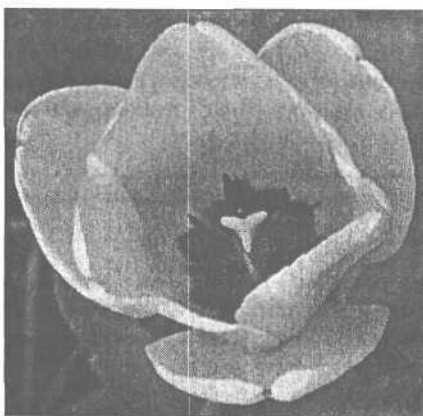
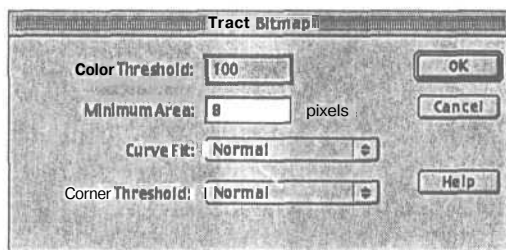


Рис. 4.8. Благодаря простой форме цветка и небольшому количеству цветовых оттенков это изображение хорошо подходит для выполнения трассировки



Рис. 4.9. Изображение, имеющее сложную структуру и множество оттенков, лучше оставить растровым



*Рис. 4.10. Диалоговое окно **Trace Bitmap** позволяет управлять процедурой трассировки изображения*

В диалоговом окне содержатся четыре параметра трассировки. Параметр **Color Threshold** (Порог цвета) определяет, насколько хорошо в трассированном растровом изображении будут сохранены цветовые детали. Чем меньше число (в диапазоне от 1 до 500), тем больше оттенков будет сохранено и тем более сложной будет трассировка. Параметр **Minimum Area** (Минимальная область) указывает число окружающих пикселей (от 1 до 1000), которые будут учитываться при назначении цвета отдельного пикселя.

Параметр **Curve Fit** (Соответствие кривой) определяет, насколько тщательно следует придерживаться дуг кривых при трассировке. На рис. 4.11 показано, что значения данного параметра варьируются от **Pixels** (Пиксели), соответствующего самой точной и сложной трассировке, до **Very Smooth** (Очень сглажено), что соответствует наименее детальной трассировке.

Параметр **Corner Threshold** (Порог угла) аналогичным образом указывает, насколько точно следует выполнять трассировку углов. Параметр может принимать значения **Many Corners** (Много углов), **Normal** (Обычно) и **Few Corners** (Мало углов). Выполните трассировку с заданными по умолчанию параметрами, чтобы получить базовое представление об этой операции, а затем попробуйте выполнить ее с другими параметрами. Вы обнаружите, что более сложная трассировка выполняется дольше. Если вы хотите получить другие результаты трассировки, выполните команду **Edit**⇒**Undo** (**Правка**⇒**Отменить**) или воспользуйтесь комбинацией клавиш <Cmd+Z> (Macintosh) или <Ctrl+Z> (Windows). Снова выполните трассировку изображения и поэкспериментируйте с различными параметрами.

Полученные результаты будут сильно отличаться друг от друга, так что не бойтесь экспериментировать. Обращайте внимание и на размер файла. Менее точные параметры трассировки приводят к созданию изображений меньшего размера и более импрессионистского вида (рис. 4.12).

При задании параметров более точной трассировки будет создано векторное изображение, очень похожее на исходное растровое, как на рис. 4.13. Однако чем сложнее трассировка, тем большим будет размер файла.

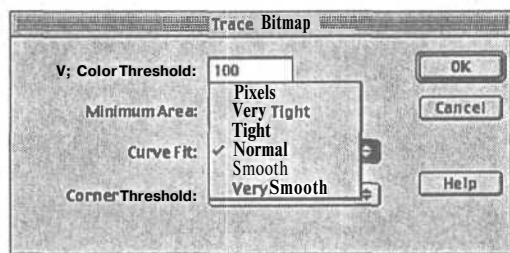


Рис. 4.11. При трассировке растрового изображения диапазон значений параметра Curve Fit варьируется от Pixels до Very Smooth

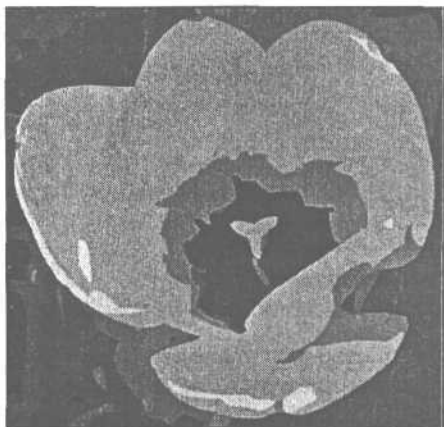


Рис. 4.12. После трассировки растровые изображения могут стать импрессионистскими

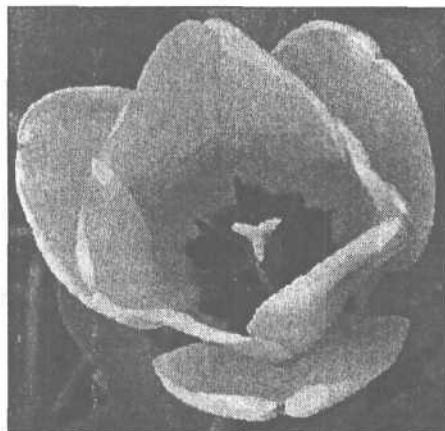


Рис. 4.13. При более точных параметрах трассировки будет получено изображение, близкое к исходному

ОПТИМИЗАЦИЯ ТРАССИРОВАННЫХ ИЗОБРАЖЕНИЙ

При выполнении сложной трассировки размер полученного файла будет большим, чем размер исходного растрового изображения. Если вы работаете не на медленном компьютере, но выполнения трассировки ожидаете больше, чем несколько секунд, обратите внимание на размер файла. Вероятнее всего, заданные параметры трассировки слишком точны.

Чтобы увидеть, насколько сложной оказалась полученная трассировка, выполните команду View⇒Outlines (Вид⇒Контур), в результате чего на экране будут отображены только контуры изображения (рис. 4.14). Чтобы вернуться к обычному виду, выполните команду View⇒Fast (Вид⇒Быстрый).



Рис. 4.14. Для оценки сложности выполненной трассировки просмотрите контуры изображения

Трассировка изображения, контуры которого изображены справа, очевидно, слишком сложна для размещения в Internet. Одним из способов уменьшить детализацию трассировки является оптимизация кривых. Для этого выполните команду **Modify⇒Optimize** (Изменить⇒Модифицировать), в результате чего откроется диалоговое окно Optimize Curves (Оптимизация кривых), приведенное на рис. 4.15.

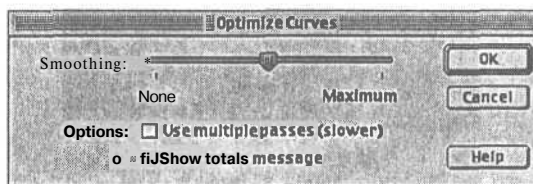


Рис. 4.15. Корректируя параметры в диалоговом окне Optimize Curves, можно упростить трассированное изображение

Поэкспериментируйте, перетаскивая ползунок Smoothing (Сглаживание). Чтобы на экран выводилось сообщение о количестве кривых и степени сглаживания, установите флажок в поле опции Show Totals Message (Показывать итоговое сообщение) и щелкните на кнопке OK. В результате при выполнении оптимизации на экран будет выводиться итоговое сообщение (рис. 4.16).

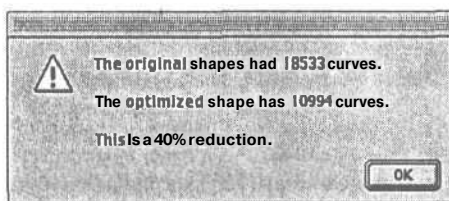


Рис. 4.16. В итоговом сообщении указываются данные об общем уменьшении числа кривых

Команда Optimize Curves является очень мощным средством. Обычно с ее помощью можно оптимизировать изображения на 40-50%. Если этого недостаточно, отмените выполнение трассировки и выполните ее снова, но с другими параметрами.

РАЗБИВКА РАСТРОВЫХ ИЗОБРАЖЕНИЙ

Разбивка растрового изображения заключается в разделении его на составляющие пиксели, которые затем можно выделить и индивидуально редактировать. Если растровое изображение не разбито, его можно выделить и работать с ним только глобально, т.е. как с единым целым. После того как растровое изображение будет разбито, вы сможете воспользоваться инструментами рисования и окраски для редактирования отдельных пикселей.

✦ Чтобы разбить растровое изображение, выделите его в рабочем поле и выполните команду **Modify⇒Break Apart (Изменить⇒Разбить)**. Затем для выделения участков растрового изображения воспользуйтесь инструментом **Lasso (Лассо)** с модификатором **Magic Wand (Волшебная палочка)**. После щелчка на растровом изображении модификатором **Magic Wand** будут выделены пиксели в соответствии с цветовым диапазоном, как показано на рис. 4.17.

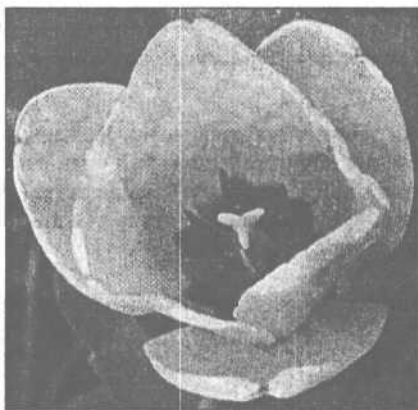


Рис. 4.17. Модификатор Magic Wand инструмента Lasso позволяет выделить отдельные участки растрового изображения, которые помечаются точками

Для доступа к модификатору **Magic Wand** выберите на панели инструментов инструмент **Lasso**. Затем в подразделе **Options** выберите модификатор **Magic Wand Properties (Свойства волшебной палочки)**, как показано на рис. 4.18, и посмотрите, какими будут параметры выделения.

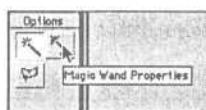


Рис. 4.18. Выберите инструмент Lasso, а затем для доступа к настройкам щелкните на кнопке Magic Wand Properties

В диалоговом окне **Magic Wand Settings (Параметры волшебной палочки)**, изображенном на рис. 4.19, параметр **Threshold (Пороговое значение)** указывает, насколько точно цвет должен соответствовать пикселю, на котором производится щелчок, чтобы быть включенным в выделенный фрагмент. Значения этого параметра могут лежать в диапазоне от 0 до 200. Чем больше число, тем шире диапазон цветов, включаемых в выделенный фрагмент. Значение **О** свидетельствует о том, что в выделенный фрагмент будут включены только те пиксели, цвет которых точно совпадает с цветом пикселя, на котором был выполнен щелчок. Параметр

Smoothing (Сглаживание) определяет, насколько гладкими должны быть края выделенного фрагмента. Он может принимать значения Pixels (Пиксели), Rough (Грубое), Normal (Обычное) и Smooth (Гладкое). Повторные щелчки приводят к добавлению элементов к выделенному фрагменту. После выделения нужного фрагмента воспользуйтесь инструментом Fill для изменения цветов заливки.

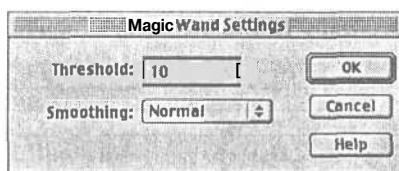


Рис. 4.19. В диалоговом окне *Magic Wand Settings* можно изменить способ выделения фрагментов изображения

СЖАТИЕ РАСТРОВЫХ ИЗОБРАЖЕНИЙ

В связи с тем, что наличие растровых изображений может существенно увеличить размер файла фильма, процедура сжатия изображений с целью уменьшения размера файла является очень важной задачей. При публикации фильма используются параметры сжатия растровых изображений, заданные по умолчанию. Однако исходные параметры сжатия можно изменить, откорректировав свойства растрового изображения. Эта процедура будет особенно уместной, если ранее выполнялась трассировка растрового изображения. Для доступа к диалоговому окну *Bitmap Properties* (Свойства растровых изображений) выделите изображение на панели *Library* (Библиотека) и щелкните на кнопке *Properties* (Свойства), как показано на рис. 4.20. Можно также выделить элемент библиотеки, щелкнуть на нем правой кнопкой мыши (*Windows*) либо щелкнуть, удерживая клавишу <Ctrl> (*Macintosh*), и из появившегося контекстного меню выбрать пункт *Properties*.

Откроется диалоговое окно *Bitmap Properties* (рис. 4.21). Из раскрывающегося меню *Compression* (Сжатие) выберите пункт *Photo (JPEG)* для сжатия растрового изображения в формат JPEG. В этом случае при выполнении сжатия будут отброшены данные о некоторых пикселях. Для сохранения исходных параметров сжатия импортированного растрового изображения установите флажок в поле опции *Use Imported JPEG Data* (Использовать импортированные данные JPEG). При каждом сжатии JPEG-файла будут отбрасываться данные, и качество изображения будет существенно ухудшаться. Как правило, во избежание ухудшения качества изображения лучше всего использовать импортированные данные JPEG. Можно задать новые параметры качества JPEG, но это не улучшит качество импортированного JPEG-файла, а, наоборот, приведет к его дальнейшему снижению. Чтобы задать новые параметры качества JPEG, снимите флажок с поля опции *Use Imported JPEG Data*, а в поле *Quality* (Качество) введите число от 1 до 100. Если вы зададите новые параметры JPEG, обязательно проверьте качество полученного изображения и убедитесь, что оно существенно не ухудшилось. Используйте формат JPEG при работе с растровыми изображениями, содержащими фотографические данные и широкий диапазон полутонов.

Чтобы выполнить сжатие без потери данных о пикселях, из раскрывающегося меню *Compression* выберите пункт *Loseless (PNG/GIF)* (Без потерь (PNG/GIF)). Данный тип сжатия лучше использовать при работе с трассированными растровыми изображениями, представляющими собой простые изображения с небольшим количеством цветов.

Чтобы сравнить размер файла, к которому применены параметры сжатия, с исходным, щелкните на кнопке Test (Тестировать), в результате чего в нижней части диалогового окна появятся данные о сжатии (рис. 4.22). После щелчка на кнопке Update (Обновить) вы сможете увидеть эффект применения параметров, просмотрев полученное изображение в левом верхнем углу диалогового окна. Поэкспериментируйте с различными параметрами в поисках оптимального соотношения между качеством изображения и размером файла. Когда вы будете удовлетворены параметрами сжатия, щелкните на кнопке OK.

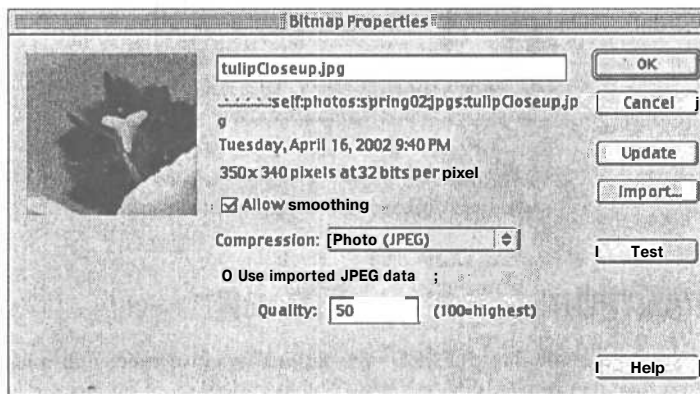


Рис. 4.21. В диалоговом окне *Bitmap Properties* можно указать параметры сжатия импортированных растровых изображений

JPEG: quality = 50; original = 476.0 kb,
compressed = 10.3 kb, 2% of original

Рис. 4.22. После щелчка на кнопке *Test* будет выполнено сравнение размера файла после сжатия с исходным файлом

Растровые изображения могут существенно увеличить привлекательность Flash-фильма, особенно в комбинации с "родной" для программы векторной графикой. Растровые изображения являются неотъемлемым элементом при передаче фотографических данных. Пользуйтесь растровыми изображениями с осторожностью, все время обращая внимание на размер файла, но всеми средствами старайтесь внедрять их в свои фильмы.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему в созданном фильме растровые изображения получились размытыми ?

Для растровых изображений, получившихся размытыми, проверьте параметры, заданные в диалоговом окне *Bitmap Properties*. Проверьте, чтобы не был установлен флажок в поле опции *Allow Smoothing* (Разрешить сглаживание), поскольку сглаживание устраняет контурные неровности растровых изображений, что может привести к их размытию или расфокусировке. Кроме того, в диалоговом окне *Publish Settings* (Параметры публикации) перейдите во вкладку *HTML* и из раскрывающегося меню *Quality* (Качество) выберите пункт *Best* (Самое лучшее).

Совет

Подробная информация о параметрах публикации приведена в главе 30 "Оптимизация, публикация и экспортирование фильмов".

Есть ли какой-то способ выполнить трассировку растрового изображения так, чтобы на одних участках оно стало более детализированным, а на других — менее детализированным?

Трассировка применяется к изображению глобально, с единым уровнем детализации на всех его участках. Однако трассировку изображения можно выполнить дважды, задав различный уровень детализации, а затем скомбинировать два полученных изображения, накладывая их друг на друга и вытирая ненужные участки. Таким образом, к большей части изображения можно применить простую трассировку, а отдельные участки сделать более детализированными. Однако этот способ приведет к увеличению размера файла, поэтому не слишком увлекайтесь им.

FLASH ЗА РАБОТОЙ: АНИМАЦИЯ РАСТРОВЫХ ИЗОБРАЖЕНИЙ

Векторная природа Flash становится очевидной при анимации растровых изображений. Программа не интерполирует растровые изображения так же легко, как векторные. Помните, что в растровых изображениях производится визуализация каждого пикселя, поэтому интерполировать нужно гораздо большее количество данных. Растровые изображения могут существенно увеличить размер файла и вызвать совершенно нежелательные эффекты, например запинки при воспроизведении.

Не слишком увлекайтесь анимацией растровых изображений, поскольку для достижения наилучших результатов анимация должна быть как можно проще. Чем меньше будут растровые изображения, тем меньше запинок будет при воспроизведении анимации. Меньшие растровые изображения существенно не повлияют на размер файла. Кроме того, постарайтесь ограничить эффекты, влияющие на размер файла, кадрами, в которых присутствует как можно меньше других эффектов. В противном случае пользователям придется долго ждать загрузки содержимого, поскольку элементы фильма будут конкурировать друг с другом из-за недостаточной полосы пропускания. Если без анимации растровых изображений не обойтись, можете для получения лучшего качества увеличить частоту смены кадров, но это также приведет к увеличению размера файла. Как всегда, поэкспериментируйте с различными значениями частоты смены кадров и найдите золотую середину между качеством изображения и размером файла.

РАБОТА С ТЕКСТОМ

В ЭТОЙ ГЛАВЕ...

Инструмент Text	99
Инспектор свойств	100
Атрибуты символов	101
Атрибуты абзаца	103
Параметры текста	105
Редактирование и преобразование текста	110
Эффективные приемы работы с текстом	112
Возможные проблемы	113
Flash за работой: копирование текстовых полей	114

В программе Flash MX представлен широчайший диапазон средств работы с текстом, начиная от простого статичного и заканчивая стилизованным и анимированным. Более того, в распоряжении пользователя имеются средства, позволяющие динамически вставлять текст из внешних источников. Чтобы в полном объеме освоить методы манипулирования текстом во Flash MX, необходимо иметь четкое представление о том, как обрабатывается текст в программе, и знать некоторые специальные приемы работы с ним.

Весь текст, который появляется на мониторах компьютеров, отображается в пикселях. Как такое может происходить, учитывая ежеминутно подчеркиваемую векторную природу Flash? Дело в том, что мониторы отображают пиксели и точки независимо от форматов приложений. Как раз такой способ представления особенно неудачен для отображения букв. Большинство букв представляют собой не блочные конструкции, а состоят из элегантных кривых. Однако единственным способом отображения букв является преобразование кривых в пиксели, а при таком способе очень многое теряется.

Прежде чем вы решите отказаться от использования текста, обратите внимание на способ преодолеть или, по крайней мере, минимизировать трудности, связанные с блочным типом отображения пикселей на компьютерных мониторах. Попыткой восстановления некоторой утонченности кривых в пиксельном окружении является процедура сглаживания. При ее выполнении между шрифтом и фоном добавляются пиксели, имеющие промежуточный цвет и оттенок. Это приводит к тому, что формы букв становятся более гладкими, хотя и несколько нечеткими. Пусть, например, имеется черный текст на белом фоне. На рис. 5.1 показано, что при выполнении процедуры сглаживания вокруг искривленных участков черных букв добавляются серые пиксели.

К сожалению, процедура сглаживания приводит к тому, что текст, особенно небольшого размера, становится нечетким (рис. 5.2).

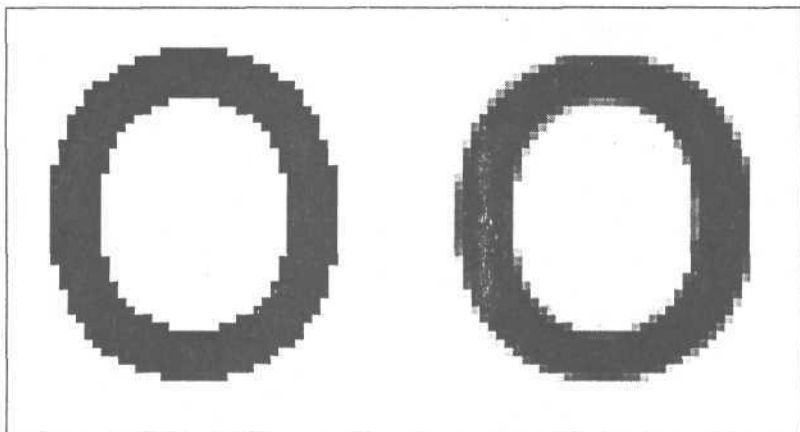


Рис. 5.1. Буква, изображенная справа, является сглаженной. Для достижения эффекта к кривым были добавлены серые пиксели

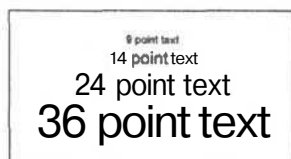


Рис. 5.2. Процедуру сглаживания лучше применять к шрифтам относительно больших размеров. В противном случае текст будет выглядеть размытым

Можно использовать растровые шрифты, которые разработаны без использования кривых и предназначены специально для отображения на компьютерах. Из-за отсутствия кривых к таким шрифтам не нужно применять процедуру сглаживания. Но в связи с тем, что данные шрифты являются пиксельными, они должны иметь строго определенные размеры и позиционироваться точно на целых пикселях. Такие шрифты лучше использовать при необходимости отображения букв с малыми размерами.

Благодаря растущей популярности Flash растровые шрифты становятся все более доступными для широкого круга пользователей. Одним из самых лучших ресурсов для получения таких шрифтов является узел www.miniml.com, на котором представлено большое количество гарнитур растровых шрифтов, в том числе и разработанные специально для заголовков и основного текста, а также шрифты с засечками.

Ресурсы растровых шрифтов

Растровые шрифты являются очень популярными среди дизайнеров. Не проходит и дня, чтобы в Internet не был представлен какой-нибудь новый шрифт. Ниже приведено несколько полезных ресурсов.

<http://cgm.cs.mcgill.ca/~luc/pixel.html>. Совершенно фантастический ресурс, на котором представлено огромное количество ссылок на всевозможные шрифты.

<http://www.miniml.com>. Предлагает большое количество качественных шрифтов разных размеров. Некоторые шрифты можно загрузить бесплатно, а за 100 долларов вы получите неограниченный доступ к огромному количеству шрифтов, созданных в 2002 году.

<http://www.fontsforflash.com>. На этом узле продается большое количество шрифтов. К сожалению, пока что эти шрифты предназначены только для PC. Бесплатно с данного узла можно загрузить только некоторые шрифты, а за каждый из остальных придется платить.

Бесплатные пиксельные шрифты имеются на следующих узлах:

<http://www.dsg4.com/04/extra/bitmap/index.html>;

<http://www.with-m.com> (для доступа к растровым шрифтам последовательно выберите пункты Item и Font);

<http://www.orgdot.com/aliasfonts>.

ИНСТРУМЕНТ TEXT

Начнем с основ. Перед началом ввода текста проверьте, чтобы была включена функция Snap to Pixels (Привязать к пикселям). (Для этого выполните команду **View** ⇨ **Snap to Pixels**.) При этом растровые шрифты будут гарантированно выровнены по целым пикселям, и когда текст будет помещен в основную временную шкалу, он не станет размытым. Однако, когда текст внедряется в пределах символов и затем помещается в основную временную шкалу, гарантии того, что он не будет размытым, нет.

Совет

Подробная информации о предотвращении нечеткости шрифтов приведена в разделе "Возможные проблемы" в конце этой главы.

Для добавления текста к фильму выполните следующие действия.

1. Выберите инструмент Text.
2. Щелкните в рабочем поле, в результате чего будет создана точка ввода текста.
3. Начните ввод текста.

Вот и все, что необходимо. Кроме того, можно выбрать инструмент Text и перетащить курсор по рабочему полю, создавая тем самым текстовый блок. Рисуя текстовый блок, вы можете задать ширину многострочного текста. Перетаскивая курсор по рабочему полю, обратите внимание, что изменяется только ширина текстового блока, тогда как высота остается неизменной. Высота определяется размером кегля, который задается на панели инспектора свойств.

Совет

Подробно о задании атрибутов текста рассказывается ниже, в разделе "Атрибуты символов".

По мере ввода текста он будет автоматически переноситься с одной строки на другую, а текстовый блок будет увеличиваться так, чтобы вмещать все содержимое. В правой части текстового поля имеется маркер, который определяет тип текстового блока и позволяет изменять

его размеры. Квадрат, расположенный в правом верхнем углу текстового блока (рис. 5.3), является маркером фиксированного блока текста и указывает на то, что задана высота текста. Круглый маркер в правом верхнем углу означает, что текстовый блок имеет неопределенные размеры, которые изменяются по мере ввода текста. Круглый маркер, расположенный в правом нижнем углу, обозначает, что текстовый блок является вертикальным. Чтобы изменить размеры текстового блока, убедитесь, что на панели инструментов выбран инструмент Text и что в текстовом блоке есть точка ввода. Разместите курсор на маркере так, чтобы он принял вид горизонтальной двунаправленной стрелки. Для изменения размеров текстового блока перетаскивайте курсор влево или вправо.

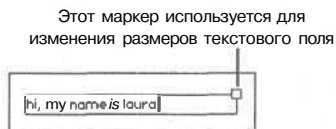


Рис. 5.3. Для изменения размеров текстового поля перетаскивайте маркер текстового блока

Внимание!

Для изменения размеров текстовых блоков не пользуйтесь полями Width (Ширина) и Height (Высота) панели инспектора свойств. Это приведет к искажению текста (его растяжению или сжатию).

ИНСПЕКТОР свойств

При работе с текстом необходимо использовать инструмент Text (Текст) или Arrow (Стрелка) в сочетании с панелью инспектора свойств. Инструмент Text определяет размещение текста, размеры текстовых блоков и используется для выделения текста в пределах текстовых блоков. Инструмент Arrow используется для изменения местоположения текстовых блоков и редактирования текстовых блоков как единого целого. На панели инспектора свойств (рис. 5.4) задают атрибуты текста. Используя версии Flash 5 панели Character (Символ) и Paragraph (Абзац) теперь заменены одной панелью инспектора свойств, что сделало редактирование текста более удобным.

Текст можно редактировать как глобально (т.е. воздействовать на текстовые блоки целиком), так и по отдельности, корректируя буквы или слова внутри текстовых блоков. Наиболее эффективным средством внесения глобальных изменений в текстовые блоки (изменение шрифта, цвета, размера или направления шрифта) является инструмент Arrow, с помощью которого следует выцепить текстовый блок. После этого внесите изменения на панели инспектора свойств, и они автоматически будут применены ко всему текстовому блоку. Однако при таком подходе нельзя вносить глобальные изменения на уровне символов, например изменять межзнаковый интервал. Чтобы отредактировать атрибуты отдельных символов, необходимо выцепить отдельный символ с помощью инструмента Text. Любые последующие изменения, сделанные на панели инспектора свойств, будут применены только к выделенным символам.

На заметку

Чтобы при редактировании текста быстро перейти от инструмента Arrow к инструменту Tool, дважды щелкните в текстовом блоке инструментом Arrow. При этом курсор примет вид точки ввода, и станет активным инструмент Text.



Рис. 5.4. Для задания и редактирования атрибутов текста воспользуйтесь панелью инспектора свойств

АТРИБУТЫ СИМВОЛОВ

Атрибуты текста можно разделить на две категории: те, что влияют на отдельные буквы или символы, и те, что применяются к абзацам. К атрибутам символов относятся выбор шрифта, кегль, направление текста, кернинг, межбуквенный интервал и расположение символов. Атрибуты можно задать как перед вводом текста в рабочее поле (для доступа к свойствам текста на панели инспектора свойств выберите инструмент Text), так и после.

Чтобы отредактировать отдельные символы, выберите инструмент Text и выделите им нужные буквы в рабочем поле, как показано на рис. 5.5. Затем измените параметры текста на панели инспектора свойств.

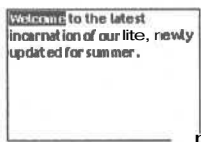


Рис. 5.5. Для редактирования отдельных СИМВОЛОВ в текстовом блоке щелкните и перетяните инструмент Text

ВЫБОР ШРИФТА И КЕГЛЯ

Чтобы выбрать шрифт, введите его имя в поле Font (Шрифт) либо щелкните на кнопке одноименного раскрывающегося меню и выберите необходимый шрифт из числа имеющихся в системе. После ввода имени шрифта вручную необходимо щелкнуть в точке ввода в рабочем поле. В противном случае Flash MX будет продолжать вводить текст в поле Font.

Аналогичным образом можно вручную ввести кегль в поле Point size (Кегль) либо выбрать его, щелкнув на кнопке раскрывающегося меню и воспользовавшись ползунком. Ввести кегль вручную гораздо легче, чем манипулировать ползунком. Я пользуюсь ползунком только тогда, когда необходимо просмотреть текст с разными размерами. В этом случае я выделяю текст и перемещаю ползунок до тех пор, пока не будет найден нужный размер.

Для выбора цвета шрифта воспользуйтесь цветовым полем. После щелчка на нем откроется палитра безопасных Web-цветов. Выберите из нее нужный цвет. Чтобы задать свой собственный цвет, не входящий в данную палитру, воспользуйтесь панелью Color Mixer (Цветовой микшер).

Совет

Подробная информация о работе с цветами во Flash и о панели Color Mixer приведена в главе 3 "Рисование во Flash". Кроме того, можно обратиться к разделу "Эффективные приемы работы с текстом" далее в этой главе.

Цвет заливки текста Flash назначает автоматически. К шрифту можно применить стилевое оформление, воспользовавшись кнопками Bold (Полужирный) и Italic (Курсив), однако при этом будет утеряна утонченность гарнитуры. Очень многие шрифты специально разработаны в версиях для различных толщины и стилей, поэтому для достижения наилучших результатов можно выбрать полужирную или курсивную версию того же самого шрифта, как показано на рис. 5.6.



Рис. 5.6. Для получения наилучшего результата вместо кнопки Bold или Italic выберите полужирную или курсивную версию шрифта

НАПРАВЛЕНИЕ ТЕКСТА



Новая функция Flash MX позволяет управлять направлением текста, что обеспечивает возможность поддержки азиатских языков. С помощью кнопки Text Direction (Направление текста), показанной на рис. 5.7, можно создать вертикальный текст, направленный слева направо или справа налево.

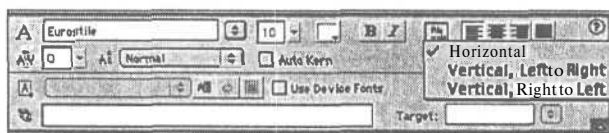


Рис. 5.7. Кнопка Text Direction позволяет выбрать горизонтальное или вертикальное направление текста

КЕРНИНГ И МЕЖЗНАКОВЫЙ ИНТЕРВАЛ

Кернингом называется интервал между двумя буквами или символами, а межзнаковым интервалом — определенный промежуток между выделенными символами или целыми текстовыми блоками. И кернинг, и межзнаковый интервал можно задавать в виде положительных и отрицательных чисел. При задании положительных значений интервал между буквами будет увеличиваться, а при задании отрицательных — уменьшаться до тех пор, пока буквы не начнут накладываться друг на друга (рис. 5.8). Нулевое значение является нейтральным. Для достижения общей удобочитаемости текста лучше всего задавать значения от 0 до +5 или даже меньше, в зависимости от размера текста. Чем меньше размер шрифта, тем больше эффект от задания межзнакового интервала. Многие шрифты сразу создаются с нужным кернингом, который можно использовать, установив флажок в поле опции Auto Kern (Автоматический кернинг).

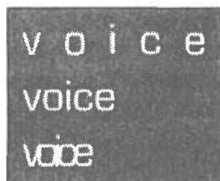


Рис. 5.8. Межзнаковый интервал: +20 — вверху, 0 — посередине и -5 — внизу

Для задания межзнакового интервала сначала выделите буквы, слова или целые текстовые блоки с помощью инструмента Text. Затем введите в поле Character Spacing (Межзнаковый интервал) положительное или отрицательное число либо щелкните на кнопке раскрывающегося меню и выберите нужное значение, перетаскивая ползунок. При работе с вертикальным текстом межзнаковый интервал определяет промежуток между символами по вертикали. Кернинг вертикального текста можно отключить. При этом значение межзнакового интервала будет применяться только к горизонтальному тексту.

Совет

Подробная информация о задании параметров приведена в главе 2 "Интерфейс Flash".

ПОЛОЖЕНИЕ СИМВОЛА

Атрибут положения символа определяет, где именно расположен текст относительно базовой линии. Базовая линия — это невидимая горизонтальная линия, на которой расположены символы шрифта (рис. 5.9). Атрибут положения символа может иметь значения Normal (Обычное), Superscript (Верхний индекс) и Subscript (Нижний индекс). В большинстве случаев используется обычное положение символов, заданное по умолчанию, при котором текст располагается точно на базовой линии. Верхний и нижний индексы, как правило, применяются к специальным символам или сноскам в пределах текстовых блоков. При выборе значения Superscript текст располагается выше базовой линии; это положение применяется к таким специальным символам, как товарный знак (™) и зарегистрированная торговая марка (®). При выборе значения Subscript текст располагается чуть ниже базовой линии.



Рис. 5.9. Базовая линия — это воображаемая линия, на которой располагаются символы. Символы, являющиеся верхним или нижним индексом, располагаются соответственно выше или ниже базовой линии

АТРИБУТЫ АБЗАЦА

Атрибуты абзаца позволяют форматировать участки текста, начиная от отдельных абзацев и заканчивая целыми текстовыми блоками. Эти атрибуты являются глобальными по своей

природе и применяются не к отдельным буквам или словам, а к целым блокам текста. К параметрам форматирования абзацев относятся выравнивание, задание полей и отступов.

Чтобы отформатировать абзац, выделите его целиком с помощью инструмента Text. Чтобы отформатировать весь текстовый блок, выделите его с помощью инструмента Arrow.

ВЫРАВНИВАНИЕ

Для выравнивания абзацев по левому или правому краю, по центру или по всей ширине воспользуйтесь кнопками Alignment (Выравнивание). При выравнивании по ширине создаются абзацы газетного типа, т.е. выровненные одновременно и по правому, и по левому краю. При выполнении этой операции обязательно проверьте, чтобы в результате расстояние между словами не было слишком большим. Опции выравнивания применимы и к вертикальному тексту.

При работе с растровыми шрифтами используйте заданное по умолчанию выравнивание по левому краю. При выравнивании по правому краю, по центру и по ширине растровые шрифты могут быть позиционированы на нецелых пикселях, что приведет к размытию букв. Самым опасным является выравнивание по ширине, при котором шрифты могут быть растянуты на дробные координаты, даже при включенной функции Snap to Pixels (рис. 5.10).

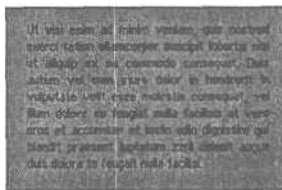


Рис. 5.10. Не выполняйте выравнивание по ширине при работе с растровыми шрифтами, так как они могут растянуться и стать нечеткими

ОТСТУПЫ, МЕЖДУСТРОЧНЫЙ ИНТЕРВАЛ И ПОЛЯ

Для задания отступов, междустроочного интервала и полей щелкните на кнопке Format (Формат), в результате чего откроется диалоговое окно Format Options (Параметры форматирования), показанное на рис. 5.11. Значения в пикселях можно ввести вручную либо задать их, перемещая ползунок. Внесенные изменения сразу же будут применены к тексту, выделенному в рабочем поле. После задания нужных параметров щелкните на кнопке Done (Готово). К сожалению, в этом диалоговом окне отсутствует кнопка Cancel (Отмена), поэтому, перед тем как щелкнуть на кнопке Done, еще раз проверьте заданные настройки. Чтобы вернуться к предыдущим параметрам форматирования, нажмите комбинацию клавиш <Cmd+Z> (Macintosh) или <Ctrl+Z> (Windows).

В поле Indent (Отступ) диалогового окна Format Options указывают величину отступа (в пикселях) от начала каждого абзаца. По умолчанию она задана равной нулю, т.е. отступ отсутствует. Отступы применяются для усиления визуального эффекта отделения одного абзаца

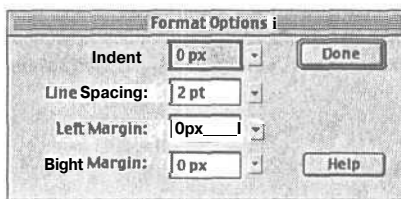


Рис. 5.11. В диалоговом окне Format Options можно указать атрибуты абзацев

от другого. При вводе в поле *Indent* положительного числа первая строка абзаца будет сдвинута вправо. При вводе отрицательного значения создается обратный отступ, когда первая строка абзаца располагается левее всех остальных, как показано на рис. 5.12. Задавать отрицательное значение отступа имеет смысл, только если задано левое поле, в противном случае слева от текстового блока просто не будет места для создания "висячей" строки.



*Рис. 5.12. При задании в поле *Indent* отрицательного значения создается обратный отступ*

Параметр *Line Spacing* (Междустрочный интервал) указывает промежуток (в пунктах) между строками текста. Этот параметр является важным для удобочитаемости текста. Если междустрочный интервал будет слишком маленьким, текст будет очень сжатым, а чтение его — затруднительным. Слишком большой междустрочный интервал приведет к тому, что строки текста будут казаться плавающими. Это также понижает его удобочитаемость. По умолчанию междустрочный интервал составляет 120% и в целом применим для основного текста. Крупный текст обычно требует меньшего междустрочного интервала, так что можете поэкспериментировать с данным параметром и найти наиболее подходящее для каждого конкретного случая значение.

Для параметров *Left Margin* (Левое поле) и *Right Margin* (Правое поле) указываются значения отступа с каждой стороны текстового блока. По умолчанию эти параметры имеют нулевые значения. Пользователь может задать любое значение в пикселях, увеличивая тем самым расстояние с каждой стороны от текстового блока, как показано на рис. 5.13.

ПАРАМЕТРЫ ТЕКСТА

При работе с *Flash* пользователь не ограничен только статическим текстом, т.е. таким, который отображается в опубликованных фильмах точно в том виде, в каком он был введен в файле *FLA*. Пользователь может вводить текст в поля ввода данных, а кроме того, текст можно добавлять к фильму динамически в процессе воспроизведения. До сих пор мы рассматривали только статический текст, который не разрешается вводить и обновлять в фильме, — все изменения должны вноситься в файле *FLA*. Поля ввода данных и динамического текста можно заполнять в процессе воспроизведения фильма, а работа с таким текстом осуществляется с помощью языка *ActionScript*. При работе с вводимым и динамическим текстом в файле *FLA* необходимо создать заполнители текстовых полей, указывающие, где должен быть размещен текст и каким образом он будет появляться при воспроизведении фильма. Атрибуты вводимого и динамического текста, как и атрибуты статического текста, назначаются на панели инспектора свойств.

Вводимый ТЕКСТ

Вводимый текст вставляется (как при заполнении формы) в процессе воспроизведения фильма. Полям вводимого текста поставлены в соответствие имена переменных, чтобы вводимые в них данные можно было сохранить и использовать с помощью *ActionScript*.

Если вы хотите работать с вводимым текстом, убедитесь, что является активным инструмент **Text**, и на панели инспектора свойств из раскрывающегося меню **Text type** (Тип текста) выберите пункт **Input Text** (Вводимый текст). В рабочем поле необходимо создать текстовое поле, в которое будет вводиться текст, или преобразовать в него уже существующий текстовый блок, выделив его и изменив значение типа текста на **Input Text**. Границы полей вводимого текста представляют собой пунктирные линии. В правом нижнем углу расположен маркер, обозначающий блок вводимого или динамического текста. На панели инспектора свойств назначьте атрибуты текста: шрифт, кегль и цвет, точно так же, как это вы делали при работе со статическим текстом. Чтобы увидеть, как будет выглядеть текст при вводе пользователем, наберите что-нибудь в поле ввода (рис. 5.14).

Очень важно сообщить пользователям, где необходимо щелкнуть мышью для ввода текста. Если в макете графически не указано, где именно пользователь должен вводить текст, включите отображение границ поля вводимого текста, щелкнув на кнопке **Show Borders** (Показать границы). Однако знайте, что при включении отображения границ поле вводимого текста будет представлять собой черный контур с белым фоном. Если такой вид нежелателен, то поле вводимого текста можно поместить поверх графического элемента, обозначающего границы ввода, как показано на рис. 5.15. Альтернативный способ — задать цвет фона и границ с помощью **ActionScript**.

ДИНАМИЧЕСКИЙ ТЕКСТ

Динамический текст загружается из внешнего источника в процессе воспроизведения фильма. "Динамическим" называется содержимое, выбираемое в процессе воспроизведения, а не составленное заранее в файле **FLA**. Это — потрясающая возможность отображения со-

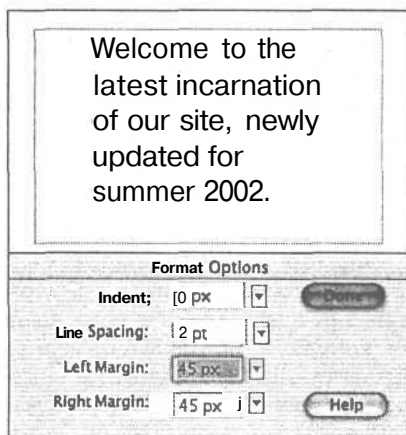


Рис. 5.13. Параметры *Left Margin* и *Right Margin* позволяют увеличить расстояние по обе стороны текстового блока



Рис. 5.14. Чтобы увидеть, как будет выглядеть текст, вводимый пользователем, введите в поле какое-нибудь слово

держимого, которое должно часто меняться (например, статьи новостей), поскольку при этом отпадает необходимость редактирования файла FLA. На рис. 5.16 видно, что текстовому полю поставлена в соответствие переменная, назначающая источник динамического содержимого.

Для добавления динамического текста выполните следующие действия.

1. Из раскрывающегося меню Text type выберите пункт Dynamic Text (Динамический текст).
2. Создайте поле динамического текста в рабочем поле.
3. Как и при работе с полями вводимого текста, границы полей динамического текста отображаются в рабочем поле в виде пунктирных линий. Маркер текстового блока расположен в правом нижнем углу и свидетельствует о наличии поля вводимого или динамического текста.

Чтобы увидеть, как будет выглядеть содержимое, введите что-нибудь в поле динамического текста.

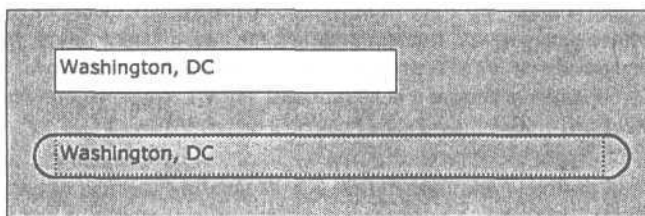


Рис. 5.15. Чтобы указать границы поля вводимого текста без отображения белого фона, поместите поле поверх графического элемента

ПАРАМЕТРЫ ФОРМАТИРОВАНИЯ ТЕКСТА

К вводимому и динамическому тексту применимо несколько специальных опций форматирования. Для этих текстовых полей необходимо задать имена экземпляров, что позволяет получить доступ к свойствам с помощью ActionScript. Для этого введите уникальное имя экземпляра в поле Instance Name (Имя экземпляра), расположенное под полем Text type на панели инспектора свойств (рис. 5.16).

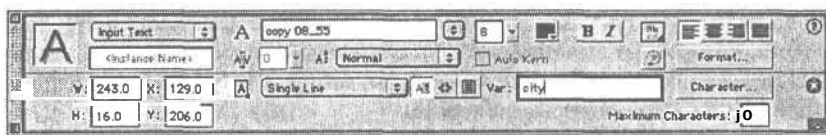


Рис. 5.16. Имена переменных позволяют получить доступ к содержимому текстовых полей с помощью ActionScript

Совет

Подробно об использовании ActionScript для доступа к текстовым полям рассказывается в главе 16 "Взаимодействие, события и установление последовательности" и в главе 20 "Использование встроенных объектов фильмов".

Имена переменных позволяют программным способом получить доступ к текстовым полям с целью выполнения вычислений, а также к содержимому, вводимому пользователем в текстовые поля ввода, или назначить содержимое динамических текстовых полей. Чтобы назначить имя переменной для текстового поля, введите уникальное имя в поле Variable (Переменная), как показано на рис. 5.16.

С помощью раскрывающегося меню Line Type (Тип строки) можно указать, разрешается ли вводить в текстовое поле больше одной строки. Из данного меню можно выбрать значение Multiline (Несколько строк), разрешающее вводить несколько строк текста, или Multiline No Wrap (Несколько строк без переноса), что позволит отображать несколько строк только в случае ввода пользователем символа жесткого переноса посредством клавиши <Return> (Macintosh) или <Enter> (Windows).

Поле Maximum Characters (Максимальное количество символов), расположенное в правом нижнем углу панели инспектора свойств (см. рис. 5.16), позволяет ограничить количество символов, вводимых в текстовое поле.

После щелчка на кнопке Render as HTML (Отображать как HTML) HTML-форматирование текста будет сохранено, если текстовым полям поставлены в соответствие значения переменных или экземпляров, содержащих такое форматирование. Если данная опция включена, то при экспортировании SWF-файлов Flash применяет к тексту соответствующие дескрипторы HTML. Во Flash поддерживаются HTML-дескрипторы ссылок, полужирного и курсивного начертания, подчеркивания, абзаца, а также цвета, гарнитуры и размера шрифта. Поддерживаются HTML-атрибуты левого и правого поля, отступа и междустрочного интервала. Отображение текста с сохранением HTML-форматирования можно задать и программным способом, с помощью свойстваhtmlобъектаTextField.

На панели инспектора свойств находится кнопка, позволяющая сделать динамический текст доступным для выбора пользователями. По умолчанию данная опция включена, но если вы не хотите, чтобы пользователи могли выделять текст, например для его копирования и вставки, отожмите кнопку Selectable Text (Выделяемый текст) (рис. 5.17).



Рис. 5.17. Если отжать кнопку Selectable Text, то пользователи не смогут копировать и вставлять динамический текст

ВНЕДРЕННЫЕ И ВСТРОЕННЫЕ ШРИФТЫ

При создании текста в программе Flash в распоряжении дизайнера имеются все шрифты, установленные в системе. Однако для различных платформ доступны разные шрифты, и не все пользователи имеют шрифты, доступные в вашей системе. Означает ли это, что форматирование текста, которое вы выполнили столь старательно, просто пропадет? Этого не произойдет, но следует предпринять некоторые шаги, которые позволят другим пользователям увидеть именно те шрифты, которые вы хотите.

Существуют два способа. Одним из них является внедрение начертания шрифтов, которое позволит сохранить начертание шрифтов, даже если они не установлены в другой системе. Flash будет сглаживать внедренные шрифты. Внедренные шрифты несколько увеличат общий размер файла фильма, но в некоторых случаях в заданный шрифт можно внедрять не все символы.

При работе со статическим текстом шрифты внедрять не нужно, поскольку начертание сохраняется автоматически. Шрифты, использующиеся в полях вводимого и динамического текста, не внедрены по умолчанию, и, следовательно, они не сглаживаются. Для внедрения шрифтов выполните следующие действия.

1. В рабочем поле выберите поле динамического или вводимого текста.
2. На панели инспектора свойств щелкните на кнопке Edit Character Options (Редактировать параметры символов). Откроется диалоговое окно Character Options (рис. 5.18), позволяющее указать параметры внедрения шрифтов.

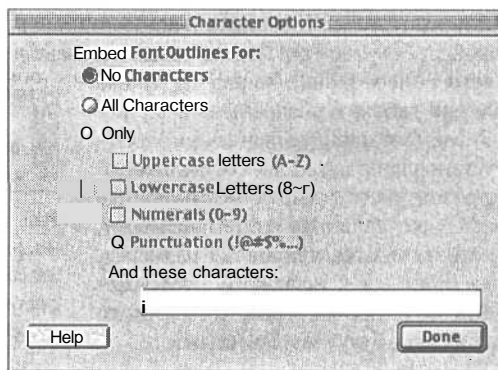


Рис. 5.18. В диалоговом окне *Character Options* можно указать, внедрено ли начертание шрифтов для полей динамического и вводимого текста

В диалоговом окне можно выбрать следующие опции внедрения шрифтов: отсутствует (No Characters), для всех символов (All Characters) или только для определенных символов (Only). Таким образом, Flash помогает несколько уменьшить размер файла, позволяя заранее спланировать, для каких символов в заданном текстовом поле потребуются внедренные контуры шрифтов. Например, для текстового поля, в которое будет вводиться год (в виде четырех цифр), не нужно внедрять никакие символы, кроме цифр. Поэтому для текстового поля даты в диалоговом окне *Character Options* установите переключатель в поле опции *Only* (Только), а затем установите флажок в поле опции *Numerals* (Цифры).

Далеко не все шрифты могут быть внедрены и экспортированы вместе с фильмом. Иногда Flash не распознает данные о начертании шрифта. Чтобы проверить, можно ли внедрить тот или иной шрифт, выделите текст и выполните команду *View⇒Antialias Text* (Вид⇒Сгладить текст). Если шрифт будет отображен с зазубренными краями, это значит, что Flash не распознает его начертание и не сможет экспортировать.

Второй способ — это встраивание шрифтов. Операционные системы имеют ограниченное число автоматически установленных шрифтов. Для систем Macintosh и Windows эти шрифты различны, но в каждой из платформ имеются шрифты с засечками, без засечек и машинописные. В шрифте с засечками вдоль основного контура символа располагаются декоративные линии (рис. 5.18), а в шрифтах без засечек они отсутствуют. Машинописные шрифты являются моноширинными, т.е. все буквы имеют одинаковую ширину. Обратите внимание, что в машинописном шрифте и буква "A", и буква "i" имеют одинаковую ширину (рис. 5.19, справа). Сравните это с примерами шрифтов с засечками и без засечек.



Рис. 5.19. Среди встроенных шрифтов есть шрифты с засечками, без засечек и моноширинные

При выборе встроенных шрифтов они не внедряются, а вместо этого надстройка *Rash Player* отображает шрифты, максимально похожие на те, что установлены в системе пользователя. Можно задать три типа встроенных шрифтов: *_sans* (без засечек), при выборе которого обычно используется шрифт *Helvetica* или *Arial*; *_serif* (с засечками), при выборе которого обычно используется шрифт *Times* или *Times New Roman*, или *_typewriter* (машинописный), при выборе которого ис-

пользуются шрифты, аналогичные Courier. Для задания встроенных шрифтов выберите соответствующий пункт из раскрывающегося меню Font (Шрифт), как показано на рис. 5.20.

При использовании встроенных шрифтов размер файла фильма будет меньше, чем при работе с внедренными шрифтами, но здесь присутствует элемент неопределенности. Если в системе пользователя не установлены шрифты, соответствующие указанным, могут произойти странные и неожиданные замены. Встроенные шрифты не являются сглаживаемыми, поэтому лучше всего они отображаются при малых размерах, когда сглаживание обычно приводит к нечеткому отображению. Поскольку эти шрифты не являются сглаживаемыми, то при выборе большого кегля их края будут зазубренными.



Рис. 5.20. Чтобы задать тип встроенных шрифтов, выберите соответствующий пункт из раскрывающегося меню Font на панели инспектора свойств

РЕДАКТИРОВАНИЕ И ПРЕОБРАЗОВАНИЕ ТЕКСТА

При редактировании текста во Flash можно применять базовые методики обработки текста. Текст во Flash можно копировать или вырезать, а затем вставлять в другие приложения. К сожалению, функция проверки орфографии во Flash отсутствует. Чтобы выполнить такую проверку, текст из Flash нужно скопировать в текстовый редактор, а затем скопировать обратно (если в него будут внесены какие-то изменения). Форматирование текста при этом сохраняется. При копировании текста обратно во Flash обязательно выделите исходный фрагмент и замените его, иначе будет создан новый блок скопированного текста, и он будет размещен в другом месте и иметь другие размеры по сравнению с оригиналом.

Для выделения символов внутри текстового блока используется инструмент Text. Для выделения нескольких символов щелкните и перетащите курсор, а для выделения целого слова дважды щелкните на нем. Чтобы выделить большой фрагмент без перетаскивания курсора, щелкните в начале фрагмента, а затем — в конце, удерживая нажатой клавишу <Shift>. Чтобы выделить все слова в текстовом блоке, воспользуйтесь комбинацией клавиш <Cmd+A> (Macintosh) или <Ctrl+A> (Windows).

Для выделения текстового поля с целью его перемещения или копирования воспользуйтесь инструментом Arrow. Если при щелчке этим инструментом удерживать нажатой клавишу <Shift>, можно выделить несколько текстовых полей. Чтобы скопировать и переместить текстовый блок целиком, щелкните на нем, удерживая нажатой клавишу <Option> (Macintosh) или <Alt> (Windows), а затем переместите блок, воспользовавшись инструментом Arrow.

РАЗБИВКА ТЕКСТА



Во Flash MX имеется возможность разбить текст и поместить каждый его символ в отдельный текстовый блок. Затем текст можно преобразовать в кривые или применить анимацию к отдельным символам. Во Flash 5 при разбивке текста он сразу преобразовывался в кривые, а во Flash MX при однократном выполнении команды разбивки каждый символ текста помещается в отдельный текстовый блок. Для разбивки текста выполните следующие действия.

1. Выделите текстовый блок с помощью инструмента Arrow.
2. Выполните команду **Modify**⇒**Break Apart** (**Изменить**⇒**Разбить**). Чтобы преобразовать текстовые блоки символов в кривые, выполните эту команду еще раз (рис. 5.21).

После того как текст будет преобразован в кривые, он больше не будет распознаваться и его нельзя будет редактировать как текст.

После выполнения разбивки текста Flash MX позволяет быстро разнести блоки символов с целью разделения слоев. Выделите текстовые блоки с помощью инструмента Arrow, а затем выполните

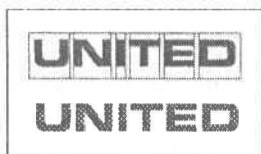
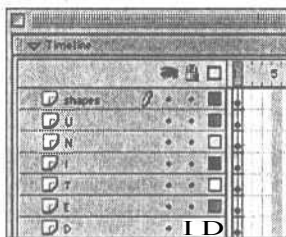


Рис. 5.21. При первом выполнении команды разбивки каждый символ текста будет помещен в отдельное текстовое поле. При ее повторном выполнении текстовые блоки будут преобразованы в кривые



*Рис. 5.22. После выполнения команды *Distribute to Layers* каждый блок символа будет помещен в отдельный слой*

команду **Modify**⇒**Distribute to Layers** (Изменить⇒Распределить по слоям). Flash поместит каждую букву в отдельный **слой**, имя которого будет соответствовать данной букве, как показано на рис. 5.22.

Присвоение имен распределенным слоям

Каждому распределенному слою присваивается имя, соответствующее букве, которую он содержит. Если в слове эта буква встречается несколько раз, вы получите несколько слоев с одинаковыми именами. Так можно очень быстро запутаться, поэтому включайте в имена слоев цифры или же каким-то другим образом создайте отличия между именами слоев, содержащих одинаковые буквы.

ПРЕОБРАЗОВАНИЕ ТЕКСТА

После того как текст разбит на символы и преобразован в кривые, его можно трансформировать и сделать невероятно впечатляющим. На рис. 5.23 видно, что с помощью инструмента **Subselection** (Частичное выделение) можно выделить точки и переместить их, изменяя форму букв. Такие преобразования можно использовать для создания букв уникальной формы и их последующей анимации с помощью промежуточных изображений.

Совет

Подробно о создании анимации и о промежуточных изображениях речь пойдет в главе 7 "Анимация во Flash".



*Рис. 5.23. После того как текст разбит и преобразован в кривые, с помощью инструмента *Subselection* можно перетаскивать анкерные точки и изменить форму букв*



Чтобы сдвинуть контуры текста, воспользуйтесь новыми модификаторами Distort (Искривление) и Envelope (Огибающая). Их нельзя применить к группам или символам, поэтому текст обязательно нужно преобразовать в кривые. Выберите либо инструмент Arrow, либо Free Transform и перетащите его поверх слова, чтобы оно преобразовывалось как единое целое. Если вы работаете с инструментом Arrow, выберите инструмент Free Transform, а затем в нижней части панели инструментов выберите модификатор Distort или Envelope. Перетащите маркеры преобразования, окружающие слово, как показано на рис. 5.24.

Самое впечатляющее преобразование можно выполнить с помощью модификатора Envelope, полностью изменив форму букв (рис. 5.25). Пользуйтесь этим мощным средством осторожно, потому что буквы могут стать настолько бесформенными, что их нельзя будет узнать и прочитать.

ЭФФЕКТИВНЫЕ ПРИЕМЫ РАБОТЫ С ТЕКСТОМ

Самым важным критерием при работе с текстом является его последующая удобочитаемость. Прежде всего, текст предназначен для донесения определенной информации. Если его нельзя прочитать, информация не будет донесена до пользователя. Поэтому самой важной задачей является создание текста, доступного и легкого для чтения.

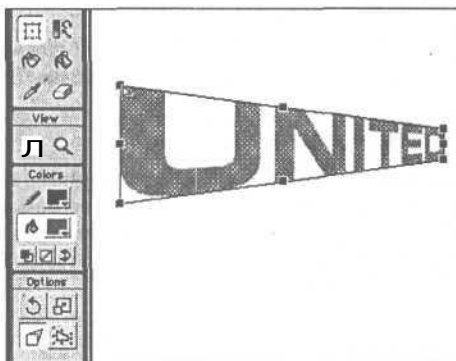


Рис. 5.24. Модификатор Distort используется для создания эффекта перспективы

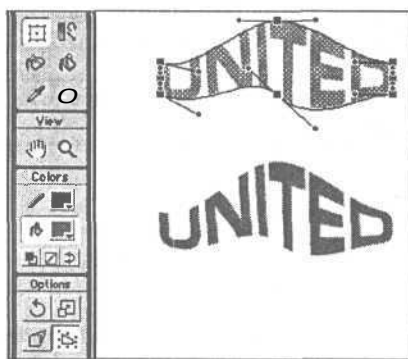


Рис. 5.25. Чтобы создать наиболее впечатляющие эффекты, воспользуйтесь модификатором Envelope

На удобочитаемость текста влияют следующие атрибуты:

- контраст цвета текста и фонового цвета;
- шрифты с засечками;
- кегль;
- межбуквенный и междустрочный интервал.

Эти атрибуты особенно важны при работе с растровыми шрифтами. Очень часто кегль растровых шрифтов задан предварительно и является небольшим. Поэтому наличие существенного контраста цвета шрифта и фонового цвета является еще более важным. Темный текст на светлом фоне гораздо легче прочитать, хотя можно использовать и светлый текст на темном фоне, но только в том случае, если цвета хорошо контрастируют. Старайтесь не комбинировать текст и фоновый цвет, имеющие похожую яркость, особенно если шрифт имеет небольшой размер, как, например, растровый.

Кроме того, не следует использовать шрифты с засечками небольших размеров. Засечки представляют собой очень тонкие линии, гораздо тоньше, чем основной штрих буквы, а если шрифт будет очень маленького размера, то эти различия будет очень трудно заметить. Сглаживание контурных неровностей еще сильнее обостряет эту проблему. На Web-узле Minimi предлагается растровый шрифт с засечками *Serif*, который более удобочитаем при малых размерах.

Совет

Подробная информация о шрифтах, доступных на Web-узле Minimi, и о других растровых шрифтах приведена выше, во врезке "Ресурсы растровых шрифтов".

Если вы работаете не с растровыми шрифтами, то можете управлять размером текста. Просмотрите текст, применяя к нему разные размеры шрифта и параметры форматирования, и подберите оптимальный размер, межбуквенное и междустрочное расстояние. Дайте прочитать текст другому человеку, который незнаком с содержанием, и проверьте, удобно ли ему читать ваш текст.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему растровые шрифты, размещенные на целых координатах пикселей, во фрагменте фильма, будучи помещенными на основную временную шкалу, становятся размытыми?

Проверьте, чтобы фрагмент фильма был помещен на целую координату в основной временной шкале, поскольку это размещение определяет координаты текста, внедренного во фрагмент фильма. Даже если текст во вложенном фрагменте размещен на целых координатах (х.0), но в основной временной шкале фрагмент фильма помещен на дробную координату (например, х.5), то текст фактически будет располагаться на дробной координате: $x.0 + x.5 = x.5$.

Кроме того, при публикации фильма, содержащего растровые шрифты, во вкладке HTML диалогового окна Publish Settings (Параметры публикации) из раскрывающегося меню Scale (Масштабирование) обязательно выберите пункт No Scale (Без масштабирования). Если фильм разрешено масштабировать, его размеры определяются размерами окна браузера, что приведет к растягиванию или сжатию SWF-файла. Текст также будет растягиваться или сжиматься и в результате будет отображен с размерами, отличными от тех, что были заданы в файле FLA. Растровые шрифты отображаются в виде определенного числа пикселей, поэтому они должны иметь точные размеры, чтобы их визуализация была корректной.

Особенно проблематичным является использование растровых шрифтов в компонентах. Дело в том, что фрагменты фильма в пределах компонентов, содержащих текст, не размещены на целых пикселях. Бренден Холл (Branden Hall) написал сценарий, позволяющий обойти размещение пикселей. Его можно загрузить с узла http://www.waxpraxis.org/entry_blog-15.html.

Почему нельзя использовать полужирный и курсивный шрифт в полях динамического текста?

При применении HTML-форматирования к внедренным шрифтам могут возникнуть проблемы. Полужирные и курсивные шрифты имеют отдельные начертания, которые также необходимо внедрять, а это приведет к существенному увеличению размера фильма — данные о шрифтах, содержащихся в фильме утратятся. Однако можно пойти на некоторые хитрости. Необходимо создать текстовое поле для указания каждого начертания. Это можно сделать путем создания псевдотекстовых полей, указывающих альтернативные начертания (одно — для полужирного шрифта, другое — для курсивного и т.д.), и их размещения за пределами видимой области рабочего поля, чтобы эти текстовые поля не были видны в конечном фильме. В некоторых случаях число внедряемых символов можно ограничить, в частности, если известны типы символов, которые будут использоваться в полях динамического текста. Тем не менее внедрение дополнительных начертаний существенно увеличит размер файла фильма.

Более простым решением будет указание для поля динамического текста встроенных шрифтов. Это гарантирует корректное отображение HTML-текста и не увеличивает конечный размер файла. Вы будете в меньшей степени контролировать способ отображения, но сэкономите на размере файла и будете твердо уверены в правильном отображении HTML-текста. Поэтому стоит пойти на такой компромисс.

FLASH ЗА РАБОТОЙ: КОПИРОВАНИЕ ТЕКСТОВЫХ ПОЛЕЙ

Если необходимо создать несколько полей вводимого или динамического текста, имеющих одинаковые размеры и атрибуты, то проще всего будет создать одно текстовое поле и скопировать его нужное количество раз. Однако скопированные текстовые поля будут иметь одинаковые атрибуты, *в том числе* и имена экземпляров и переменных. Это внесет полную неразбериху в сценарий. Поэтому вы обязаны присвоить уникальные имена экземплярам и переменным для каждого текстового поля.

Если при отладке сценария для нескольких текстовых полей вы не можете найти никакие ошибки, проверьте имена экземпляров и переменных на наличие повторов. Поскольку на панели инспектора свойств одновременно могут быть отображены атрибуты только одного текстового поля, для просмотра всего списка переменных воспользуйтесь панелью Movie Explorer (Проводник фильма), которая показана на рис. 5.26, или вкладкой Variables (Переменные) панели Debugger (Отладчик).

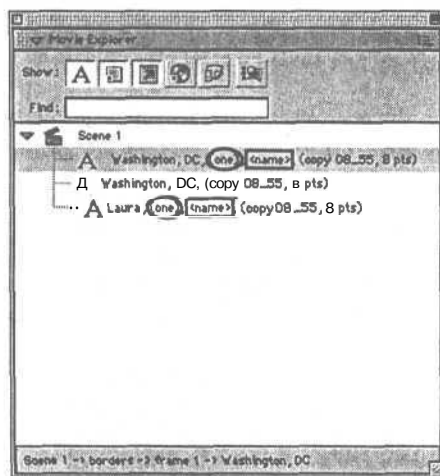
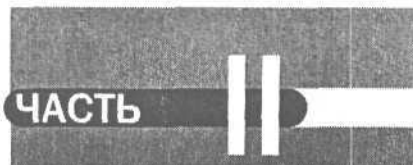


Рис. 5.26. На панели Movie Explorer видно, что два текстовых поля имеют одинаковые имена экземпляров



АНИМАЦИЯ и ЗВУК

В ЭТОЙ ЧАСТИ...

Глава 6. Символы, экземпляры и элементы библиотек

Глава 7. Анимация во Flash

Глава 8. Использование звука

Глава 9. Кнопки, меню и поля ввода данных

Глава 10. Интеграция видео

СИМВОЛЫ, ЭКЗЕМПЛЯРЫ И ЭЛЕМЕНТЫ БИБЛИОТЕК

В ЭТОЙ ГЛАВЕ...

Понятие символов	117
Создание символов	119
Редактирование символов	123
Экземпляры	129
Библиотека	132
Возможные проблемы	134
Flash за работой: вложение символов	135

ПОНЯТИЕ СИМВОЛОВ

Краеугольными камнями Macromedia Flash и основными строительными блоками создаваемых фильмов являются символы. Всякий раз при импортировании растрового изображения, рисовании контура или анимации объекта Flash должна сохранять данные, необходимые для отображения этих элементов в опубликованном фильме. При работе с любым содержанием, предназначенным для размещения в Web, стратегической задачей является достижение разумного компромисса между уменьшением размера файла и увеличением фактического содержимого. Этот вопрос приходится решать постоянно. Символы представляют собой хранящиеся образцы, т.е. объекты, которые можно использовать нужное число раз.

Символы хранятся в библиотеке, а их экземпляры перетаскивают из библиотеки в рабочее поле. Каждый экземпляр символа, расположенный в рабочем поле, является копией, независимой от своего оригинала. Каждым экземпляром можно совершенно свободно манипулиро-

вать, не изменяя при этом оригинал. Поскольку хранится только оригинал, повторно применяющийся при использовании каждого экземпляра, то можно создать любое необходимое количество экземпляров, что окажет лишь минимальное воздействие на размер файла. Использование символов ускоряет процесс воспроизведения фильма, поскольку надстройке Flash Player необходимо загрузить символ только один раз.

Символы можно считать конструктивными элементами Rash. Они не только способствуют уменьшению размера файла, но и при правильном использовании существенно повышают эффективность работы дизайнера. Основная задача заключается в том, чтобы научиться разделять элементы фильма на составляющие конструктивные блоки, которые можно связывать друг с другом с целью создания сложной анимации и системы интерфейса. Когда необходимо внести изменения, их можно сделать всего в одном месте, воздействуя на многие части фильма.

В символ можно преобразовать любой объект рабочего поля. Помимо того, что повторное использование элемента позволяет сэкономить на размере файла, преобразование любого элемента, который когда-либо можно будет использовать повторно, в символ позволяет сэкономить затрачиваемое на работу время и тем самым оптимизировать рабочий процесс. Существуют три типа символов: графические, кнопки и видеоклипы. Каждый из них имеет собственную временную шкалу, рабочее поле и слои.

ГРАФИЧЕСКИЕ СИМВОЛЫ

Графические символы — наиболее простые и наименее мощные. Они привязаны к основной временной шкале и лучше всего подходят для статических изображений, например для фона. Графические символы могут быть анимированными, но не могут содержать звуки или интерактивные элементы. Кроме того, поскольку графические символы управляются с помощью основной временной шкалы, они будут воспроизводиться только в течение кадра в основной временной шкале. Для создания анимации в распоряжении пользователя имеются видеоклипы, поэтому графические символы лучше приберечь для повторно используемых статических элементов.

Кнопки

Кнопки являются интерактивными символами, которые реагируют на определенные действия пользователя, например на щелчок мышью или наведение указателя на ролловер. На временной шкале кнопок (рис. 6.1) имеются кадры Up, Over, Down и Hit, соответствующие различным состояниям кнопок. Временная шкала кнопок не разворачивается во времени, подобно другим временным зависимостям. Вместо этого в ней происходит перемещение между четырьмя кадрами в ответ на действия, предпринятые пользователем.

Состояние Up (Отжата) определяет, как будет выглядеть кнопка в рабочем поле без каких-либо действий со стороны пользователя. Over (Поверх) соответствует состоянию ролловера в случае, если пользователь наводит указатель мыши на кнопку. Down (Нажата) — это состояние или внешний вид кнопки, когда пользователь щелкает на ней. Состояние Hit (Активная зона) позволяет установить физические границы кнопки, задавая область, которая будет давать отклик на действия пользователя. Содержимое кадра Hit в рабочем поле видно не будет. Для кнопок не обязательно создавать все четыре состояния. Состояния кнопок могут содержать графические элементы, звуки и даже видеоклипы.

Совет

Подробная информация о кнопках представлена в главе 9 "Кнопки, меню и поля ввода данных".

ВИДЕОКЛИПЫ

Видеоклипы представляют собой самодостаточные мини-фильмы. При необходимости для них можно задать собственную, независимую временную шкалу, не привязанную к основной. Возможность размещения одной временной шкалы в пределах другой является

"изюминкой" Flash и основой комплексной интерактивности и многострочного потока данных. Основную временную зависимость можно считать головным видеоклипом. Видеоклипы могут содержать графику, кнопки, интерактивные элементы управления, звуки, видео и даже другие вложенные видеоклипы.

Совет

Об объекте Movie Clip (Видеоклип) и его использовании с ActionScript рассказывается в главе 17 "Демонстрация мощи видеоклипов".

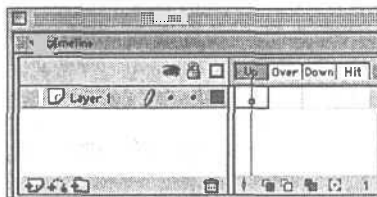


Рис. 6.1. Кнопки имеют уникальную временную шкалу с состояниями Up, Over, Down и Hit

СОЗДАНИЕ СИМВОЛОВ

С чего же начать создание фильмов Flash? Сначала нужно создать конструктивные элементы определенного фильма.

Символы можно создать с нуля или преобразовать в них объекты, расположенные в рабочем поле. Чем больше вы работаете во Flash, тем лучше сможете планировать создание символов. Если перед созданием какого-то объекта вы знаете, что его нужно будет использовать неоднократно, то лучше всего сразу создать символ, а затем создать объект как символ.

Может случиться и так, что сначала вы создадите элемент фильма, а только потом поймете, что его нужно будет использовать неоднократно. В этом случае созданный элемент можно легко преобразовать в символ.

Чтобы создать новый пустой символ, проверьте, чтобы в рабочем поле не было ничего выделено, а затем выполните команду **Insert**⇒**New Symbol** (Вставка⇒Новый символ) или воспользуйтесь комбинацией клавиш <Cmd+F8> (Macintosh) или <Ctrl+F8> (Windows). В результате откроется диалоговое окно **Create New Symbol** (Создать новый символ), показанное на рис. 6.2.

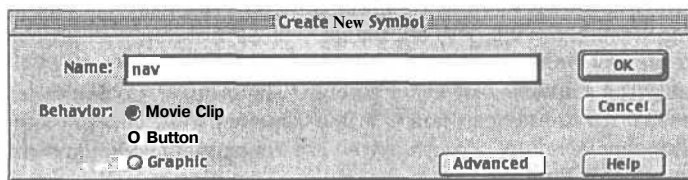


Рис. 6.2. В диалоговом окне *Create New Symbol* можно задать имя и тип символа

Введите имя символа и выберите его тип: **Movie Clip** (Видеоклип) (задан по умолчанию); **Button** (Кнопка) или **Graphic** (Графика). После этого щелкните на кнопке **OK**. Символ будет добавлен в библиотеку и в режиме редактирования откроется в рабочей области. На рис. 6.3 видно, что имя символа появится в информационной строке, расположенной сверху рабочего поля, а посередине рабочего поля появится значок перекрестия, обозначающий точку регистрации символа. Теперь к символу можно добавить содержимое.

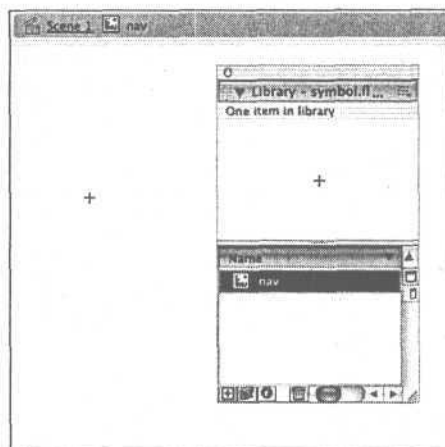


Рис. 6.3. Новый созданный символ помещается в библиотеку и открывается в режиме редактирования

На заметку

В процессе работы всегда ищите возможность преобразовать объект в составляющие части, которые затем можно неоднократно использовать. Символы повышают эффективность работы, поэтому старайтесь использовать их в полном объеме.

РЕГИСТРАЦИЯ

Точка регистрации — это контрольная точка, используемая при размещении символа цифровым образом. Существует девять стандартных точек регистрации: каждый из четырех углов, центральная точка и средняя точка с каждой стороны объекта. Точка регистрации определяет координату 0,0 (0,x и 0,y) символа.

Совет

Процедура изменения регистрационной точки подробно описана ниже, в разделе "Редактирование точек регистрации".

Система координат представляет собой двумерную плоскость, позволяющую располагать объекты в определенных положениях. Во Flash используется обратная прямоугольная система координат, где координата 0,0 (x,y) обозначает верхний левый угол рабочего поля или экрана (в опубликованном фильме). Ось x расположена по горизонтали, а ось y — по вертикали. Чтобы не возвращаться к школьному курсу алгебры, напомним, что при перемещении объекта вправо увеличивается его координата x, а при перемещении объекта вниз увеличивается координата y. При перемещении объекта влево его координата x уменьшается и после пересечения оси y принимает отрицательные значения, а сам объект смещается за пределы рабочего поля (или экрана). При перемещении объекта вверх уменьшается его координата y и после пересечения оси x принимает отрицательные значения, а сам объект смещается за пределы рабочего поля (или экрана). Прямоугольная система координат изображена на рис. 6.4.

Чтобы точно разместить объекты в рабочем поле, необходимо знать внутреннюю регистрационную точку каждого объекта. Например, если в верхнем левом углу рабочего поля (координаты 0,0) необходимо разместить квадрат размером 10x10 пикселей, нужно указать, какая часть квадрата должна быть помещена в точку с координатами 0,0. Если точка регистрации находится в центре квадрата, то именно она и будет помещена в точку с координатами 0,0. В этом случае большая часть квадрата окажется за пределами рабочего поля, как показано на рис. 6.5.

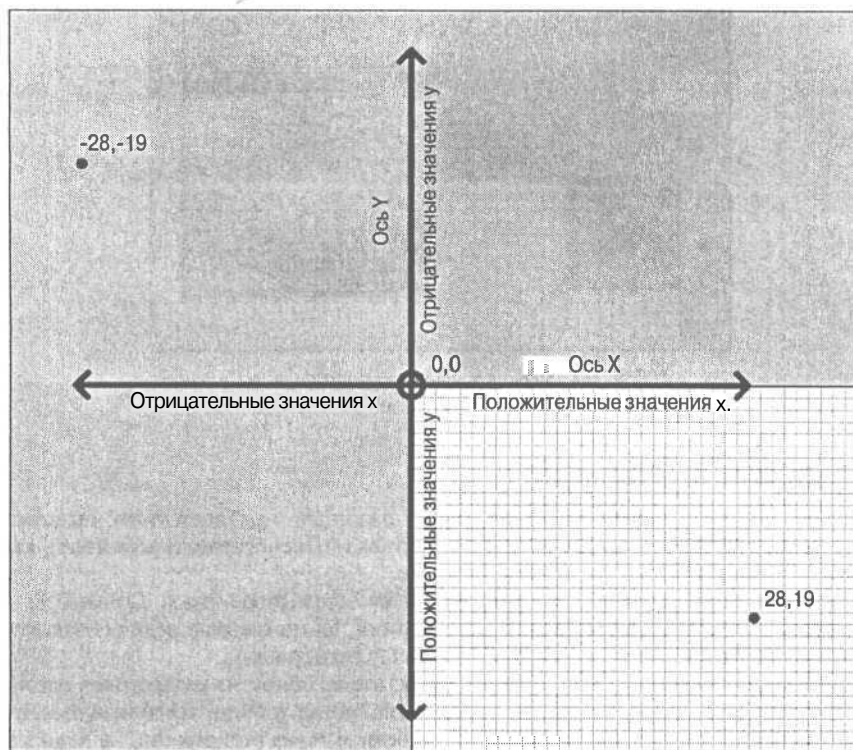


Рис. 6.4. В системе координат, используемой во Flash, координата 0,0 соответствует левому верхнему углу рабочего поля

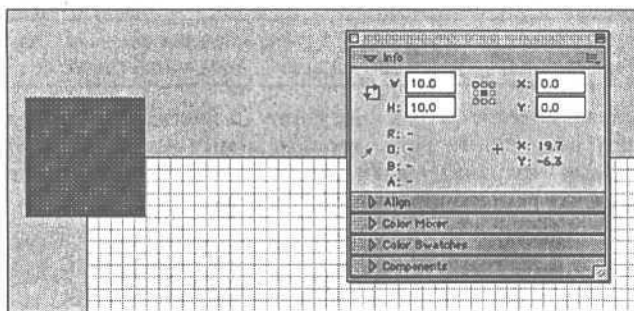


Рис. 6.5. Если точкой регистрации квадрата является его центр, объект будет расположен таким образом

Если точкой регистрации квадрата является его верхний левый угол, то она будет помещена в начало координат и весь квадрат расположится в пределах рабочего поля или экрана (рис. 6.6).

На заметку

При выполнении преобразования объекта в символ возможными параметрами являются девять стандартных точек регистрации. Однако впоследствии символ можно отредактировать, изменив его точку регистрации на любую другую в пределах символа.

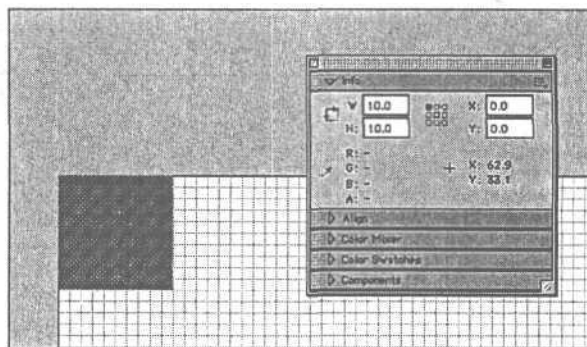


Рис. 6.6. Если точкой регистрации квадрата является его верхний левый угол, то объект будет размещен таким образом

ПРЕОБРАЗОВАНИЕ В СИМВОЛ

Чтобы преобразовать в символ объект, расположенный в рабочем поле, выделите его и выполните команду **Insert**⇒**Convert to Symbol** (Вставка⇒Преобразовать в символ), как показано на рис. 6.7, или нажмите клавишу <F8>.

После выполнения данной команды откроется диалоговое окно **Convert to Symbol** (рис. 6.8), очень похожее на окно **Create New Symbol**. Единственное отличие заключается в том, что в окне **Convert to Symbol** можно задать точку регистрации.

Чтобы указать точку регистрации объекта, щелкните на одном из квадратиков опции **Registration** (Регистрация). Вообще, в качестве точки регистрации удобнее всего использовать верхний левый угол объектов. Я практически всегда изменяю точку регистрации на левый верхний угол, поскольку эта контрольная точка является наилучшей для размещения объектов. При выполнении преобразования объекта в символ точкой регистрации по умолчанию является центральная точка объекта. Однако при ее изменении в диалоговом окне **Convert to Symbol** новая точка регистрации будет по умолчанию использоваться в течении всего сеанса работы.

Совет

Чтобы узнать, как изменить точку регистрации символа, обратитесь к разделу "Редактирование точек регистрации" далее в этой главе.

Задайте имя символа, укажите его тип и щелкните на кнопке **ОК**. Символ будет добавлен в библиотеку, а элемент, выделенный в рабочем поле, станет экземпляром этого символа и будет помечен квадратным ограничительным контуром, как показано на рис. 6.9. Крестик в кружочке обозначает точку регистрации данного символа.

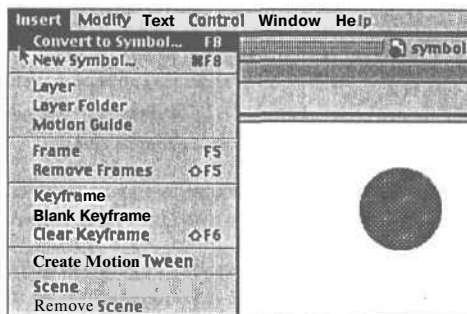


Рис. 6.7. Чтобы преобразовать в символ объект, расположенный в рабочем поле, выполните команду **Insert**⇒**Convert to Symbol**

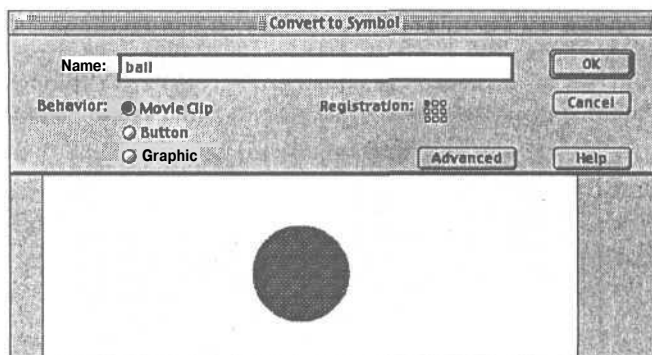


Рис. 6.8. Диалоговое окно *Convert to Symbol* позволяет преобразовать объект рабочего поля в символ и задать его точку регистрации



Рис. 6.9. При выполнении преобразования в символ фрагмент, выделенный в рабочем поле, станет экземпляром и будет обрамлен прямоугольным ограничительным контуром

Существующий символ можно продублировать и использовать в качестве шаблона нового символа. Выделите в рабочем поле экземпляр и выполните команду **Modify**⇒**Duplicate Symbol** (**Изменить**⇒**Дублировать** символ). В открывшемся диалоговом окне **Symbol Name** (Имя символа), которое показано на рис. 6.10, можно задать имя нового символа. Затем новый символ можно редактировать.

Пусть во Flash требуется создать игру в крестики-нолики. Сетку игры можно создать с помощью символов. Первым символом будет вертикальная линия. Для создания горизонтальных линий можно продублировать символ вертикальной линии, как показано на рис. 6.11.

После дублирования вертикальной линии ее можно развернуть и создать символ горизонтальной линии (рис. 6.12).

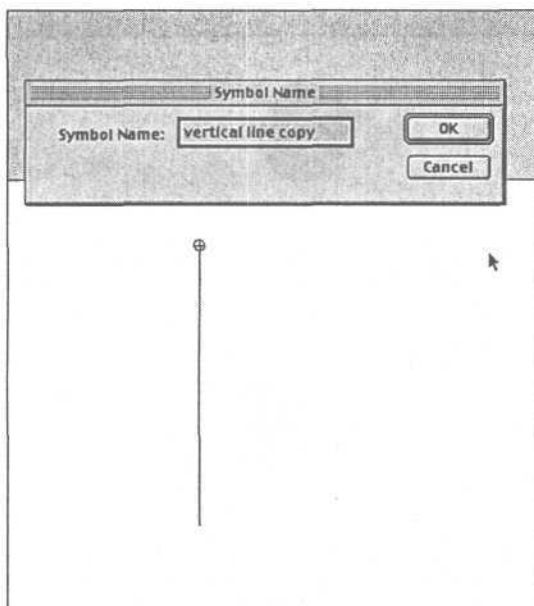
РЕДАКТИРОВАНИЕ СИМВОЛОВ

В процессе разработки проект обязательно претерпевает определенные изменения, и символы приходится редактировать. Измениться может цветовая схема, графика и даже навигационная структура фильма. Поскольку образцами элементов фильма являются символы, то при их редактировании будет изменен каждый экземпляр. В этом и заключается мощь символов. Если клиенту перестанет нравиться, например, цвет, то с помощью символов его можно быстро изменить и обновить по всему фильму. Символы можно редактировать тремя способами: на месте, в новом окне и в режиме редактирования символов.

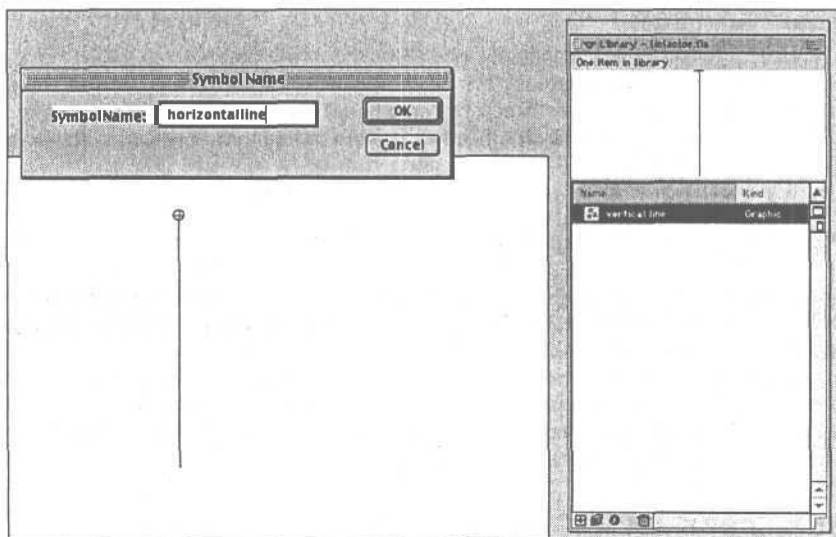
РЕДАКТИРОВАНИЕ НА МЕСТЕ

Из самого имени команды **Edit in Place** (Редактировать на месте) следует, что она позволяет внести изменения в символ в контексте содержимого рабочего поля. Эту команду удобно при-

менять для корректировки размера, цвета и формы. Для применения команды дважды щелкните на символе в рабочем поле либо выделите символ и выполните команду **Edit⇨Edit in Place**. Можно также щелкнуть правой кнопкой на символе (Windows) или щелкнуть, удерживая клавишу <Ctrl> (Macintosh), и из выпадающего меню выбрать пункт **Edit in Place** (рис. 6.13).



*Рис. 6.10. Чтобы продублировать символ, выделите существующий экземпляр и выполните команду **Modify⇨Duplicate Symbol***



*Рис. 6.11. В диалоговом окне **Symbol Name** дубликату символа можно присвоить имя*

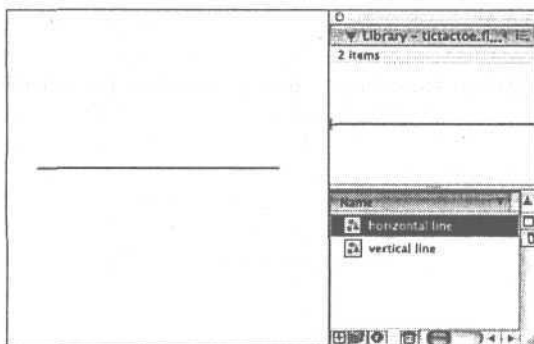


Рис. 6.12. Затем дубликат символа можно отредактировать и создать новый уникальный символ

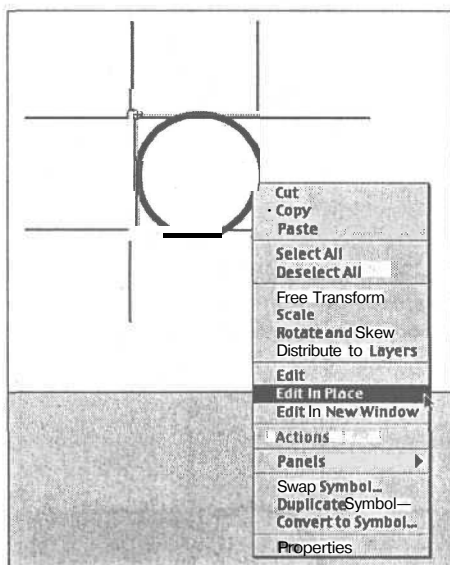


Рис. 6.13. Чтобы отредактировать символ на месте, выделите его и из выпадающего меню выберите пункт *Edit in Place*

Вернемся к примеру с игрой в крестики-нолики. Если символ круга слишком большой и не соответствует сетке игры, то с помощью команды *Edit in Place* его можно масштабировать согласно размерам сетки, добившись тем самым согласования размеров относительно других элементов (рис. 6.14).

По завершении редактирования щелкните на кнопке возврата или на имени текущей сцены в информационной строке в левом верхнем углу рабочего поля, либо выберите текущую сцену из выпадающего меню *Scene* (Сцена), либо выполните команду *Edit ⇨ Edit Document* (Правка ⇨ Правка документа).

РЕДАКТИРОВАНИЕ в новом ОКНЕ

Символы можно редактировать и в новом окне. При этом в новом окне открывается временная шкала символа, а основной фильм остается в исходном окне. Это позволяет при ре-

дактировании символа обращаться к разным элементам фильма, даже к тем, которые располагаются не в одном кадре с символом.

Чтобы отредактировать символ в новом окне, выделите его в рабочем поле, а затем щелкните на нем правой кнопкой мыши (Windows) или щелкните, удерживая клавишу <Ctrl> (Macintosh), и из выпадающего меню выберите пункт Edit in New Window (Редактировать в новом окне), как показано на рис. 6.15.

После того как символ откроется в новом окне (рис. 6.16), внесите в него необходимые изменения, а затем щелкните на кнопке закрытия, расположенной в левом (Macintosh) или в правом верхнем углу окна (Windows), и вернитесь к основной временной шкале.

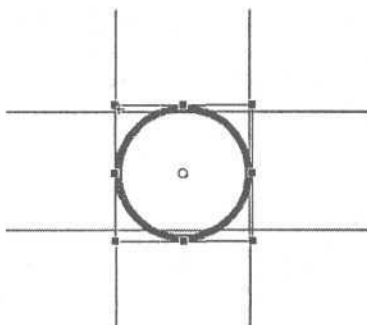


Рис. 6.14. Команда Edit in Place позволяет отредактировать символы в контексте других объектов рабочего поля

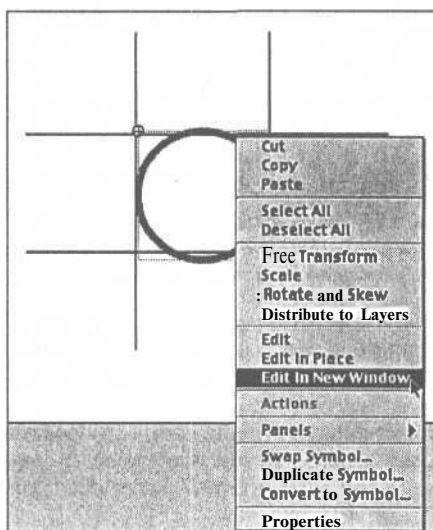


Рис. 6.15. Чтобы отредактировать символ в новом окне, из выпадающего меню выберите пункт Edit in New Window

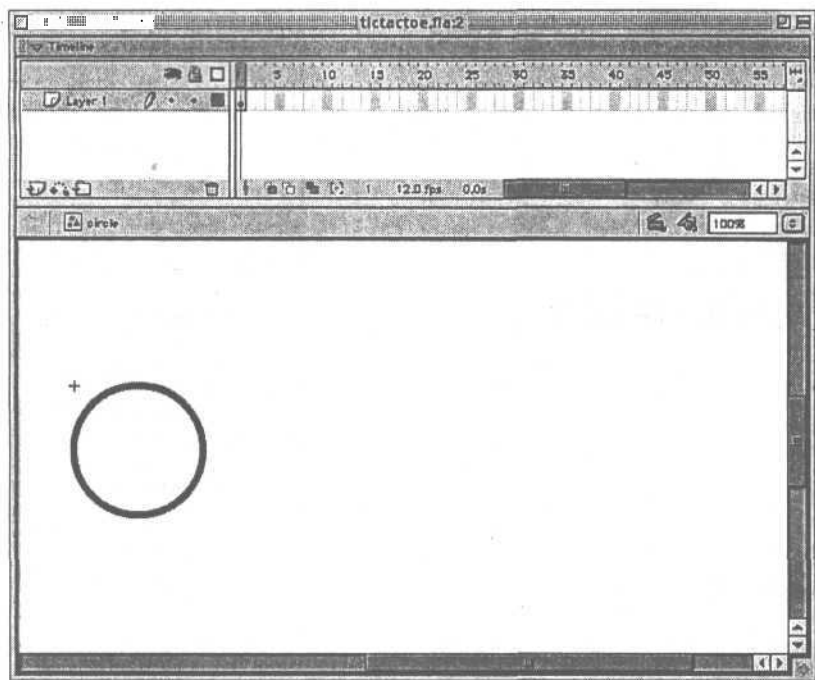


Рис. 6.16. После выполнения команды *Edit in New Window* символ откроется в отдельном окне

РЕЖИМ РЕДАКТИРОВАНИЯ СИМВОЛОВ

При редактировании в этом режиме вы перейдете из основной рабочей области во временную шкалу символа, в какой бы шкале не работали до этого. Если в символ требуется внести значительные изменения, его легче всего редактировать отдельно. Перейти в режим редактирования символов можно несколькими способами.

- Выделите символ в рабочем поле. Затем щелкните на нем правой кнопкой мыши (Windows) или щелкните, удерживая клавишу <Ctrl> (Macintosh), и из выпадающего меню выберите пункт **Edit**.
- Выделите символ в рабочем поле и выполните команду **Edit⇒Edit Symbols** (Правка⇒Редактировать символы).
- Дважды щелкните на пиктограмме символа (расположенной слева от его имени) на панели Library (Библиотека), как показано на рис. 6.17.
- Выделите символ на панели Library и из меню параметров этой панели выберите пункт **Edit** (рис. 6.18). Либо же щелкните на пиктограмме символа правой кнопкой мыши (Windows) или щелкните, удерживая клавишу <Ctrl> (Macintosh), и из выпадающего меню выберите пункт **Edit**.

Внесите необходимые изменения и вернитесь к основной временной шкале, щелкнув на кнопке возврата, расположенной в информационной строке, либо выполнив команду **Edit⇒Edit Document**.

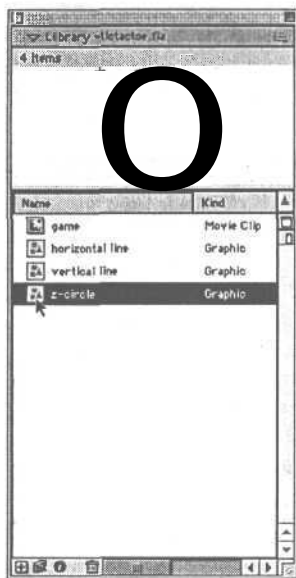


Рис. 6.17. Чтобы перейти в режим редактирования символов, дважды щелкните на пиктограмме символа на панели Library

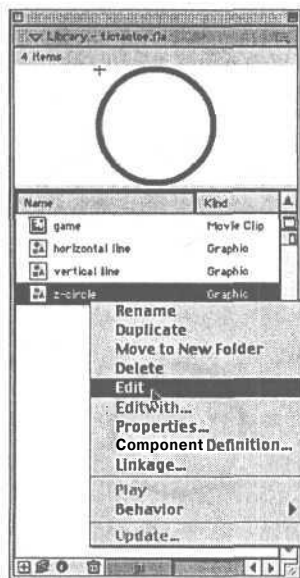


Рис. 6.18. В качестве альтернативного способа можно выбрать из выпадающего меню пункт Edit

РЕДАКТИРОВАНИЕ ТОЧЕК РЕГИСТРАЦИИ

Поскольку базовой точкой при выполнении преобразований и размещении символа является его точка регистрации, вполне естественно, что может возникнуть ситуация, когда ее потребуется отредактировать. Наиболее распространенным примером такой ситуации является сценарная строка состояния, которая часто используется для отображения хода выполнения загрузки. Пусть строка состояния представляет собой символ (обычно это прямоугольник), который расширяется согласно определенному сценарию, отображая ход выполнения загрузки. Если регистрационной точкой является центр символа строки состояния, то прямоугольник будет расширяться в обе стороны от центра (рис. 6.19), а это не совсем то, что нужно.

Чтобы панель загрузки расширялась только вправо (обычно именно таким образом отображается процесс выполнения задачи), регистрационной точкой символа строки состояния должна быть центральная точка левой стороны прямоугольника (рис. 6.20).

Быстрый способ изменения регистрационной точки символа, позволяющий визуально расположить точку в необходимом месте объекта, заключается в выделении символа в рабочем поле и использовании инструмента Free Transform (Свободное преобразование). На рис. 6.21 показано, что внутри выделенного символа появится кружок, обозначающий точку регистрации объекта. Чтобы изменить расположение регистрационной точки, щелкните и перетащите этот кружок.

PROGRESS: 50 % downloaded



Рис. 6.19. Если точкой регистрации является центр объекта, то любые преобразования будут выполняться в направлении от этой точки

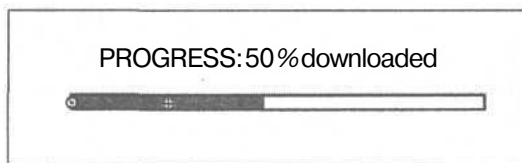


Рис. 6.20. Регистрационной точкой символа строки состояния является центр левой стороны прямоугольника, поэтому ширина будет изменяться слева направо

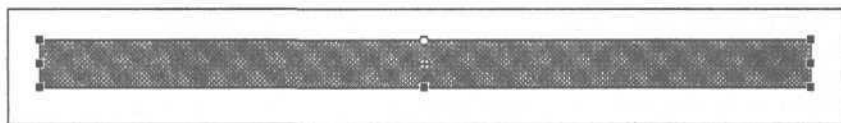


Рис. 6.21. Для изменения регистрационной точки символа выберите инструмент *Free Transform*, а затем щелкните и перетащите белый кружок

Совет

О точках регистрации рассказывалось выше, в разделе "Регистрация".

ЭКЗЕМПЛЯРЫ

Все символы размещаются на панели Library. Всякий раз при помещении символа в рабочее поле вы фактически помещаете туда экземпляр символа, а не сам символ. Это — важная деталь. Изменение исходного символа на панели Library означает изменение оригинала и каждого экземпляра этого символа. Однако экземпляры независимы от своих исходных символов. Экземпляр можно редактировать без изменения самого символа или других экземпляров этого же символа.

Чтобы отредактировать экземпляр, выделите его и внесите необходимые изменения. Если выделить экземпляр и преобразовать его с помощью инструмента *Free Transform*, то изменены будут размеры только этого экземпляра. Чтобы изменить размеры самого символа, необходимо воспользоваться одним из способов, описанных ранее в этой главе.

Кроме того, можно изменить тип поведения экземпляра. Пусть, например, имеется экземпляр графического символа, который нужно анимировать независимо от основной временной шкалы. Тип этого экземпляра легко изменить с графики на видеоклип. Для этого выделите его, а затем на панели инспектора свойств из раскрывающегося меню выберите нужный тип экземпляра (рис. 6.22).

ИЗМЕНЕНИЕ СВОЙСТВ ЭКЗЕМПЛЯРА

Экземпляр имеет некоторые свойства, которые не зависят от его исходного символа, — это яркость, оттенок, коэффициент прозрачности и расширенная настройка цвета. Задать эти свойства можно с помощью раскрывающегося меню *Color* (Цвет) на панели инспектора свойств (рис. 6.23).

Воспользуйтесь параметром *Brightness* (Яркость), если хотите сделать экземпляр светлее или темнее. Введите нужное значение прямо в поле *Brightness* или, перетаскивая ползунок, измените относительную яркость экземпляра в диапазоне от 100% (белый) до -100% (черный), как показано на рис. 6.24.



Рис. 6.22. Для изменения поведения экземпляра воспользуйтесь раскрывающимся меню *Instance* (Экземпляр) на панели инспектора свойств



Рис. 6.23. С помощью меню Color можно управлять свойствами цвета экземпляра

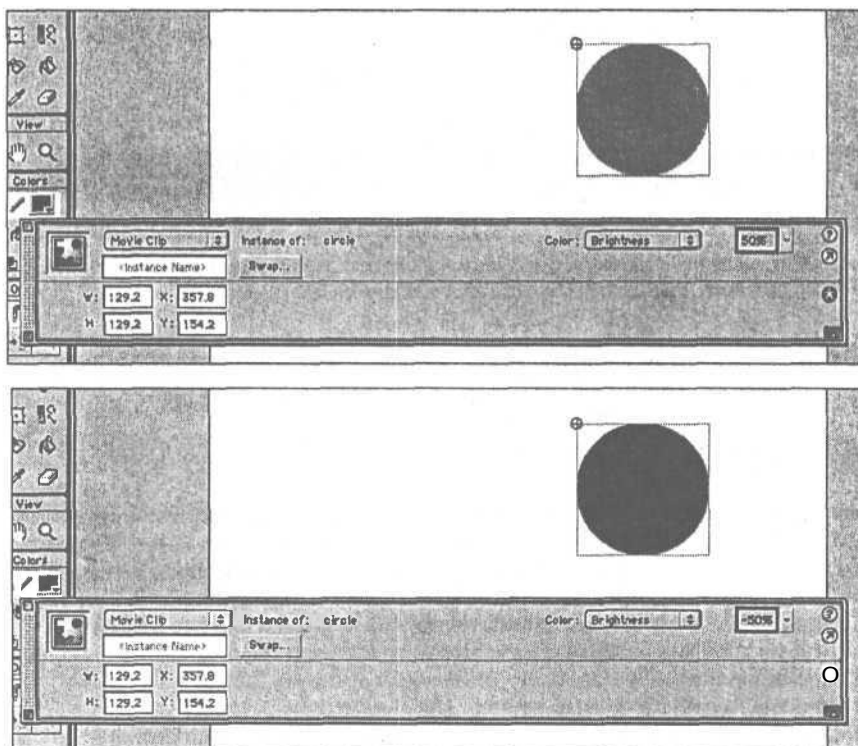


Рис. 6.24. Изменение яркости экземпляра делает его светлее или темнее

Параметр Tint (Оттенок) позволяет изменять оттенок цвета экземпляра путем корректировки значений красной, зеленой и синей составляющих, а также коэффициента прозрачности. Сначала выделите цвет, оттенок которого необходимо откорректировать, и либо введите его значения в поля RGB (Красный–зеленый–синий), либо выберите безопасный Web-цвет, щелкнув в поле цветовой палитры. После этого задайте значение коэффициента прозрачности, корректируя тем самым насыщенность оттенка, как показано на рис. 6.25.

Параметр Alpha позволяет изменять прозрачность экземпляра в диапазоне от 100% (непрозрачный) до 0% (полностью прозрачный). Введите нужное значение в поле Alpha или задайте его, перетаскивая ползунок.

Параметр Advanced (Дополнительные настройки) позволяет отдельно откорректировать значения красной, зеленой и синей составляющих, а также значение коэффициента прозрачности, что в результате создаст более тонкий оттенок цвета, чем просто с помощью параметра Tint. Из раскрывающегося меню Color выберите пункт Advanced (рис. 6.26). Щелкните на кнопке Settings (Настройка), в результате чего откроется диалоговое окно Advanced Effect (Расширенная настройка).



Рис. 6.25. Чтобы изменить оттенок цвета, выберите параметр **Tint** и измените значения красной, зеленой и синей составляющих, а также коэффициента прозрачности

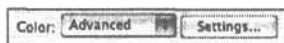


Рис. 6.26. Для доступа к диалоговому окну **Advanced Effect** из раскрывающегося меню **Color** выберите пункт **Advanced** и щелкните на появившейся кнопке **Settings**

В диалоговом окне **Advanced Effect** (рис. 6.27) можно изменить либо процентные значения (поля в левой части окна), либо абсолютные значения (поля в правой части окна) цветовых составляющих и коэффициента прозрачности.

Дополнительная настройка цвета осуществляется путем умножения значений цветовых составляющих на соответствующие процентные значения. Полученный результат затем прибавляется к абсолютным значениям, указанным в правой части окна, в результате чего создаются новые значения цветов. Не бойтесь **выполнять** расширенную настройку; просто потратьте некоторое время на то, чтобы поэкспериментировать с цветами. Нагляднее всего будут результаты, полученные при работе с растровыми цветами.

Совет

Подробная информация об использовании растровых изображений представлена в главе 4 "Использование растровых изображений во Flash".

ЗАМЕНА СИМВОЛОВ

При разработке фильма может возникнуть необходимость изменить один экземпляр символа на другой, например заменить графическое изображение. Будучи конструктивными элементами фильма, символы легко заменяют друг друга. Чтобы заменить экземпляр символа, выделите его и на панели инспектора свойств щелкните на кнопке **Swap** (Заменить), в результате чего откроется диалоговое окно **Swap Symbol** (Замена символа) (рис. 6.28). В этом окне показаны все символы, имеющиеся в библиотеке. Выберите нужный символ и щелкните на кнопке **OK**.

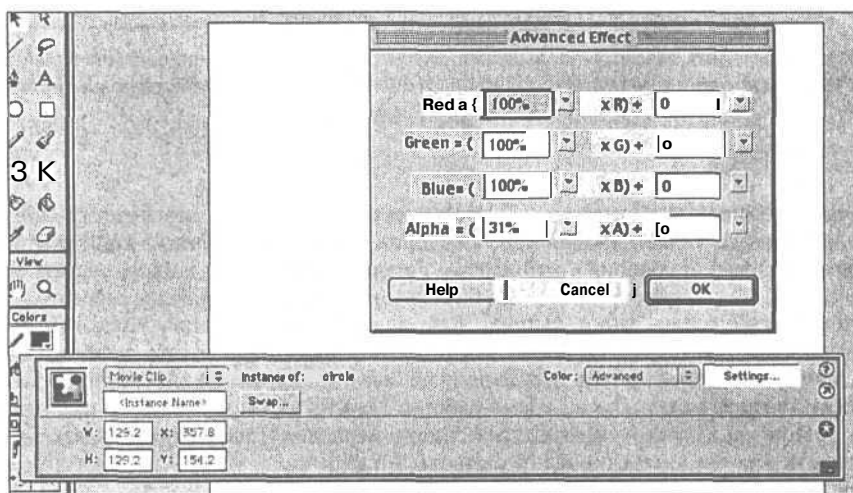


Рис. 6.27. В диалоговом окне **Advanced Effect** можно задавать либо процентные, либо числовые значения

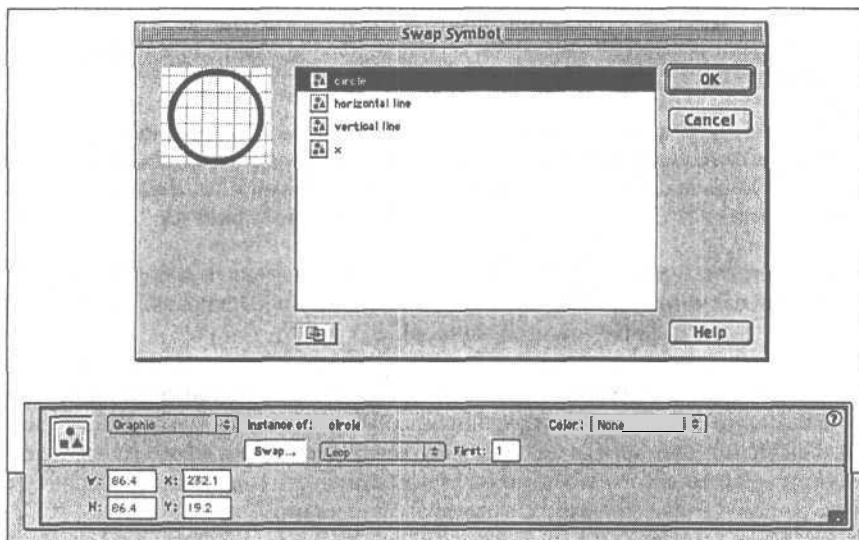


Рис. 6.28. Диалоговое окно *Swap Symbol* позволяет заменить символ, представленный определенным экземпляром

ИМЕНА СИМВОЛОВ

Экземплярам видеоклипов и кнопок можно присвоить уникальные имена, которые позволяют получить к ним доступ с помощью **ActionScript**. После присвоения имен экземплярам символов ими можно управлять программными средствами. Сценарии дают большую степень контроля над символом и позволяют создать содержимое, которое может изменяться в ответ на те или иные действия пользователя.

Совет

Подробно об использовании **ActionScript** для доступа к экземплярам рассказывается в главе 17 "Демонстрация мощи видеоклипов".

Чтобы назначить имя экземпляра, выделите экземпляр кнопки или видеоклипа в рабочем поле и введите уникальное имя в поле **Instance Name** (Имя экземпляра), как показано на рис. 6.29.

БИБЛИОТЕКА

Символы размещаются и систематизируются на панели **Library** (Библиотека). Представьте себе, что библиотека является коробкой, в которой сложен игрушечный конструктор. Помните, ваша мама всегда говорила, что каждая деталь конструктора должна лежать на своем месте в коробке? Чем лучше организована библиотека, тем легче вам будет работать.

Чтобы открыть панель **Library**, выполните команду **Window⇒Library** (Окно⇒Библиотека) или нажмите клавишу **<F11>**. Чтобы вы имели возможность перетаскивать экземпляры символов в рабочее поле, библиотека всегда должна быть открытой. Ее можно привязать к другим панелям, которые открыты во время работы. Панель **Library** состоит из окна предварительного просмотра и прокручивающегося списка символов и импортированных элементов. Чтобы увидеть, как систематизированы символы в библиотеке, щелкните на кнопке **Wide Library View** (Расширить вид библиотеки) (рис. 6.30).

В первом столбце перечислены имена символов (вместе с пиктограммами их типов), по умолчанию отсортированные в алфавитном порядке. Они будут видны даже тогда, когда па-

нель Library не развернута полностью. Чтобы изменить порядок сортировки символов в библиотеке, щелкните на заголовке одного из столбцов. Символы могут быть отсортированы по именам, типам, количеству применений, связям или датам изменения.

В столбце Use Count (Количество применений) указывается число, обозначающее количество применений символа в фильме. Сортировка по этому столбцу поможет выяснить, какие символы больше не используются, и соответствующим образом модифицировать файл фильма. Если размер файла слишком велик, выясните, какие символы имеют большой объем. Возможно, некоторые из них можно не использовать. Щелкните на заголовке столбца Use Count, чтобы отсортировать элементы по количеству их применений в фильме. По умолчанию столбец Use Count не обновляется, но чтобы увидеть текущее число применений, из меню параметров панели Library выберите пункт Keep Use Counts Updates (Обновлять число применений), как показано на рис. 6.31.

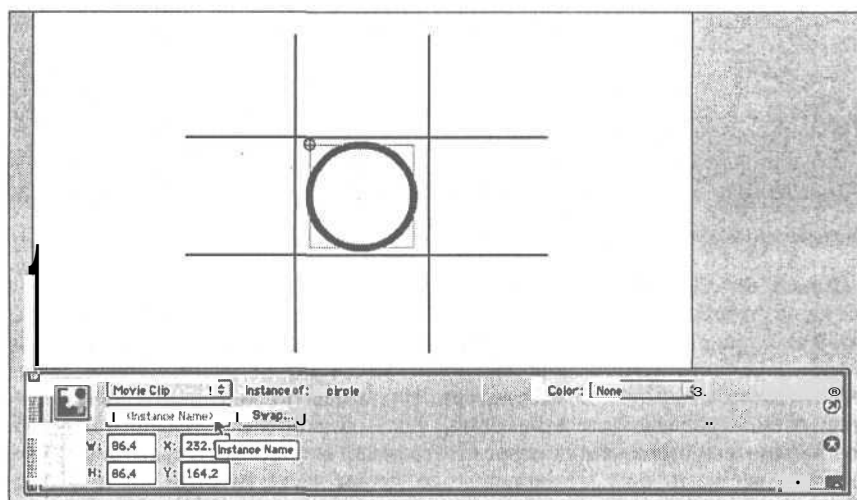


Рис. 6.29. Имена экземпляров позволяют получить доступ к отдельным экземплярам видеоклипов и кнопок с помощью *ActionScript*

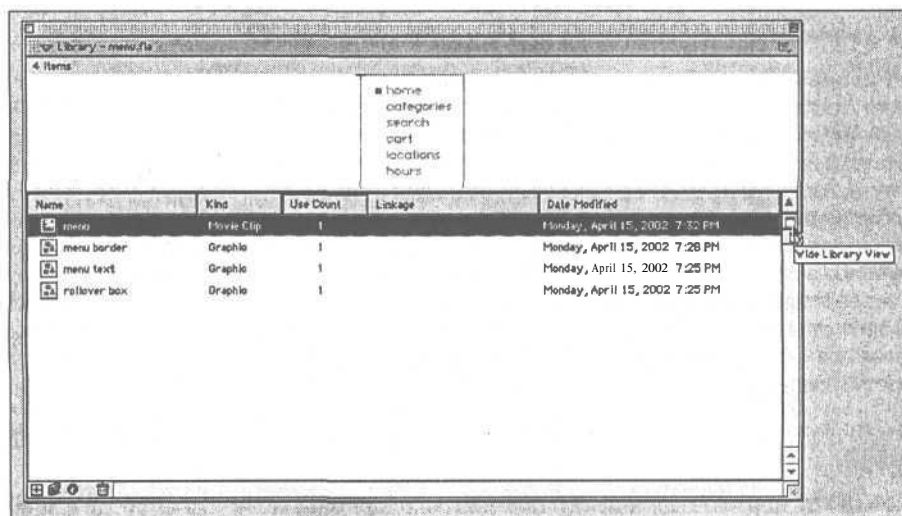


Рис. 6.30. Чтобы расширить панель Library, щелкните на кнопке *Wide Library View*

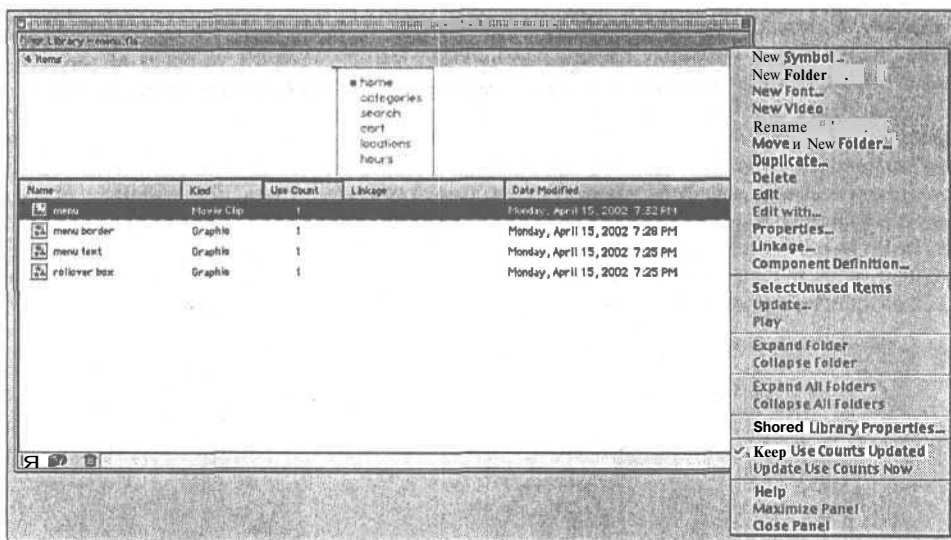


Рис. 6.31. Меню параметров панели Library

Содержимое библиотеки можно распределить по папкам. Для создания новой папки щелкните на пиктограмме New Folder (Создать папку), расположенной в нижней части панели Library. Как и при работе с папками слоев, использование папок библиотеки позволит оптимизировать и без того недостаточно свободное место экрана. Папки можно использовать для систематизации содержимого библиотеки. Например, можно создать папки для импортированных элементов, символов навигации и т.д. При работе над большим проектом число элементов библиотеки будет увеличиваться с огромной скоростью, и если не систематизировать символы и элементы с помощью папок, то, чтобы найти нужный символ, придется затратить достаточно много времени на прокручивание списка содержимого библиотеки.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Как можно скопировать символ из одного фильма в другой?

Во Flash MX можно легко скопировать символы из одного фильма в другой. Если одновременно открыто несколько файлов FLA, то просто откройте панель Library, и на ней появятся все библиотеки открытых документов. В строке заголовка панели Library будут указаны имена всех файлов FLA, как показано на рис. 6.32. Каждую библиотеку можно свернуть или развернуть, щелкнув на треугольнике слева от имени файла.

Чтобы скопировать символ из одной библиотеки в другую, просто щелкните на нем и перетащите в нужную библиотеку либо перетащите его в рабочее поле другого файла FLA. При этом на рабочее поле будет помещен экземпляр символа, а сам символ будет добавлен в библиотеку.

Если вы хотите воспользоваться несколькими символами из другого фильма, откройте соответствующий файл FLA как библиотеку (при этом откроется только панель Library выбранного файла без временной шкалы и рабочего поля). Для этого выполните команду File ⇒ Open as Library (Файл ⇒ Открыть как библиотеку) и выберите нужный файл FLA.

Почему выводится сообщение Resolve Library Conflict (Разрешить конфликт библиотек)?

Во Flash MX можно легко заменить существующий символ на новый с тем же самым именем. Но если вы попытаетесь перетащить символ с тем же именем из другого файла FLA, то на

экран будет выведено сообщение о конфликте библиотеки. Таким образом Flash предостерегает о непреднамеренном переписывании символов. В диалоговом окне *Resolve Library Conflict* (Разрешить конфликт библиотеки), которое показано на рис. 6.33, можно выбрать опцию замены существующего символа.

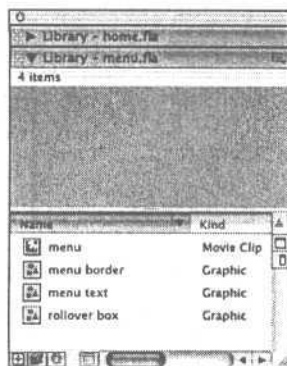


Рис. 6.32. На панели *Library* будут перечислены библиотеки всех открытых документов

FLASH ЗА РАБОТОЙ: ВЛОЖЕНИЕ СИМВОЛОВ

Мы все время рассматривали символы как конструктивные элементы Flash, а теперь возникает вопрос, как сложить их вместе? Одни символы можно вкладывать в другие. Пусть, например, требуется создать выпадающее навигационное меню на Web-узле розничного продавца. Можно начать с создания графических символов, которые будут являться границами этого выпадающего меню. Затем символ границы можно преобразовать в видеоклип, в одном слое которого будет находиться граница, в другом слое — текстовые символы для пунктов меню, а в третьем — графические символы ролловеров (рис. 6.34).

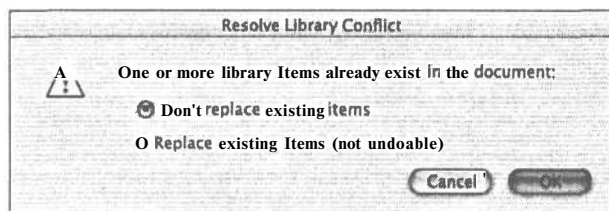


Рис. 6.33. В диалоговом окне *Resolve Library Conflict* можно выбрать опцию замены существующего символа

Сведение символов воедино таким образом позволяет быстро отредактировать их. Если необходимо изменить или реорганизовать текстовые пункты меню, потребуется внести изменения только в символ текста меню, даже если видеоклип меню используется в нескольких местах фильма. Изменения будут отображены везде, где применялся символ.

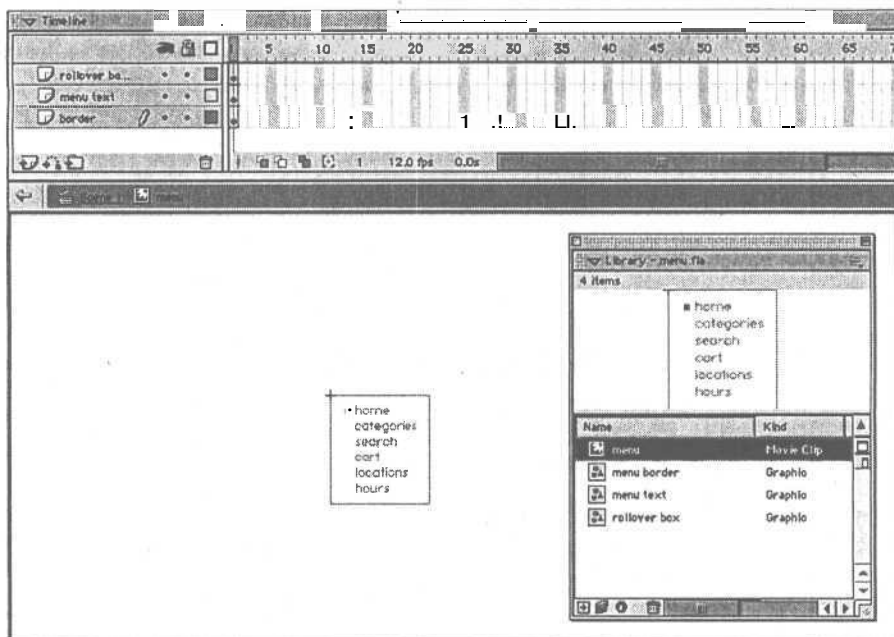


Рис. 6.34. Символы можно вкладывать друг в друга, создавая сложную графику, которую впоследствии можно будет модифицировать

АНИМАЦИЯ во FLASH

В ЭТОЙ ГЛАВЕ...

Знакомство с анимацией	137
Кадры и ключевые кадры	138
Создание промежуточных отображений	141
Перемещение по временной шкале	151
Мультипликация во Flash	152
Основные правила при создании анимации	153
Возможные проблемы	154
Flash за работой: упрощение анимации	155

ЗНАКОМСТВО с АНИМАЦИЕЙ

Анимация — это иллюзия движения, создаваемая с помощью последовательности неподвижных изображений. Ключевым здесь является слово *последовательность*. При создании иллюзии движения определяющим фактором является качество последовательности, которое оказывается более важным, чем качество изображений. Это утверждение справедливо, так как анимация не является искусством перемещения объектов, а представляет собой искусство рисования и записи движения. Искусство заключается в действии.

Программа Flash обладает уникальными средствами создания анимации. Векторный формат превосходно подходит для создания рисованной анимации или мультипликации. Векторные графические изображения имеют формы, которые можно легко и точно преобразовать, сохраняя при этом небольшие размеры файлов. Кроме того, при работе с векторной графикой объекты можно увеличивать либо уменьшать, что помогает создать убедительную анимацию. Во Flash можно анимировать и растровые изображения. Система символов Flash

также идеально подходит для анимации. Символы позволяют разбить анимацию на отдельные составляющие, которые можно использовать повторно, что также способствует повышению эффективности работ и оптимизации размера файла.

Совет

Подробная информация о векторной графике была представлена в главе 3 "Рисование во Flash", о растровых изображениях — в главе 4 "Использование растровых изображений во Flash", а о символах — в главе 6 "Символы, экземпляры и элементы библиотек".

КАДРЫ И КЛЮЧЕВЫЕ КАДРЫ

Для систематизации и управления фильмом и анимацией во Flash используется временная шкала. Панель Timeline (Временная шкала) в горизонтальном направлении поделена на кадры, представляющие собой дискретные единицы времени, а в вертикальном направлении — на слои, представляющие собой дискретные единицы пространства, уложенные в стек. Анимация протекает во времени и в пространстве. Временной интервал анимации определяется общим количеством кадров, а занимаемое ими пространство, в свою очередь, определяется размерами рабочего поля (хотя в его пределах можно разместить практически неограниченное число слоев).

Магическое число 16000

Интересовались ли вы когда-нибудь, насколько большим может быть Flash-фильм? Знайте, что при работе во Flash часто приходится иметь дело с магическим числом 16000. Фильмы могут иметь до 16000 кадров. При превышении этого значения воспроизведение фильма останавливается. Максимальное число экземпляров символов, которое может содержаться в фильме, тоже составляет 16000. Та же цифра фигурирует в максимально поддерживаемом количестве слоев, которое может быть экспортировано, и максимальном количестве загруженных фильмов.

Кадр представляет собой статический временной интервал — содержимое рабочего поля в определенный момент времени. В отличие от обычных кадров, ключевые кадры являются критическим местом стыковки при изменении содержимого рабочего поля. Это существенное отличие. Анимация описывает действие, а сущностью действия является изменение. Анимация во Flash создается путем отслеживания изменений: где находится место изменения содержимого рабочего поля, т.е. ключевые кадры, и как долго содержимое остается без изменений, т.е. кадры. Кадры указывают на содержимое, а ключевые кадры сигнализируют о действии, т.е. об изменении.

Воспроизводящая головка отображает отдельный кадр за один раз, хотя в одном кадре может быть отображено неограниченное количество слоев. Очень важно, чтобы содержимое было разнесено в отдельные слои, поскольку это позволяет анимировать объекты индивидуально. Фильм может иметь статический фон, который остается неподвижным, и несколько слоев с содержимым, которое анимируется в различных точках фильма. Хотя содержимое может быть размещено в нескольких слоях, маловероятно, чтобы вы захотели отображать все содержимое в каждом кадре фильма. Кадры существуют в пределах слоя только тогда, когда содержимое слоя видно в рабочем поле.

ДОБАВЛЕНИЕ, РЕДАКТИРОВАНИЕ, УДАЛЕНИЕ И КОПИРОВАНИЕ КАДРОВ

При создании нового документа каждый слой помещается в отдельный кадр, как показано на рис. 7.1. Добавление кадров определяет, когда содержимое слоя является видимым. Общая длина фильма определяется максимальным кадром какого-либо слоя. Добавление кадров к слою удлиняет содержимое в рабочем поле. Удаление кадров укорачивает время, в течение которого содержимое будет видимым. При добавлении к фильму содержимого и слоев, веро-

ятнее всего, возникнет необходимость управлять промежутком времени, в течение которого различные объекты будут видны на рабочем поле.

Для добавления кадров к слою выделите кадр на панели Timeline и нажмите клавишу <F5>. В результате будет выполнена вставка кадров, как показано на рис. 7.2.

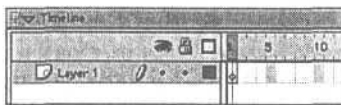


Рис. 7.1. Новый документ содержит один слой и один кадр

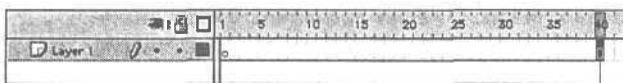


Рис. 7.2. Для добавления кадров выделите кадр на временной шкале и нажмите клавишу <F5>

Когда в рабочее поле добавляется содержимое, Flash преобразовывает текущий кадр в **ключевой**. В кадрах отображается содержимое предшествующего им **ключевого** кадра. Если не будет добавлено промежуточных отображений, содержимое между двумя ключевыми кадрами останется статичным.

Заполнение кадров

Заполнение кадров, т.е. добавление промежуточных отображений, состоит в заполнении промежуточных кадров, расположенных между ключевыми, в которых происходит действие. Заполнение кадров Flash выполняет автоматически. Вы можете задать движение или промежуточное отображение, а Flash интерполирует изменения между ключевыми кадрами.

Чтобы добавить ключевой кадр, щелкните на кадре на панели Timeline (Временная шкала) и нажмите клавишу <F6>. Вставленный ключевой кадр будет отображаться в виде пустого кружка на панели Timeline (рис. 7.3).

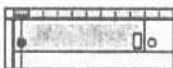


Рис. 7.3. Пустые ключевые кадры имеют вид пустых кружков, а ключевые кадры с содержимым — сплошных

В процессе работы можно легко вносить изменения в кадры и ключевые кадры. Для удаления кадров выделите их, щелкните правой кнопкой мыши (Windows) или щелкните, удерживая клавишу <Ctrl> (Macintosh), и из контекстного меню выберите пункт Remove Frame (Удалить кадр). Чтобы преобразовать ключевой кадр в обычный, т.е. удалить изменения, происходившие в рабочем поле этого ключевого кадра без удаления кадра, выделите ключевой кадр и из контекстного меню выберите пункт Clear Keyframe (Очистить ключевой кадр).

Чтобы выделить несколько кадров, щелкните и **перетащите** по ним курсор. Кадры можно копировать и вставлять. Для этого выделите кадры и/или ключевые кадры, щелкните правой кнопкой мыши (Windows) или щелкните, удерживая клавишу <Ctrl> (Macintosh), и из контекстного меню выберите пункт Copy Frames (Копировать кадры). Затем на временной шкале выделите другой кадр, щелкните правой кнопкой мыши (Windows) или **щелкните**, удерживая клавишу <Ctrl> (Macintosh), и из контекстного меню выберите пункт Paste Frames (Вставить кадры). Можно копировать и вставлять кадры сквозь слои, например, кадры с 4-го по 12-й в слоях 4 и 5 можно скопировать и вставить в другие слои. Если кадры скопированы из нескольких слоев, то и вставляются они тоже в несколько слоев.

УПРАВЛЕНИЕ КЛЮЧЕВЫМИ КАДРАМИ

Ключевые кадры можно добавить в любом месте в пределах слоя. По необходимости Flash вставляет кадры для заполнения расстояния между существующим и новым ключевым кадром. В ключевых кадрах будет отображено содержимое предыдущих ключевых кадров, которое можно изменить. Чтобы убрать содержимое из рабочего поля, можно добавить пустые ключевые кадры. Для вставки пустого ключевого кадра выделите кадр и нажмите клавишу <F7>.

Существующие кадры можно преобразовать в ключевые или пустые ключевые кадры. Выделите кадр и щелкните на нем правой кнопкой мыши (Windows) или щелкните, удерживая клавишу <Ctrl> (Macintosh), в результате чего откроется контекстное меню (рис. 7.4). Выберите из него пункт Convert to Keyframes (Преобразовать в ключевые кадры) или Convert to Blank Keyframes (Преобразовать в пустые ключевые кадры).

Чтобы переместить ключевой кадр во временной шкале, сначала щелкните на нем. После того как вы отпустите кнопку мыши, ключевой кадр будет выделен. Теперь перетащите его на новое место или даже в новый слой.

ЧАСТОТА СМЕНЫ КАДРОВ

Скорость анимации определяется как частотой смены кадров фильма, так и числом кадров, в которых воспроизводится анимация во временной шкале. Переменная частота смены кадров является сугубо цифровой концепцией воспроизведения анимации. Анимация в традиционной среде происходит при фиксированной частоте смены кадров. Flash позволяет задать для фильма частоту смены кадров, выражаемую в числе кадров в секунду. Частоту смены кадров лучше задать сразу же при открытии документа. Если вы впоследствии решите изменить частоту, придется откорректировать длительность анимации. Для задания частоты смены кадров фильма выполните следующие действия.

1. Щелкните на незанятом участке рабочего поля. На панели инспектора свойств появятся параметры документа, как показано на рис. 7.5.

1. Щелкните в поле Frame Rate (Частота смены кадров) и введите нужное число. По умолчанию в этом поле введено число 12 (кадров в секунду), но для отображения в Web приемлемо любое значение от 8 до 18.

Чем выше частота смены кадров, тем больше будет передаваться информации, тем более плавной будет анимация и тем больше будет размер файла. Если вы знаете, что пользовательская аудитория имеет соединение с высокой пропускной способностью и работает на быстрых компьютерах, можете смело задавать большую частоту смены кадров. Однако в большинстве случаев лучше перестраховаться и использовать не очень высокую частоту смены кадров. Если загрузка анимации будет длиться слишком долго, пользователь может просто не дожидаться ее завершения и никогда не увидеть ваше творение.

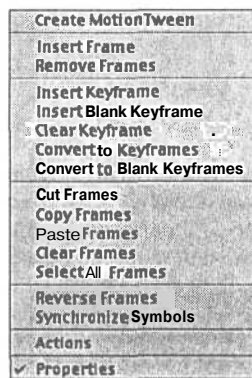


Рис. 7.4. Для доступа к данному контекстному меню щелкните на кадре правой кнопкой мыши (Windows) или щелкните, удерживая клавишу <Ctrl> (Macintosh)



Рис. 7.5. Частота смены кадров фильма задается в параметрах документа, отображаемых на панели инспектора свойств

После задания частоты смены кадров основные усилия можно сосредоточить на распределении анимации во времени. Анимация, разворачивающаяся в большем числе кадров, протекает медленнее. Чем меньше количество кадров, тем быстрее выполняется анимация. Чтобы добиться идеальной скорости воспроизведения анимации, поэкспериментируйте с ее длительностью.

СОЗДАНИЕ ПРОМЕЖУТОЧНЫХ ОТОБРАЖЕНИЙ

Во Flash существуют два типа анимации: покадровая и с созданием промежуточных отображений. Покадровая анимация изменяет содержимое рабочего поля в каждом кадре и состоит из ряда ключевых кадров. Покадровая анимация — это очень трудоемкий процесс, в результате которого получают файлы больших размеров. Она является более органичной, но требует индивидуального управления изображениями.

Анимация с созданием промежуточных отображений характеризуется нестационарными ключевыми кадрами. Ключевые изменения происходят в рабочем поле, а промежуточные кадры задаются программой *Rash*. Программа берет два ключевых кадра, расположенных в одном слое, и создает промежуточные отображения, постепенно изменяя содержимое рабочего поля из одного состояния в другое. Такая процедура интерполяции отличий между ключевыми кадрами называется созданием промежуточных отображений, или заполнением кадров.

Чтобы создать простую анимацию подпрыгивающего мяча, выполните следующие действия.

1. Откройте новый документ, выполнив команду **File⇒New (Файл⇒Создать)** или воспользовавшись комбинацией клавиш **<Cmd+N>** (Macintosh) или **<Ctrl+N>** (Windows).
2. Щелкните на инструменте **Oval (Овал)** и выберите заливку красного цвета без штриха.

Штрихи и анимация

При анимации сплошных изображений используйте штрихи только по мере необходимости, когда цвет штриха отличается от цвета заливки. Если цвет штриха не отличается от цвета заливки, то Flash придется выполнять ненужные вычисления, что замедлит скорость воспроизведения изображения и увеличит размер файла.

3. Вверху рабочего поля нарисуйте красный круг, перетаскивая инструмент **Oval** и удерживая при этом нажатой клавишу **<Shift>** (рис. 7.6).
4. Выделите круг и преобразуйте его в символ, нажав клавишу **<F8>**. В диалоговом окне **Convert to Symbol (Преобразовать в символ)** присвойте символу имя **ball**, переключатель **Behavior (Поведение)** установите в поле опции **Graphic (Графика)** и щелкните в центральном квадрате сетки регистрации. После этого закройте диалоговое окно **Convert to Symbol**, щелкнув на кнопке **OK**.
5. Теперь нужно заставить мяч двигаться. Для этого нужно изменить его положение в рабочем поле. На панели **Timeline (Временная шкала)** перейдите к кадру 12 и для вставки ключевого кадра нажмите клавишу **<F6>**. Обратите внимание, что в рабочем поле пока ничего не изменилось.
6. Выберите инструмент **Arrow (Стрелка)**, выделите им мяч и переместите его в нижнюю часть рабочего поля, как показано на рис. 7.7.
7. Воспроизведите фильм в рабочей области, нажав клавишу **<Return>** (Macintosh) или **<Enter>** (Windows). Обратите внимание на то, что во время воспроизведения 11 кадров мяч будет оставаться вверху рабочего поля и только на 12-м кадре перепрыгнет вниз. Вспомните, что в кадрах отображается содержимое предшествующего ключевого кадра.
8. Чтобы добиться эффекта постепенного перемещения мяча в нижнюю часть рабочего поля, выберите любой кадр между первым и двенадцатым. Откройте контекстное меню, щелкнув на кадре правой кнопкой мыши (Windows) или щелкнув на нем, удерживая клавишу **<Ctrl>** (Macintosh). Из контекстного меню выберите пункт **Create Motion Tween (Создать эффект промежуточного движения)**.

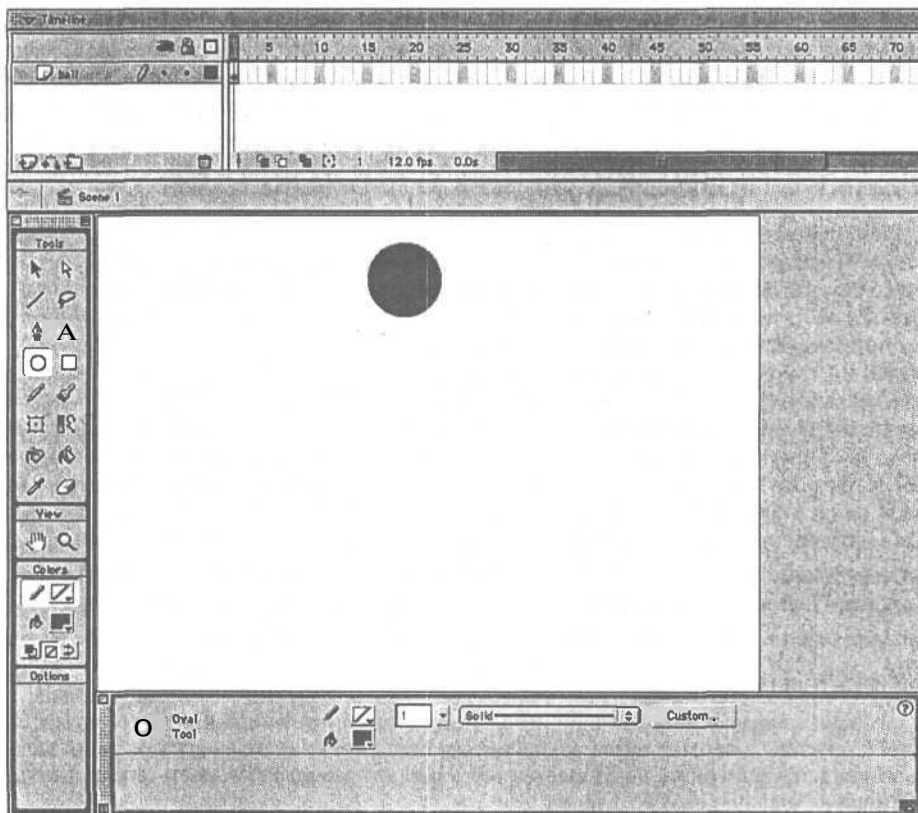


Рис. 7.6. При первом добавлении содержимого в рабочее поле нового документа Flash преобразует единственный существующий кадр в ключевой

9. Снова воспроизведите анимацию, нажав клавишу **<Return>** (Macintosh) или **<Enter>** (Windows). Теперь мяч будет постепенно падать во всех 12 кадрах.
10. Сохраните файл анимации под именем **ball.fla**. Мы еще вернемся к нему в этой главе.

Совет

Подробная информация о процедуре рисования простых фигур приведена в главе 3 "Рисование во Flash", а о символах — в главе 6 "Символы, экземпляры и элементы библиотек".

Можно создать и покадровую анимацию мяча, но лучше просто задать путь анимации и позволить Flash выполнить всю работу автоматически. Тем самым вы сэкономите собственные силы и не увеличите размер файла. Анимация движения мяча позволяет оценить преимущество использования методики создания промежуточных отображений.

ПРОМЕЖУТОЧНОЕ ОТОБРАЖЕНИЕ ДВИЖЕНИЯ

Существуют два типа промежуточных отображений: формы и движения. При промежуточном отображении движения интерполируются изменения положения, масштаба, вращения, цвета и прозрачности изображения, расположенного в различных местах в нескольких ключевых кадрах. При промежуточном отображении формы интерполируются отличия меж-

ду двумя изображениями, позволяя плавно преобразовывать одно в другое. Промежуточное отображение создается только в пределах одного слоя. В анимации мяча используется промежуточное отображение движения. Для его создания требуется, чтобы объекты в ключевых кадрах представляли собой символы или группы и располагались в пределах одного слоя. При попытке создания промежуточного отображения без преобразования изображений в символы между ключевыми кадрами появится пунктирная линия, а на панели инспектора свойств — значок предупреждения (рис. 7.8).

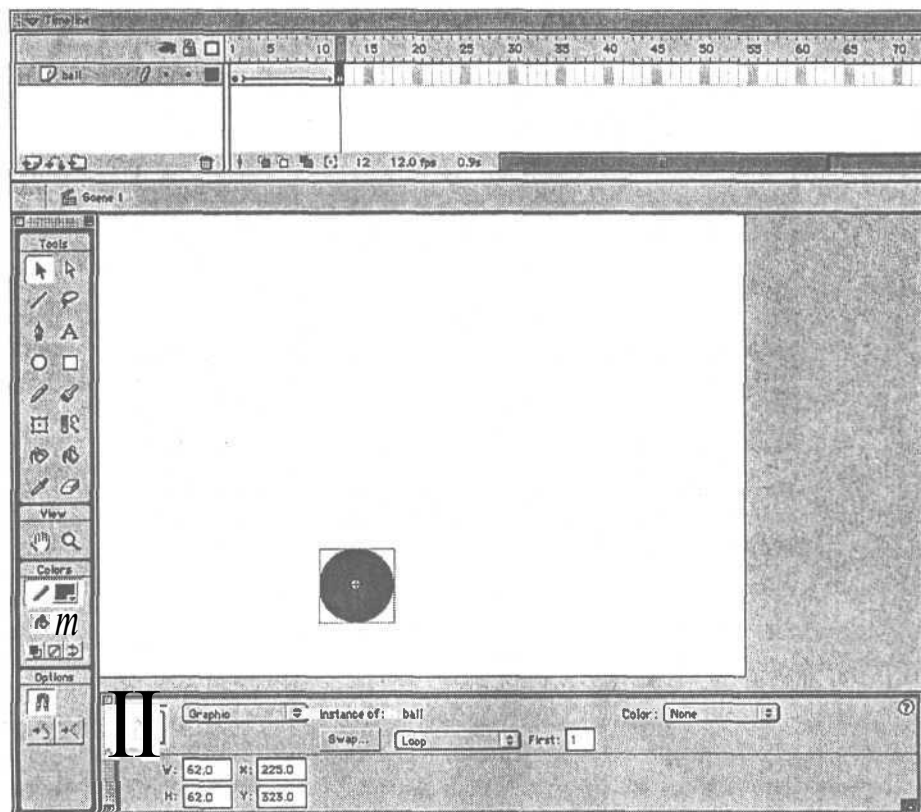


Рис. 7.7. В кадр 12 вставьте ключевой кадр и с помощью инструмента *Arrow* переместите мяч в нижнюю часть рабочего поля

Эти признаки характеризуют незавершенное создание промежуточных отображений, и при предварительном просмотре анимации вы увидите, что предполагаемые промежуточные отображения не выполняются.

НАПРАВЛЯЮЩИЕ ДВИЖЕНИЯ

При создании промежуточного отображения движения Flash прокладывает кратчайший путь между двумя положениями в ключевых кадрах. Хотя это и позволяет сэкономить на размере файла, но не всегда желательно. В примере с анимацией мяча мы не получим естественную траекторию падения. Для получения правильной траектории необходимо создать направляющую движения. Для этого выполните следующие действия.

1. Откройте файл **ball.fla**.
2. Для создания иллюзии отскока мяча необходимо сместить точку, в которой мяч касается земли. Щелкните на панели Timeline и выберите кадр 12.
3. Выберите инструмент **Arrow** и перетащите экземпляр мяча вправо по рабочему полю.
4. Воспроизведите анимацию, нажав клавишу **<Return>** (Macintosh) или **<Enter>** (Windows). На рис. 7.9 видно, что мяч перемещается из верхней части рабочего поля в правую нижнюю под неестественным углом. Чтобы решить эту проблему, необходимо создать направляющую движения.

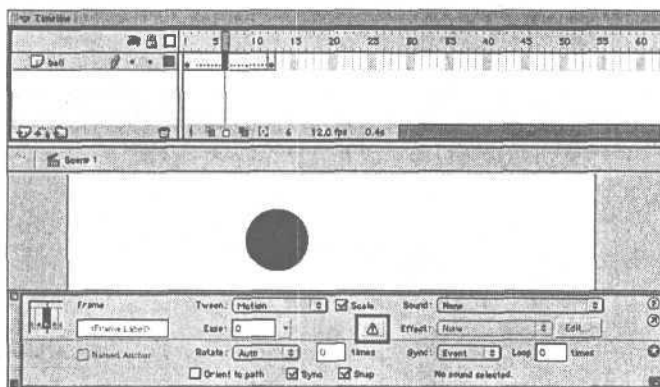


Рис. 7.А Пунктирная линия на временной шкале и значок предупреждения на панели инспектора свойств свидетельствуют о незавершенном создании промежуточного отображения

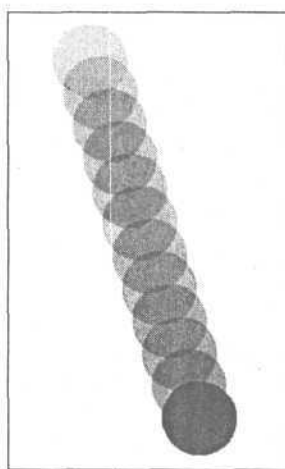


Рис. 7.9. При переходе в режим *Onion Skin* (Калькирование) видно, что при создании промежуточного отображения движения мяч движется по кратчайшему расстоянию между положениями в ключевых кадрах

5. Выделите слой, в котором располагается мяч, и выполните команду **Insert⇒Motion Guide** (Вставка⇒Направляющая движения). На рис. 7.10 показано, что над выделенным слоем добавляется слой направляющей движения.

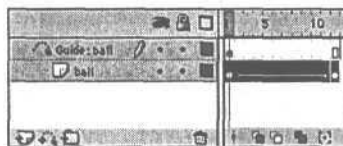


Рис. 7.10. Для создания направляющей движения выделите слой и выполните команду **Insert⇒Motion Guide**

6. Выберите инструмент Реп (Перо), щелкните им на рабочем поле и перетащите так, чтобы нарисовать две точки, образующие дугу реальной траектории падения мяча из верхней части рабочего поля в нижнюю (рис. 7.11).

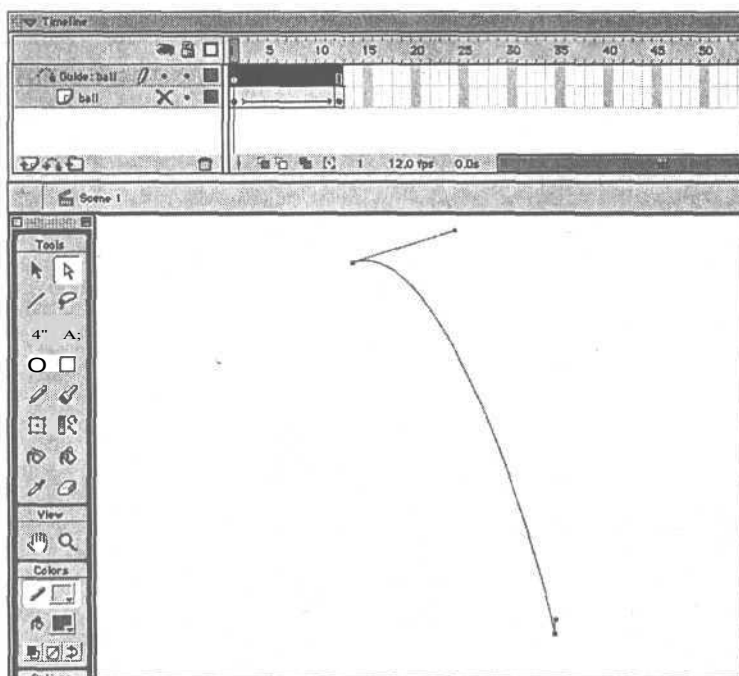


Рис. 7.11. Для создания эффекта отскока воспользуйтесь инструментом Реп и нарисуйте дугу, вдоль которой будет осуществляться движение мяча

7. В слое мяча выделите кадр 1 и проверьте, чтобы на панели инспектора свойств в поле опции Snap (Привязка) был установлен флажок (рис. 7.12). В этом случае точка регистрации промежуточного отображения объекта будет привязана к траектории направляющей движения, а в итоге мяч будет привязан к концу этой направляющей.
8. В слое мяча выделите кадр 12 и с помощью инструмента Arrow перемещайте мяч по рабочему полю вдоль траектории направляющей движения до тех пор, пока он не будет привязан к ее концу, как показано на рис. 7.13.

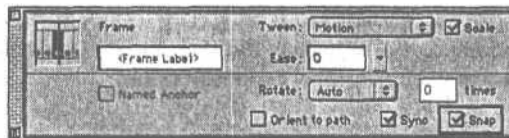


Рис. 7.12. Для привязки точки регистрации промежуточного отображения объекта к направляющей движения на панели инспектора свойств установите флажок в поле опции *Snap*

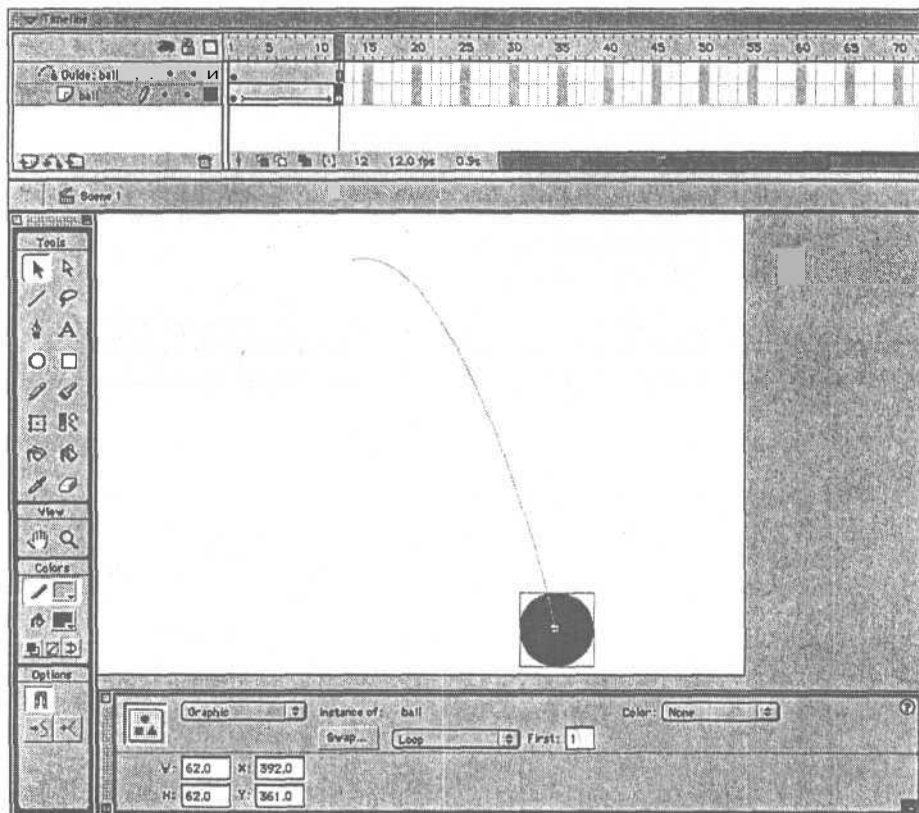


Рис. 7.13. Выделив кадр 12, с помощью инструмента *Arrow* переместите мяч вдоль траектории направляющей движения к ее концу

9. Сохраните файл под именем **ball12.fla**. Воспроизведите анимацию, нажав клавишу **<Return>** (Macintosh) или **<Enter>** (Windows). Теперь мяч должен перемещаться по дуге направляющей движения, как показано на рис. 7.14.

Данная схема будет работать только в том случае, если объекты привязаны к начальной и конечной точке траектории направляющей движения. В опубликованном фильме эта траектория видна не будет, поэтому для упрощения процесса привязки можно совершенно спокойно нарисовать траекторию цветом, резко отличающимся от цвета объекта.

РЕГУЛИРОВКА

Частота смены кадров Flash-фильма является постоянной. При создании промежуточных отображений перемещение объекта равномерно распределяется по числу кадров между ключевыми кадрами. Однако при таком подходе может возникнуть некоторая неестественность движения. Большинство движений начинаются и оканчиваются постепенно, если только в этот процесс не вмешается никакая другая сила, как, например, сила тяжести в примере с падающим мячом. Чтобы симитировать естественные отклонения скорости, в промежуточных отображениях необходимо **откорректировать** параметр регулировки, который корректирует скорость изменения между промежуточными кадрами. Отрицательные значения параметра (от -1 до -100) обозначают высокую стартовую скорость, которая затем плавно снижается, и наоборот, положительные значения (от 1 до 100) соответствуют постепенному ускорению объекта.

Для изменения параметров регулировки в анимации падающего мяча выполните следующие действия.

1. Откройте файл `Ball12.fla`.
2. Щелкните на слое мяча и выделите промежуточный кадр движения.
3. На панели Timeline щелкните на кнопке Onion Skin (Калькирование), чтобы можно было просмотреть анимацию, после чего щелкните на кнопке Onion Marker (Маркер калькирования) и из выпадающего меню выберите пункт Onion All (Калькировать все), как показано на рис. 7.15.

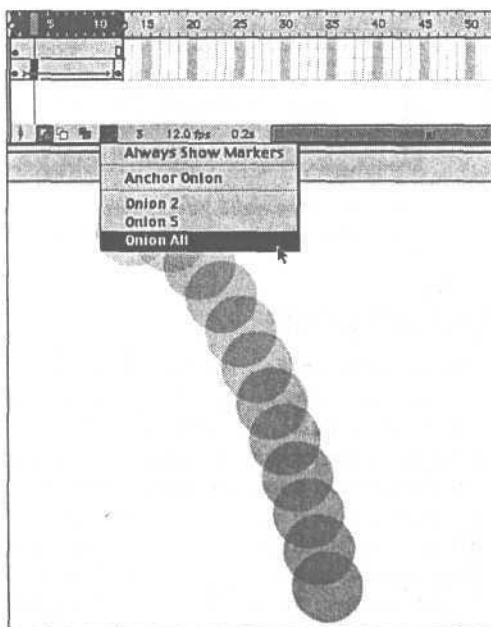
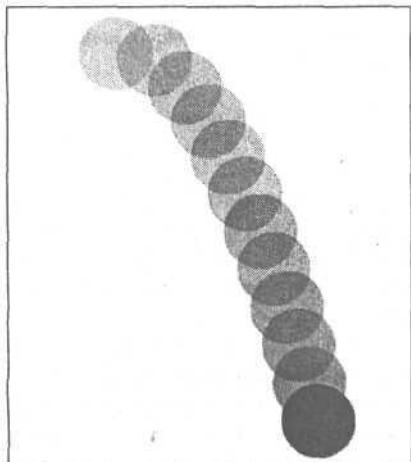


Рис. 7.14. Если мяч привязан к концам направляющей движения, он будет перемещаться по дуге направляющей

Рис. 7.15. Параметры калькирования позволяют просматривать анимацию путем одновременного отображения содержимого нескольких кадров

4. На панели инспектора свойств щелкните на ползунке Ease (Регулировать) и перетащите его вверх, чтобы анимация на рабочем поле возле края происходила быстрее (рис. 7.16). При включенном режиме калькирования изменения скорости представляются в виде изменяющихся расстояний между объектами в различных кадрах. Для рассматриваемого примера с мячом лучше всего подходит значение, равное приблизительно 30.
5. Сохраните файл под именем **ball3.fla** и протестируйте фильм, выполнив команду **Control⇒Test Movie** (Управление⇒Тестировать фильм).

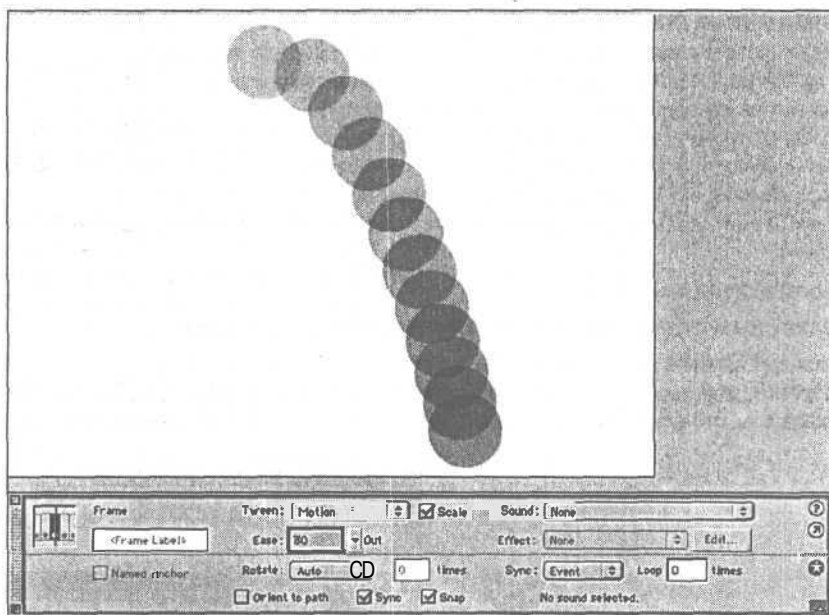


Рис. 7.16. В режиме калькирования изменения регулировки отображаются в виде различных расстояний между объектами в разных кадрах

КАЛЬКИРОВАНИЕ

Как правило, в рабочем поле виден только один кадр. Однако в предыдущем примере мы убедились, что в режиме калькирования одновременно можно просматривать несколько кадров. При создании анимации возможность видеть кадры в контексте является просто бесценной. Поэтому при создании промежуточных отображений как можно чаще включайте режим калькирования для проверки выравнивания и регулировки.

Калькирование очень полезно при корректировке распределения изображений во времени. На рис. 7.17 показан падающий мяч при различном числе кадров и разных скоростях.

Мяч, изображенный слева, падает быстро. Если в режиме калькирования объекты в кадрах не перекрываются, это означает, что анимация настолько быстрая, что объект будет казаться прыгающим с места на место. При этом теряется эффект непрерывности движения. Мяч, изображенный справа, падает медленно, при этом кадры существенно накладываются друг на друга. Мяч, изображенный посередине, падает не очень медленно. Любая из этих скоростей может оказаться востребованной, в зависимости от того, какой тип движения необходимо отобразить.

СОЗДАНИЕ ПРОМЕЖУТОЧНЫХ ОТОБРАЖЕНИЙ с ИЗМЕНЕНИЕМ ФОРМЫ

Во Flash можно интерполировать существенные изменения формы объектов. Построение промежуточных форм создает иллюзию плавного перехода из одной формы в другую. В отличие от промежуточных отображений движения, создание промежуточных отображений формы можно применять только к объектам простой формы. Для применения этого способа экземпляры символов, группы или растровые изображения необходимо разделить.

Для создания промежуточного отображения простой формы выполните следующие действия.

1. Откройте новый документ, воспользовавшись комбинацией клавиш <Cmd+N> (Macintosh) или <Ctrl+N> (Windows).
2. Щелкните в рабочем поле и на панели инспектора свойств измените размеры фильма на 400x200.
3. Выберите инструмент Rectangle (Прямоугольник) и с помощью модификатора задайте любой цвет заливки без штриха. Щелкните в рабочем поле и нарисуйте большой прямоугольник.
4. Откройте панель Align (Выравнивание), воспользовавшись комбинацией клавиш <Cmd+K> (Macintosh) или <Ctrl+K> (Windows). Выделите прямоугольник в рабочем поле, а на панели Align щелкните на кнопке To Stage (К рабочему полю) и разместите прямоугольник в центре рабочего поля по горизонтали и по вертикали. Для этого щелкните на второй и пятой кнопках в строке Align (рис. 7.18).

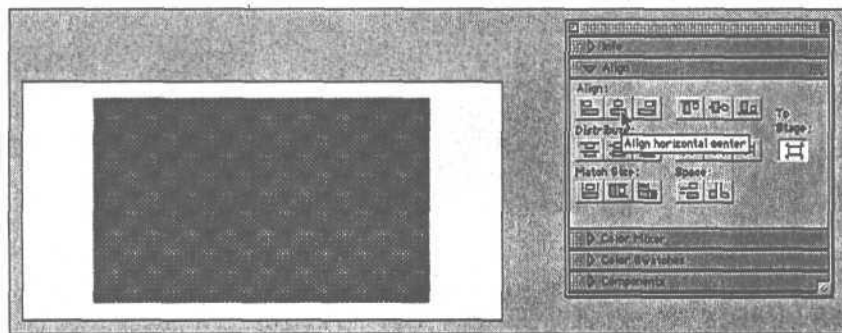


Рис. 7.18. На панели Align щелкните на кнопке To Stage и разместите прямоугольник в центре рабочего поля по горизонтали и по вертикали

5. На панели Timeline перейдите к кадру 15 и нажмите клавишу <F7> для вставки пустого ключевого кадра.
6. Выберите инструмент Text, а на панели инспектора свойств щелкните на кнопке полужирного начертания. Щелкните инструментом Text в рабочем поле и введите свое имя,
7. Выделите текст с помощью инструмента Arrow. С помощью панели Align разместите его в центре рабочего поля по горизонтали и по вертикали, как это делалось на шаге 4.
8. Выделив текст, выполните команду **Modify⇒Break Apart (Изменить⇒Разбить)**. В результате текстовый блок будет разбит на отдельные буквы (рис. 7.19).
9. Выполните команду **Modify⇒Break Apart** второй раз, в результате чего буквы будут преобразованы в изображения (рис. 7.20).
10. Выделите кадр между двумя ключевыми кадрами и на панели инспектора свойств из раскрывающегося меню Tweens (Промежуточные отображения) выберите пункт Shape (Форма), как показано на рис. 7.21.



Рис. 7.19. Чтобы разбить текст на буквы, выделите его и выполните команду **Modify**⇒**Break Apart**



Рис. 7.20. Чтобы преобразовать буквы в изображения, снова выполните команду разбивки текстового блока

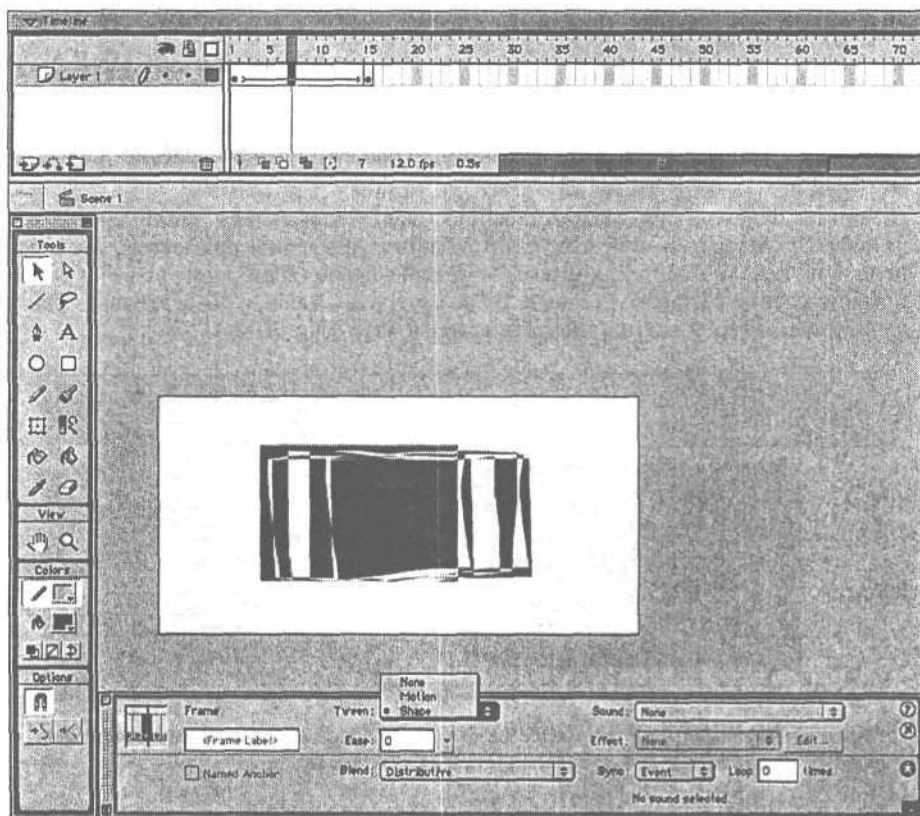


Рис. 7.21. Выделите кадр между двумя ключевыми кадрами и на панели инспектора свойств из раскрывающегося меню **Tween** выберите пункт **Shape**

11. Сохраните фильм и протестируйте его, выполнив команду **Control**⇒**Test Movie**. Вы увидите, что прямоугольник будет преобразован в ваше имя (рис. 7.22).

К промежуточным отображениям формы также можно применить регулировку. Кроме того, на панели инспектора свойств имеется еще одна опция, **Blend** (Сопряжение), которая показана на рис. 7.23. Сопряжение может быть либо дистрибутивным, когда форма при интерполяции сглаживается, либо угловым, когда сохраняются ее резкие края. Угловое смешивание применимо только к формам с четкими краями и углами.

Промежуточное отображение формы является очень мощным средством, но при этом размер файла может существенно увеличиться. Поэтому пользуйтесь им достаточно осторожно и поэкспериментируйте с простыми формами.



Рис. 7.22. Прямоугольник преобразовывается в имя



Рис. 7.23. При промежуточном отображении с изменением формы с помощью опции сопряжения можно сохранить четкие края и углы формы

УКАЗАТЕЛИ ФОРМЫ

При построении промежуточных отображений Flash создает простейшую интерполяцию изображений. Иногда это может привести к странным и нежелательным эффектам, как, например, при преобразовании прямоугольника в ваше имя, рассмотренное в предыдущем примере. Во время интерполяции участки некоторых форм могут выворачиваться. К счастью, вы можете указать точки, которые в начальной и конечной формах и в процессе интерполяции должны оставаться неизменными. При создании промежуточных отображений с изменением формы можно назначить указатели формы. Для этого выполните следующие действия.

1. Выберите в последовательности промежуточных отображений первую форму.
2. Выполните команду **Modify⇒Shape⇒Add Shape Hint** (Изменить⇒Форма⇒Добавить указатель формы). Появится начальный указатель формы, имеющий вид красного кружка с буквой "a".
3. Щелкните на указателе формы инструментом Arrow и перетащите его в точку, которую хотите отметить (рис. 7.24).
4. Щелкните на последнем кадре промежуточных изображений. На рис. 7.25 видно, что соответствующий указатель формы появится в аналогичном месте последнего изображения.
5. Снова протестируйте фильм, выполнив команду **Control⇒Test Movie**, и вы увидите, как указатель формы влияет на промежуточные отображения. При создании промежуточных отображений можно добавить до 26 указателей формы.



Рис. 7.24. Указатели формы имеют вид красных кружков с буквами. Перетащите их в точки формы, которые хотите отметить при создании промежуточных отображений

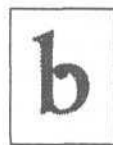


Рис. 7.25. Соответствующий указатель формы появится в том же месте последнего кадра промежуточных отображений

ПЕРЕМЕЩЕНИЕ по ВРЕМЕННОЙ ШКАЛЕ

Сущностью анимации является движение, поэтому предварительный просмотр анимации в движении при работе над фильмом является очень важным моментом. Во Flash можно вручную перемещать воспроизводящую головку по временной шкале вперед и назад, чтобы увидеть создаваемый фильм в развитии. Для просмотра кадров **щелкните** на воспроизводящей головке в верхней части панели Timeline (рис. 7.26) и перетащите ее по кадрам. Текущий

кадр указывает красная линия, проходящая от воспроизводящей головки к нижней части панели Timeline. Продвижение по временной шкале особенно полезно при совершенствовании анимации, поскольку позволяет перемещаться по фильму быстрее или медленнее, в зависимости от того, насколько быстро вы перетаскиваете воспроизводящую головку.



Рис. 7.26. Для перемещения по временной шкале щелкните и перетаскивайте воспроизводящую головку, представляющую собой красный прямоугольник, расположенный в верхней части панели Timeline

Просмотреть фильм в процессе работы можно также с помощью меню Control (Управление) (рис. 7.27). Чтобы просмотреть фильм с заданной частотой смены кадров, из меню Control выберите пункт Play (Воспроизведение) либо нажмите клавишу <Return> (Macintosh) или <Enter> (Windows). Особенно полезной является возможность перемещения по фильму на один кадр вперед (клавишей быстрого доступа является О) или назад (клавишей быстрого доступа является <, >).

МУЛЬТИПЛИКАЦИЯ во FLASH

Векторный формат Flash превосходно подходит для создания мультипликации. Векторная графика напоминает используемые в мультипликации рисунки, а кроме того, в программе Flash ее очень легко преобразовать. Во Flash можно применять многие традиционные методики мультипликации.

СОЗДАНИЕ ЭФФЕКТА ПАНОРАМЫ

Классической методикой анимации, которую можно перенести в Web, является создание эффекта панорамы. Панорама создает иллюзию глубины. В фильме это достигается путем панорамирования камеры либо с помощью перемещения камеры по сцене на фиксированной высоте с целью полного охвата фона. В анимации этот эффект моделируется путем перемещения фона при неподвижном рабочем поле.

Секрет создания убедительной панорамы заключается в создании фоновых элементов, анимированных с различными скоростями. Самые дальние элементы должны двигаться с самой низкой скоростью. Объекты, расположенные на переднем плане, двигаются относительно расположенной позади них сцены, что создает иллюзию глубины.

Для создания эффекта панорамы выполните следующие действия.

1. Создайте фоновый элемент, который будет шире рабочего поля. В идеальном варианте левый и правый края фона должны располагаться так, чтобы при движении фон казался непрерывным.
2. Создайте промежуточное отображение фоновых элементов в рабочем поле так, чтобы в первом ключевом кадре правый край фона совпал с правым краем рабочего поля. В последнем ключевом кадре левый край фона должен совпасть с левым краем рабочего поля. Это создает иллюзию постоянного движения и разрастающегося обширного фона.

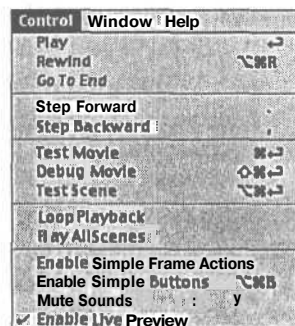


Рис. 7.27. Чтобы просмотреть фильм в процессе работы, выберите нужный пункт из меню Control

ЦИКЛЫ СИМВОЛОВ

Самым простым способом **создания** анимации символов является использование вложенных символов. Иерархическая структура символов идеально подходит для разделения сложной анимации на механические части, которые можно использовать повторно или циклически. Спланируйте, как должен двигаться объект или символ, и разделите каждую движущуюся часть на отдельные символы, которые затем можно будет скомпоновать в головной символ.

Анимацией, реалистичности которой добиться сложнее всего, является ходьба человека. Для достижения максимальной эффективности и реальности такую анимацию можно создать с помощью вложенных символов. Для упрощения процесса проанализируйте походку человека в профиль. Символ цикла ходьбы имеет самый высокий уровень. В нем содержатся слои головы, торса и ног. Голова может представлять собой одно графическое изображение. Видеокадр торса может содержать два **анимированных** экземпляра символа руки, а видеокадр ног — два **анимированных** экземпляра символа ноги. Вместе символы, образующие цикл ходьбы, и являются полным шагом, который можно повторять, не перерисовывая изображения.

ОСНОВНЫЕ ПРАВИЛА ПРИ СОЗДАНИИ АНИМАЦИИ

Любая анимация, будь то классическая мультипликация или сложное движение, выполняемое по определенному сценарию, будет гораздо эффективнее, если при ее создании руководствоваться следующими основополагающими принципами.

- **Распределение во времени и движение.** Распределение во времени придает особое значение перемещению. Скорость, с которой происходит перемещение, может придавать одному и тому же движению разные смысловые оттенки. Представьте себе удар кулаком по столу. В зависимости от того, как реализовать анимацию этого движения (быстро или медленно), можно передать настроение: плохое или игривое.

Распределение во времени и пошаговое перемещение являются основой создания реалистичного и интересного движения. Управляя перемещением объекта во времени, вы можете сделать его более реалистичным, отобразив его атрибуты. Тяжелые объекты сложнее перемещать, а процедура их разгона или торможения занимает больше времени. Далеко не все объекты двигаются с одинаковым шагом, поэтому вам придется задавать разную скорость, с которой выполняется то или иное действие.

Пошаговое продвижение также является важным фактором при создании анимации. Существует очень хрупкое равновесие между процессом ожидания действия, самим действием и реакцией на него. Если шаг продвижения слишком медленный, зритель может потерять интерес к анимации, но если он слишком быстрый, то движение может быть непонятным и неверно проинтерпретированным. Кроме того, поэкспериментируйте, выбирая длину каждой сцены, — они не должны иметь одинаковую продолжительность или нести одинаковую смысловую нагрузку. Обычно основой анимации является действие, но чтобы подчеркнуть его значимость, можно выдержать паузу перед началом действия. С другой стороны, длительное ожидание, сопровождаемое быстрым действием и реакцией на него, может выглядеть комично.

- **Ожидание.** Движение будет наиболее динамичным и реалистичным, если ему будет предшествовать процедура ожидания. Бегуны, прежде чем помчаться по дорожке, немного подаются назад и замирают. По аналогии, в начале движения попробуйте немного переместить объект в противоположном направлении.
- **Завершение.** Движение не может прекратиться мгновенно. Бегуны не останавливаются сразу же после того, как пересекают финишную черту, а продолжают бег, замедляя его

и постепенно останавливаясь. Представьте себе движение руки спортсмена, разыгрывающего мяч (например, в волейболе), после того, как была выполнена подача. Поэтому не прекращайте движение резко, а постарайтесь плавно завершить его немного дальше точки, в которой оно должно было бы полностью прекратиться.

- Деформация. Далеко не все объекты в процессе движения остаются недеформированными. Большинство объектов при движении или при столкновении слегка меняют свою форму. Чтобы сделать отскок мяча более реалистичным, необходимо добавить промежуточную форму, создающую впечатление того, что при падении мяч слегка растягивается (это достигается за счет того, что мяч делается тоньше и длиннее), а при столкновении с землей сплющивается (становится короче и толще).
- Накладывающиеся действия. Действие становится более интересным, если оно варьируется. Следует избегать действий, которые начинаются и заканчиваются одновременно. По возможности комбинируйте движение: пусть второй объект начнет двигаться до того, как остановится первый.
- Вторичное действие. Анимация будет более эффективной, если она порождает волновой эффект. Движение редко происходит автономно. Результатом первичного действия является вторичное действие, выражаемое в отражении или усилении основного действия. Пусть объекты взаимодействуют, что приведет к созданию более сложной анимации.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Я создал направляющий слой и траекторию движения, вдоль которой должна выполняться анимация. Почему она не работает?

Анимация выполняется вдоль направляющей движения только в том случае, если изображение привязано к начальной и конечной точкам траектории направляющей. Проверьте начальный и конечный ключевые кадры и убедитесь, что изображения непосредственно привязаны к траектории.

Я не могу добавить промежуточное отображение с изменением формы. Почему?

Проверьте, являются ли объекты, промежуточное отображение которых вы хотите создать, символами. Чтобы создать промежуточное отображение движения, объекты должны быть символами или быть сгруппированными, а вот промежуточное отображение с изменением формы не может быть применено к группам или символам. Эту проблему можно обойти путем разбивки экземпляра символа. Для этого выделите экземпляр в рабочем поле и воспользуйтесь комбинацией клавиш <Cmd+B> (Macintosh) или <Ctrl+B> (Windows), в результате чего символ будет преобразован в объект. Если символ содержит вложения, его придется разбивать несколько раз. Процедура разбивки относится только к экземпляру символа, а сам символ остается в библиотеке в неизменном состоянии.

Можете дать совет по управлению и пошаговому продвижению сложной анимации?

Если фильм содержит большое количество сложной анимации, особенно анимации персонажей, лучше всего выполнить раскадровку действия. Возьмите карандаш и изобразите анимацию схематически. В особо сложных случаях структуру анимации придется описывать **покадрово**. В противном случае набросайте только главные моменты действия. Таким образом можно спланировать взаимодействие объектов и более точно представить себе распределение во времени.

FLASH ЗАРАБОТОЙ: УПРОЩЕНИЕ АНИМАЦИИ

Математическая точность векторной графики Flash способствует тому, что анимацией может заниматься всякий, кто способен работать с графическими программами. Представьте себе, какого уровня были специалисты и художники, стоявшие у истоков создания мультфильмов Диснея. Благодаря Flash анимация стала гораздо более доступной.

Обратной стороной медали векторного формата Flash является то, что векторы могут быть настолько точными, что потеряют индивидуальность и станут менее интересными. В реальном мире объекты могут иметь дефекты формы, а движение часто бывает неравномерным. Чтобы с помощью Flash создать наиболее правдоподобную анимацию, пользуйтесь функциями программы только в качестве стартовой площадки. Немного подкорректируйте формы, делая их менее "правильными". В полной степени воспользуйтесь функцией регулировки, чтобы создать разнообразие шагов перемещения. В промежуточных отображениях создайте ключевые кадры и комбинируйте покадровую и промежуточную анимацию для получения наиболее реалистичных эффектов. И самое главное, черпайте вдохновение именно из несовершенства окружающего вас мира.

ИСПОЛЬЗОВАНИЕ ЗВУКА

В ЭТОЙ ГЛАВЕ...

Знакомство со звуком	157
Подготовка звука для Flash	158
Импортирование звука	160
Синхронизация звуков	161
Редактирование звука	163
Управление звуком	165
Оптимизация звука	167
Возможные проблемы	169
Flash за работой: отключение звука	170

ЗНАКОМСТВО со ЗВУКОМ

Звук придает фильмам Macromedia Flash глубину и резонанс. Даже до того, как в фильмы можно было встраивать звуковые дорожки, для акцентирования внимания на каком-то действии в немых фильмах использовалось фоновое звуковое сопровождение. Чтобы в полной мере оценить значимость звукового сопровождения, представьте себе, как смотрелся бы фильм "Челюсти" без угрожающей музыки.

Между восприятием видеоинформации и восприятием звука существует несколько существенных отличий. Обычно человек мигает около 20 раз в минуту, а каждое мигание длится приблизительно четверть секунды. В связи с этим мозг приучен к прерыванию потока визуальных данных и приспособлен к заполнению пауз в визуальной информации. Способность к

соединению визуальных точек позволяет воспринимать **фильмы**, которые представляют собой набор статических изображений, отображаемых в виде быстрой последовательности, что создает иллюзию непрерывного движения.

Мозг человека не является столь же снисходительным к заполнению пауз при восприятии аудиоинформации. Наоборот, человек очень чувствителен к вариациям звука. Хотя отображение фильма реализуется в виде статических кадров, звуковое сопровождение подается в виде непрерывного потока или звуковой дорожки. Однако особенности передачи содержимого Web затрудняют имитацию непрерывной звуковой дорожки.

Звук создается при колебаниях объектов. Эти колебания смещают частицы воздуха и переносятся в пространстве в виде звуковых волн. Когда эти волны достигают барабанной перепонки, мозг воспринимает колебания как звук. Звуки отличаются друг от друга частотой, т.е. скоростью колебания атмосферного давления. Чем больше колебаний, тем выше их частота, и при этом тон звука воспринимается как более высокий. Амплитудой является уровень давления воздуха. Более высокие уровни давления воспринимаются как более громкий звук. Звуковая волна изображена на рис. 8.1. Этот момент является очень важным, так как для подачи звука программа Flash должна воспроизвести звуковые волны.

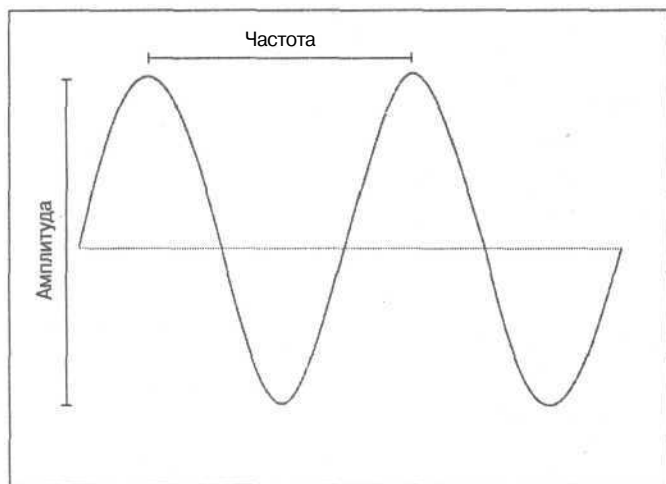


Рис. 8.1. "Анатомия" звуковой волны. Частота определяет тон, а амплитуда — громкость

ПОДГОТОВКА ЗВУКА для FLASH

Чтобы иметь возможность воспроизвести звук, звуковые волны дискретизируют. Дискретизация представляет собой процесс разделения звуковых волн на отдельные составляющие (звуковые биты) и как можно более частое воспроизведение этих выборок (рис. 8.2). Число выборок в секунду называется частотой дискретизации и измеряется в герцах (Гц).

Процесс дискретизации может показаться вам знакомым, так как он подобен созданию анимации с помощью кадров. Понятие частоты дискретизации аналогично частоте смены кадров фильма (выражаемое в кадрах в секунду). Это понятие также аналогично точечному разрешению. Чем выше частота дискретизации (чем большей является выборка), тем точнее будет воспроизведена исходная звуковая волна. В табл. 8.1 указаны стандартные частоты дискретизации и соответствующие им значения точности воспроизведения, т.е. качества звука.

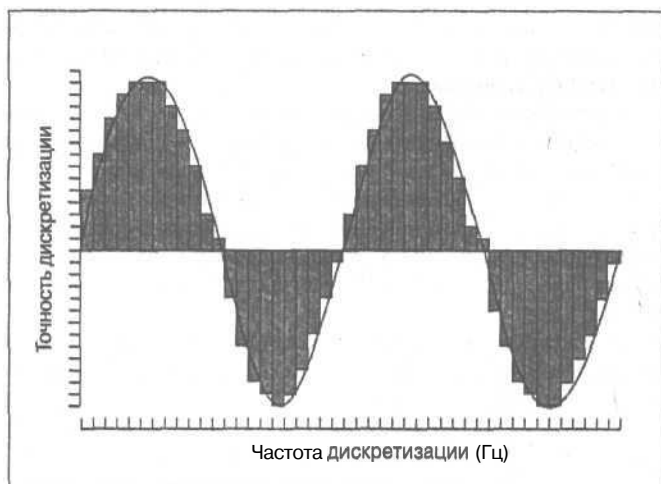


Рис. 8.2. Дискретизация звука: серые полосы представляют собой выборку звуковой волны

ТАБЛИЦА 8.1. ЧАСТОТЫ ДИСКРЕТИЗАЦИИ ЗВУКА

ЧАСТОТА ДИСКРЕТИЗАЦИИ	КАЧЕСТВО ЗВУКА
44,1 КГц	Качество компакт-диска
22,05 КГц	Качество радио FM-диапазона; широко используется при воспроизведении звука в Web
11,025 КГц	Самое низкое качество, рекомендуемое для коротких музыкальных фрагментов; высокое качество речи
5 КГц	Низкое качество воспроизведения речи

Точность дискретизации, или битовая глубина, определяет, сколько разных градаций или дискретных уровней амплитуд применяется при выполнении дискретизации звука. Это понятие аналогично битовой глубине растровых изображений. Чем больше битовая глубина, тем больше данных хранится в каждом колебательном сигнале. В табл. 8.2 представлены значения битовой глубины звука и соответствующее им качество звука.

ТАБЛИЦА 8.2. БИТОВАЯ ГЛУБИНА ЗВУКА

БИТОВАЯ ГЛУБИНА	ЧАСТОТА ДИСКРЕТИЗАЦИИ (ЧИСЛО ИМПУЛЬСОВ В СЕКУНДУ)	КАЧЕСТВО ЗВУКА
16	65 536	Качество компакт-диска
12	4 096	Среднее, близкое к компакт-диску
8	256	Качество радио FM-диапазона
4	16	Самое низкое, допустимое для воспроизведения музыки

Вполне естественно, что высокая частота дискретизации и большая битовая глубина приводят к увеличению размеров звуковых файлов. Однако обе эти составляющие можно регулировать независимо друг от друга в целях уменьшения размеров файлов. По сравнению с музыкой речь имеет меньше тональных отклонений, но больше колебаний амплитуды или звука из-за пауз между словами и предложениями. Во время этих пауз может возникать нежелательное шипение и шумы. Следовательно, для уменьшения шумов и шипения во время пауз речевые файлы должны иметь меньшую частоту дискретизации и большую битовую глубину.

И наоборот, музыка имеет гораздо больший диапазон тональности или частоты, поэтому для воспроизведения тоновых вариаций требуется более высокая частота дискретизации. Как правило, музыка имеет более узкий диапазон громкости по сравнению с речью, поэтому битовая глубина может быть меньше.

Кроме того, большое влияние на размер файла оказывает выбор между стерео- и монозвуком. Монозвук записывается на одной звуковой дорожке или в одном канале. Если имеются два динамика, то в них будет звучать один моноканал. Стереозвук записывается в двух каналах, что способствует лучшему воспроизведению эффекта слуха человека. Совершенно очевидно, что два **аудиоканала** в два раза увеличивают размер файла.

ИМПОРТИРОВАНИЕ ЗВУКА

Flash позволяет импортировать элементы, созданные в других приложениях (например, звуковые файлы или растровые изображения), и затем использовать их в фильмах. Звуки, импортируемые во Flash, помещаются в библиотеку. Во Flash MX можно импортировать звуковые файлы следующих форматов.

- WAV (только Windows). Формат Waveform Audio File Format разработан для платформы Windows и является для нее стандартным форматом звуковых файлов.
- AIFF (только Macintosh). Формат Audio Interchange File Format разработан компанией Apple и является стандартным форматом звуковых файлов для Macintosh.
- MP3 (Macintosh и Windows). Формат **MPEG-1 Audio Layer-3** обеспечивает значительное сжатие звуковых данных без ущерба для качества звука. С помощью MP3-сжатия исходные данные компакт-диска можно сжать в 12 раз.

Если на компьютере установлена надстройка QuickTime 4 или последующей версии, можно также импортировать файлы следующих форматов.

- AIFF (Windows или Macintosh).
- Sound Designer II (только Macintosh).
- Звук фильмов QuickTime (Windows или Macintosh),
- Sun AU (Windows или Macintosh).
- System 7 Sounds (только Macintosh).
- WAV (Windows или Macintosh).

MP3 является единственным импортируемым звуковым форматом, который предварительно сжат. Следовательно, файлы формата MP3 меньше файлов WAV или AIFF. Для импортирования звукового файла выполните команду **File⇒Import to Library** (**Файл⇒Импортировать в библиотеку**) и в открывшемся диалоговом окне Import to Library (рис. 8.3) укажите нужный звуковой файл.

Выберите звуковой файл и щелкните на кнопке Open (Открыть). Файл будет добавлен в библиотеку, как показано на рис. 8.4. Чтобы предварительно прослушать звуковой файл, выберите его в библиотеке и щелкните на кнопке с изображением треугольника, расположенной в верхней части окна просмотра.

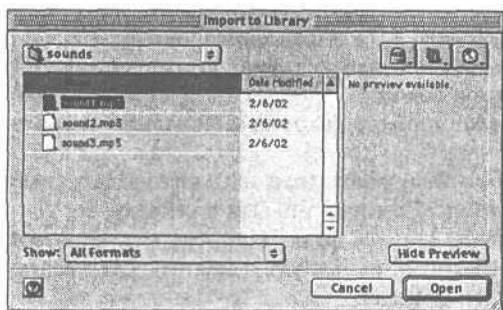


Рис. 8.3. Чтобы импортировать звуковой файл в библиотеку, выполните команду **File ⇒ Import to Library**

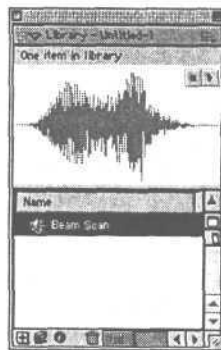


Рис. 8.4. Импортированные звуки добавляются в библиотеку

Лучше всего поместить каждый звуковой файл в отдельный слой. Для добавления слоя в фильм выполните команду **Insert ⇒ Layer** (Вставка ⇒ Создать слой) или щелкните на пиктограмме **Insert Layer** (Вставить слой), расположенной под списком слоев на временной шкале. Выделив слой, перетащите звуковой файл с панели **Library** (Библиотека) в рабочее поле. (Во Flash MX имеются образцы звуков, доступ к которым осуществляется с помощью команды **Window ⇒ Common Libraries ⇒ Sounds** (Окно ⇒ Общие библиотеки ⇒ Звуки). При этом откроется библиотека звуков.) На рис. 8.5 показано, что звук будет помещен в выделенный слой.

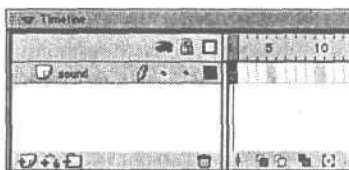


Рис. 8.5. При перетаскивании звука в рабочее поле он будет помещен в выделенный слой

Свойства звукового файла отображаются на панели инспектора свойств, как показано на рис. 8.6. Обратите внимание на то, что в правом нижнем углу панели указаны значения частоты дискретизации звука, число звуковых каналов и битовая глубина.

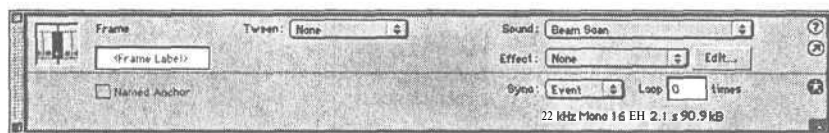


Рис. 8.6. Атрибуты звука указаны и доступны для редактирования на панели инспектора свойств

СИНХРОНИЗАЦИЯ ЗВУКОВ

Важно понимать, что в отличие от анимации, звук реализуется не посредством кадров, а во времени. Звук передается с одинаковой скоростью, поскольку это необходимо для сохранения его целостности. При изменении скорости передачи изменится частота, и в результате

будет получен совершенно другой звук. В отличие от этого, анимацию можно запускать, останавливать и переходить от кадра к кадру.

Как же синхронизировать звук с анимацией, если управляются они совершенно по-разному? К сожалению, не существует способа абсолютно точной синхронизации звука и изображения, такой как при наличии звуковой дорожки, внедренной в кинофильм. Максимум, что можно сделать, — это поместить звук в то место, в котором он должен находиться, и поработать над оптимизацией его передачи.

При работе с содержимым, размещаемым в Web, необходимо идти на определенные компромиссы. Если частота дискретизации составляет тысячи выборок в секунду, то, как и видеоматериалы, звук будет влиять на увеличение размера файла Flash-фильма. При создании файлов озвученной анимации, которые необходимо разместить в Internet, следует отдавать предпочтение чему-то одному: или звуку, или анимации.

ЧТО ПЕРВИЧНО? ЗВУК ИЛИ АНИМАЦИЯ

Для того чтобы поместить звук в фильм, в программе Flash существуют два способа, позволяющие **определить**, что именно будет первичным в фильме: звук или анимация. Событийные звуки являются второстепенными по отношению к анимации. В этом случае воспроизведение звука инициируется **какими-либо** событиями, например, наведением указателя мыши на кнопку или достижением определенного кадра во временной шкале при воспроизведении фильма. После запуска событийные звуки будут воспроизводиться абсолютно независимо от временной шкалы, даже если воспроизведение самого фильма будет прекращено. Если при воспроизведении событийного звука повторится само инициирующее событие, то первый звук продолжит звучание, но одновременно с ним будет запущен и второй экземпляр звука.

Событийные звуки должны быть полностью загружены до того, как начнется их воспроизведение. Этот факт необходимо принимать во внимание при их использовании. Если событийный звук поместить в первый кадр фильма, то пользователю, вероятнее всего, придется подождать завершения загрузки звука.

Другим способом синхронизации является использование потокового звука. Потоковый звук близок к звуковым дорожкам. В этом случае в фильме отдается предпочтение именно звуку, и Flash может при необходимости даже отбросить кадры анимации. Потоковый звук привязан к временной шкале. Такой звук воспроизводится и останавливается при воспроизведении или остановке фильма, и если **временная** шкала содержит меньше кадров, чем занимает звук, то звук прекратится после воспроизведения последнего кадра временной шкалы. Потоковый звук начинает воспроизводиться с момента начала загрузки, поэтому его часто используют в качестве непрерывного фонового озвучивания.

При помещении звукового файла в рабочее поле на панели инспектора свойств необходимо указать один из параметров синхронизации (рис. 8.7).



Рис. 8.7. Доступ к параметрам синхронизации осуществляется на панели инспектора свойств

ПАРАМЕТРЫ СИНХРОНИЗАЦИИ

Параметрами синхронизации являются Event (Событие), Start (Запуск), Stop (Остановка) и Stream (Поток). При выборе параметра Event звук будет синхронизован с ключевым кадром, в который он помещен. В этой роли может выступать и кадр состояния кнопки. Параметр Start назначает событийный звук и препятствует наложению звука при повторении ини-

цирующего события. При выборе параметра Start одновременно может воспроизводиться только один звук. Параметр Stop останавливает любой событийный звук. При выборе параметра Stream звук синхронизируется с анимацией.

При помещении звука в рабочее поле обратите внимание на то, что экземпляр звука займет только один кадр на временной шкале (см. рис. 8.5). Это происходит потому, что для полного воспроизведения событийного звука требуется только один кадр временной шкалы. И наоборот, потоковый звук воспроизводится в нескольких кадрах временной шкалы, содержащих этот звук. При помещении звука в рабочее поле Flash никоим образом не сможет узнать, как именно вы собираетесь синхронизировать этот звук. При использовании потокового звука для его полного воспроизведения к слою звука потребуется добавить кадры, как показано на рис. 8.8. В слое звука выделите кадр, например 30-й, и для вставки кадров нажмите клавишу <F5>. Во всех кадрах, занимаемых звуком, появится изображение звуковой волны. Для корректировки длительности звука вам, возможно, придется добавить еще несколько кадров или, наоборот, удалить лишние.

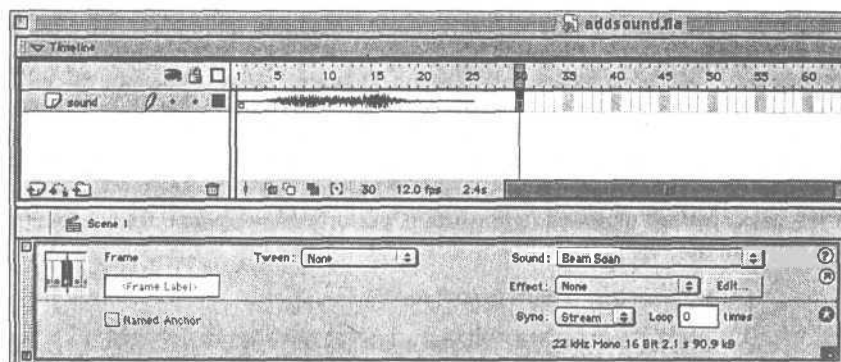


Рис. 8.8. Для корректировки длительности потокового звука придется добавить кадры к временной шкале

РЕДАКТИРОВАНИЕ ЗВУКА

После размещения звука есть несколько возможностей отредактировать его. Несмотря на то что во Flash нет возможности полноценного редактирования звука (т.е. когда можно изменить саму звуковую волну), можно изменить способ воспроизведения звука. Для этого на панели инспектора свойств имеется раскрывающееся меню Effect (Эффект), показанное на рис. 8.9, позволяющее регулировать громкость воспроизведения (опции Fade In (Увеличение) и Fade Out (Уменьшение)).

При выборе опции None (Нет) (по умолчанию) к звуку не будут применяться никакие эффекты либо будут удалены ранее заданные эффекты. Параметры Left Channel (Левый канал) и Right Channel (Правый канал) позволяют воспроизводить только один из звуковых каналов. При выборе параметра Fade Left to Right (Затухание слева направо) или Fade Right to Left (Затухание справа налево) звук смещается из одного звукового канала в другой, что создает эффект панорамирования. При выборе параметра Fade In (Увеличение) в процессе воспроизведения будет плавно повышаться громкость (амплитуда) звука. При выборе параметра Fade Out (Уменьшение) громкость звука будет плавно уменьшаться.

При выборе параметра Custom (Настройка) или после щелчка на кнопке Edit (Редактирование) откроется диалоговое окно Edit Envelope (Редактировать огибающую) с более сложными элементами управления громкостью (рис. 8.10).

Чтобы изменить начало воспроизведения звука в звуковой волне, перетащите ползунок Time In (Время вступления) на середину диалогового окна Edit Envelope. В конце полосы воспроизведения есть соответствующий ползунок Time Out (Время окончания). С помощью этих элементов управления можно воспроизвести только часть звука.

Маркеры огибающей, расположенные в звуковом канале и представляющие собой квадраты, которые можно перетаскивать, позволяют регулировать громкость в заданном канале. После открытия диалогового окна Edit Envelope в каждом канале имеется по одному маркеру. Чтобы изменить громкость звука в целом, перетащите маркер вверх или вниз. Для постепенного увеличения или уменьшения громкости щелкните в строке огибающей, чтобы создать дополнительные маркеры (их максимальное число равно 8). Дополнительные маркеры можно перетаскивать и тем самым независимым образом регулировать громкость отдельных участков звука.

Кнопки Zoom In (Увеличить масштаб) и Zoom Out (Уменьшить масштаб) используются соответственно для увеличения участков звукового файла или для просмотра всей звуковой волны. После щелчка на кнопке Seconds (Секунды) или Frames (Кадры) звук можно соотносить со временем или с количеством кадров. Кнопки Play (Воспроизвести) и Stop (Остановить) используются для предварительного прослушивания внесенных правок. Если вы удовлетворены внесенными изменениями, щелкните на кнопке OK.

Звук можно воспроизводить циклически. Выделив экземпляр звука, на панели инспектора свойств в поле Loop (Цикл) введите числовое значение, как показано на рис. 8.11. Если вы хотите, чтобы звук воспроизводился непрерывно, введите достаточно большое число, и тогда звук будет длительным.



Рис. 8.9. Раскрывающееся меню Effect на панели инспектора свойств позволяет применять основные эффекты к экземплярам звука



Рис. 8.10. Диалоговое окно Edit Envelope позволяет управлять звуковыми эффектами

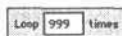


Рис. 8.11. Чтобы звук воспроизводился повторно, на панели инспектора свойств в поле Loop введите какое-либо число

При циклическом воспроизведении потокового звука в конец временной шкалы **будут** добавляться кадры с целью реализации цикличности. Это приведет к увеличению размера файла фильма.

УПРАВЛЕНИЕ ЗВУКОМ

Начало и окончание воспроизведения звука регулируются с помощью ключевых кадров на временной шкале или путем доступа к объекту Sound (Звук) с помощью **ActionScript**. Несмотря на то что звук реализуется во времени, его начало и окончание можно привязать к содержанию других кадров, используя ключевые кадры. Таким образом звук можно синхронизировать с анимацией. Объект Sound позволяет расширить возможности управления воспроизведением звука.

КЛЮЧЕВЫЕ КАДРЫ

Ключевые кадры используются для начала и завершения воспроизведения звука в синхронизации с анимацией. Для добавления звука к анимации выполните следующие действия.

1. Создайте новый слой для звука и вставьте в него ключевой кадр в месте предполагаемого начала воспроизведения звука (рис. 8.12).

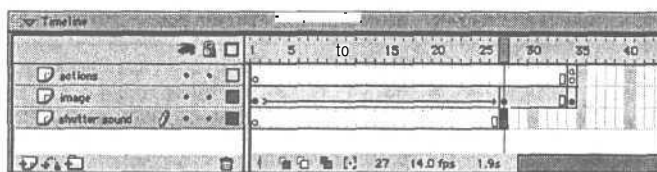


Рис. 8.12. В слое звука добавьте ключевой кадр там, где будет начинаться воспроизведение

2. Выделив ключевой кадр в слое звука, перетащите звук из библиотеки в рабочее поле, в результате чего звук будет добавлен во временную шкалу (рис. 8.13).

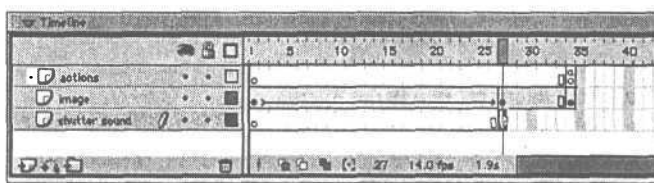


Рис. 8.13. Выделив ключевой кадр в слое звука, перетащите экземпляр звука из библиотеки в рабочее поле

3. Вставьте ключевой кадр в слой звука там, где необходимо завершить воспроизведение (рис. 8.14).
4. На панели инспектора свойств выберите параметр синхронизации звука и для предварительного прослушивания звука во временной шкале нажмите клавишу **<Return>** (Macintosh) или **<Enter>** (Windows).

Особенно важно предварительно прослушать звук, если вы хотите его уменьшить (остановить до окончательного завершения). Это позволит убедиться, что звучание является

приемлемым, и избежать ошибочного прерывания, отвлекающего внимание пользователя от содержимого фильма. Поэкспериментируйте с размещением ключевого кадра, останавливающего звук, и добейтесь, чтобы остановка не казалась преждевременной или неожиданной (если только вы не хотите добиться именно этого эффекта).

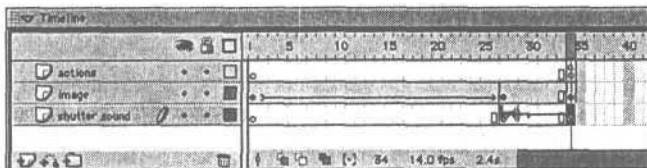


Рис. 8.14. Чтобы остановить воспроизведение звука, вставьте ключевой кадр в нужное место слоя звука

СВЯЗЫВАНИЕ И ОБЪЕКТ SOUND

Звук можно добавить и отрегулировать с помощью объекта Sound (Звук) и языка ActionScript. Использование этого объекта позволяет пользователю самостоятельно регулировать воспроизведение звука в фильмах. Объект Sound позволяет включать и отключать звук, изменять его громкость или панорамирование, а также начинать воспроизведение одного звука после завершения другого.

Совет

Чтобы узнать, как использовать ActionScript для управления звуком, обратитесь к главе 16 "Взаимодействие, события и установление последовательности" и к главе 20 "Использование встроенных объектов фильмов".

Чтобы получить доступ к звуку с помощью объекта Sound, на панели Library (Библиотека) для образца звука необходимо назначить идентификатор. Откройте панель Library и выделите необходимый звук. Щелкните на нем правой кнопкой мыши (Windows) или, удерживая клавишу <Ctrl> (Macintosh), из всплывающего меню выберите пункт **Linkage** (Связывание), как показано на рис. 8.15. Команду Linkage можно также выбрать из меню параметров панели Library, расположенного в ее правом верхнем углу.

Установите флажок в поле опции Export for ActionScript (Экспортировать для ActionScript). При этом имя символа звука будет указано в поле Identifier (Идентификатор), а кроме того, будет автоматически установлен флажок в поле опции Export in First Frame (Экспортировать в первый кадр) (рис. 8.16).

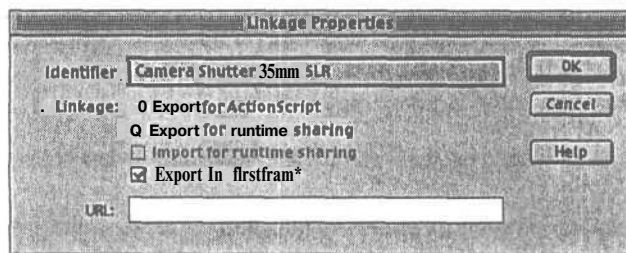


Рис. 8.16. Чтобы указать идентификатор связи, в диалоговом окне Linkage Properties установите флажок в поле опции Export for ActionScript

Внимание!

Все экспортируемые звуки загружаются в первый кадр фильма — не тогда, когда их присоединяют или воспроизводят. Это может привести к длительной загрузке и задержке перед началом воспроизведения фильма.

Может возникнуть необходимость внести изменение в поле Identifier, в которое автоматически вносится имя символа. Поле с отредактированным идентификатором изображено на рис. 8.17. Присвойте идентификатору простое и описательное имя и не используйте в нем символы пробелов, даже если имя состоит из нескольких слов.



Рис. 8.15. Чтобы открыть диалоговое окно *Linkage Properties* (Свойства связей), на панели *Library* щелкните на имени символа правой кнопкой мыши (Windows)



Рис. 8.17. Возможно, вам понадобится изменить имя идентификатора, которое автоматически указывается при установке флажка в поле опции *Export for ActionScript*

После внесения всех необходимых правок щелкните на кнопке **ОК**, чтобы закрыть диалоговое окно *Linkage Properties*.

Альтернативой присоединению звука является его помещение в отдельный фильм (внешний SWF-файл), который при необходимости можно импортировать и загрузить с помощью функции *loadMovie*. Этот подход позволяет во время загрузки звука воспроизводить основной фильм.

ОПТИМИЗАЦИЯ ЗВУКА

Поскольку звук может существенно повлиять на размер файла Flash-фильма, его необходимо оптимизировать. В принципе оптимизация должна начинаться еще до импортирования звука. Так как во Flash функции редактирования звука ограничены, лучше всего воспользоваться специальными приложениями, если, конечно, у вас есть к ним доступ.

Звуки можно сжимать индивидуально, чтобы файлы имели минимально возможные размеры. Чтобы создать индивидуальные параметры сжатия, дважды щелкните на пиктограмме

символа звука на панели Library, либо щелкните на символе правой кнопкой мыши (Windows), либо, удерживая нажатой клавишу <Ctrl> (Macintosh), из контекстного меню выберите пункт Properties (Свойства). В результате откроется диалоговое окно Sound Properties (Свойства звука), изображенное на рис. 8.18.

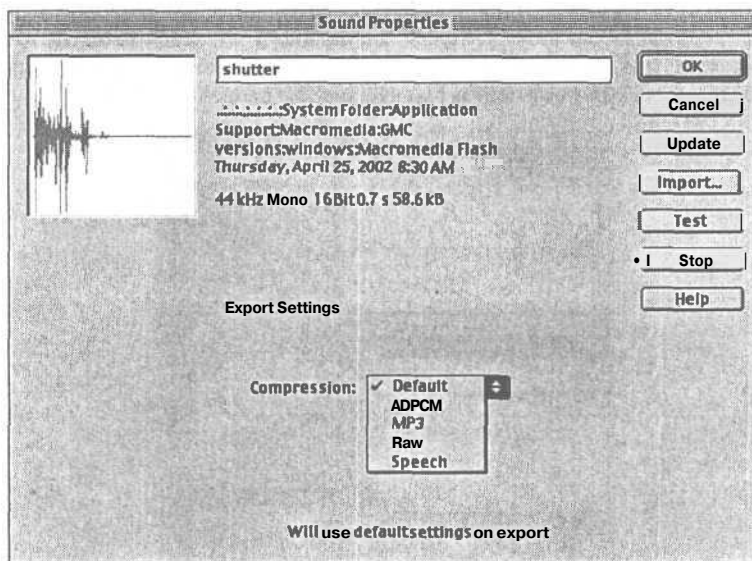


Рис. 8.18. Каждый звуковой файл можно сжать индивидуально, выбрав нужные параметры в диалоговом окне Sound Properties

Из раскрывающегося меню Compression (Сжатие) выберите нужный параметр. При выборе первого из них, Default (По умолчанию), будут использоваться глобальные параметры сжатия, заданные в диалоговом окне Publish Settings (Параметры публикации). По умолчанию параметром сжатия для Flash является MP3 16-bit mono, однако в диалоговом окне Publish Settings эту настройку можно изменить. Если в диалоговом окне Sound Properties выбрать опцию Default, то никакие другие параметры сжатия доступны больше не будут.

ADPCM

ADPCM (Adaptive Differential Pulse Code Modulation — адаптивная дифференциальная импульсно-кодовая модуляция) сжимает 8- или 16-битовые звуковые данные. При выборе этого параметра в диалоговом окне Sound Properties становится доступным ряд дополнительных параметров (рис. 8.19). Сжатие ADPCM часто используется для экспортирования коротких событийных звуков, подобных тем, которые используются при щелчке на кнопках.

Опция Convert Stereo to Mono (Преобразовать стерео в моно) используется для сжатия двух звуковых каналов в один. На монозвук эта опция не влияет.

Частота дискретизации 22 КГц является стандартной для Web и равна половине частоты 44 КГц, являющейся стандартом для компакт-дисков. Увеличить значение исходной частоты дискретизации звукового файла нельзя.

Однако можно уменьшить битовую глубину, что уменьшит диапазон отклонений в пределах выборки звука. Уменьшение битовой глубины и частоты дискретизации способствует уменьшению размера файла, но понижает качество звука. Поэтому, воспользовавшись кнопкой Test (Тест), обязательно прослушайте звук при задании разных параметров.

MP3-СЖАТИЕ

В диалоговом окне Sound Properties для звуковых файлов можно выбрать параметр MP3-сжатия. Этот тип сжатия позволяет получить минимальные размеры звуковых файлов при наилучшем качестве звука, поэтому естественно, что этот параметр применяется чаще всего. Данный тип сжатия является наилучшим для длительных и потоковых звуковых файлов.

Выберите пункт MP3 из раскрывающегося меню Compression. При этом можно выбрать либо исходные параметры MP3-сжатия импортированного звука, установив флажок в поле опции Use Imported MP3 Quality (Использовать качество MP3 импортированного файла), либо детализировать эти параметры (рис. 8.20).

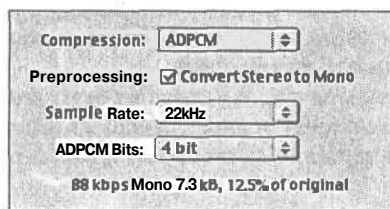


Рис. 8.19. Параметры сжатия ADPCM

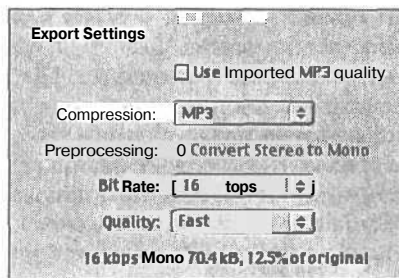


Рис. 8.20. Параметры MP3-сжатия

Чтобы отредактировать параметры, убедитесь, что не установлен флажок в поле опции Use Imported MP3 Quality. Затем перейдите к меню Bit Rate (Скорость передачи), которое определяет скорость передачи экспортируемого звука в битах в секунду. Для поддержания должного качества звука из данного меню необходимо выбрать значение 16 kbps или выше.

При выборе MP3-сжатия можно также сжимать стереодорожки в моно. Наконец, из раскрывающегося меню Quality (Качество) можно выбрать три параметра: Fast (Быстрое), Normal (Обычное) и Best (Наилучшее). Чтобы получить наилучшее сжатие, протестируйте звуковой файл с различными параметрами качества.

Особенно важным является тщательное сжатие потокового звука. Несмотря на то что параметры сжатия потоковых звуковых файлов можно задать индивидуально, все потоковые файлы фильма экспортируются вместе в один потоковый файл, к которому применяется самое высокое значение параметров сжатия из тех, что задавались для отдельных файлов. Потоковый звук должен быть хорошо сжат, иначе размер полученного файла будет огромным.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему кажется, что качество событийных звуков выше качества потокового звука?

Вероятно, вы обратили внимание на то, что звуки, синхронизированные с событиями, звучат лучше, чем потоковые. Это происходит потому, что событийные звуки полностью загружаются до начала воспроизведения, что обеспечивает сохранность всех звуковых данных. Воспроизведение потоковых звуков начинается с момента их загрузки, и при этом они используют полосу пропускания канала совместно с изображениями, анимацией и видео, которые загружаются параллельно. Преимущество отдается потоковому звуку, но некоторые звуковые данные при этом все же теряются.

Есть ли какой-то способ предотвратить пропуск кадров при использовании потокового звука ?

При использовании потокового звука, чтобы анимация не отставала от звука, некоторые кадры могут быть пропущены. Однако кадры анимации можно сохранить, выбрав меньшую частоту смены кадров. Воспроизведение фильма будет соответствовать указанной частоте смены кадров. При более медленном воспроизведении отводится больше времени на загрузку элементов фильма.

Как лучше всего синхронизировать звук и анимацию ?

Лучше всего управлять звуком, запуская его из временной шкалы. По возможности разбейте звук на короткие сегменты, которые можно поместить в кадры и скоординировать с анимированными частями фильма. Благодаря краткости такими звуками будет легче управлять и привязывать их к определенным кадрам. Представьте себе синхронизацию песни Happy Birthday с анимированным поющим персонажем. Лучше всего разбить песню на отдельные слова и строфы, которые затем можно индивидуально синхронизировать с анимацией.

Во Flash можно даже вручную микшировать короткие звуки, помещая их в отдельные слои и накладывая участки этих слоев друг на друга. Это позволяет полностью управлять короткими звуками и создавать более сложное и изысканное звучание, когда звуки синхронизируются не только с анимацией, но и друг с другом.

FLASH ЗА РАБОТОЙ: ОТКЛЮЧЕНИЕ ЗВУКА

В связи с тем, что звук является столь объемным дополнением к фильму, пользователь должен иметь возможность отключать его. Вряд ли вы захотите, чтобы посетители ушли с узла из-за того, что им не понравилась музыка.

При использовании потокового звука обязательно добавьте кнопку для его отключения. Это очень легко сделать даже тем, кто не силен в написании кода. Чтобы отключить звук, выполните следующие действия.

1. Создайте кнопку и выделите ее в рабочем поле.
2. Откройте панель Actions (Действия), нажав клавишу <F9>. Если вы не очень хорошо знакомы с языком ActionScript, перейдите в обычный режим, воспользовавшись комбинацией клавиш <Cmd+N> (Macintosh) или <Ctrl+N> (Windows).
3. В левой части окна Actions откройте папку Actions, дважды щелкнув на ней. Затем дважды щелкните на подпапке Movie Control (Управление фильмом). Дважды щелкните на элементе stopAllSounds (Остановить все звуки).
4. При этом действие stopAllSounds будет указано в правой части окна Actions и добавится к выделенной кнопке (рис. 8.21).
5. Протестируйте фильм, нажав комбинацию клавиш <Ctrl+Enter> (Windows) или <Cmd+Return> (Macintosh). После щелчка на кнопке воспроизведение всех звуков должно прекратиться.

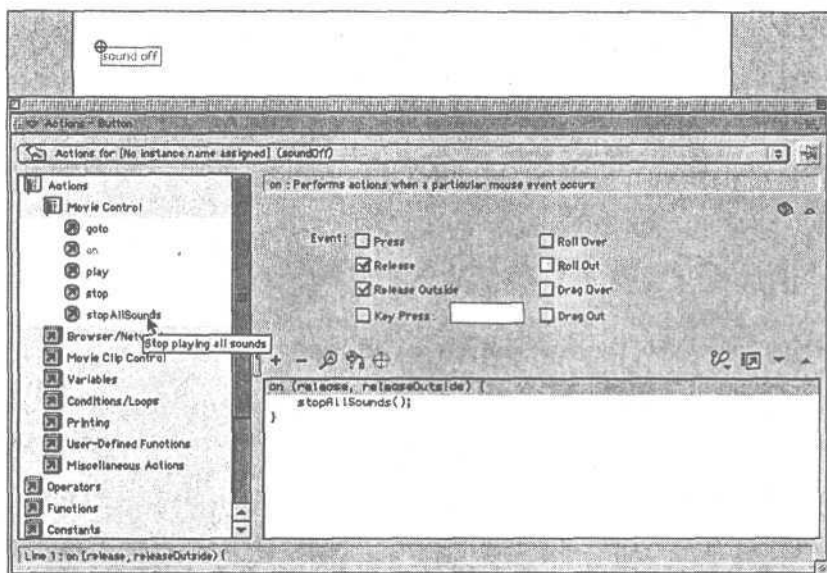


Рис. 8.21. Добавление действия *stopAllSounds* к кнопке позволит пользователям самостоятельно отключать звук в фильме

КНОПКИ, МЕНЮ И ПОЛЯ ВВОДА ДАННЫХ

В ЭТОЙ ГЛАВЕ...

Интерактивность Flash	173
Кнопки	174
Меню	180
Использование компонентов	182
Возможные проблемы	184
Flash за работой: команда Open as Library	185

ИНТЕРАКТИВНОСТЬ FLASH

Flash обеспечивает исключительное сочетание графики, анимации и интерактивности. Во Flash можно не только создавать графику и анимированное содержимое, но и с помощью языка сценариев ActionScript обеспечить пользователям возможность самостоятельно управлять подачей этого содержимого.

Совет

Подробная информация о языке ActionScript приводится в главе 11 "Знакомство с ActionScript".

Для Flash-фильмов можно создавать разнообразные сценарии, выполняемые в ответ на те или иные действия пользователей, начиная от щелчка мышью и заканчивая вводом данных в текстовые поля. Реализация интерактивности может быть как относительно простой (переход на другой кадр при воспроизведении фильма после щелчка мышью), так и сложной, например с использованием динамически загружаемых меню.

Во Flash MX создать интерактивные элементы проще, чем где-либо еще. Например, для создания интерактивных кнопок и видеоклипов можно легко использовать символы. Во Flash MX имеются встроенные компоненты: видеоклипы с предварительно заданными пара-

метрами стандартных элементов, таких как кнопки и элементы форм. Чтобы быстро создать формы и получить данные, вводимые пользователем, достаточно просто **перетащить** элементы в **фильм**, а затем задать параметры соответствующих сценариев.

Кнопки

Кнопки представляют собой интерактивные видеоклипы, состоящие из четырех кадров. Они обеспечивают обратную связь с пользователями путем изменения отклика на их действия. Тем самым пользователь предупреждается о том, что можно выполнить какое-то новое действие. При создании символа кнопки Flash автоматически открывает **временную** шкалу кнопки, состоящую из четырех кадров. Четыре кадра соответствуют четырем состояниям кнопки: Up (Отжата), Over (Поверх), Down (Нажата) и Hit (Активная зона). В этих кадрах может содержаться графика, **анимированные** видеоклипы или звуки. Несмотря на то что для различных состояний кнопки можно использовать разные графические изображения, очень важно сохранять единообразный внешний вид всех кнопок. Если в ответ на перемещение мыши графическое изображение кнопки будет слишком сильно изменяться, это может запутать пользователя и он не сможет понять назначение этой кнопки.

Совет

Подробная информация об использовании инструментов рисования для создания кнопок приведена в главе 3 "Рисование во Flash".

ПРОСТЫЕ кнопки

На простых кнопках для создания различных состояний (Up, Over и Down) применяется незначительное изменение графики, например цвета. При этом пользователь получает визуальную подсказку о том, что он может выполнить какое-то действие и управлять воспроизведением фильма. Чтобы создать простую кнопку, содержащую статическую **графику**, выполните следующие действия.

1. Выполните команду **Insert⇒New Symbol** (Вставка⇒Новый символ) или воспользуйтесь комбинацией клавиш <Cmd+F8> (Macintosh) либо <Ctrl+F8> (Windows). В результате откроется диалоговое окно Create New Symbol (Создать новый символ).
2. В поле Name (Имя) введите имя кнопки и установите переключатель Behavior (Поведение) в поле опции Button (Кнопка), как показано на рис. 9.1.

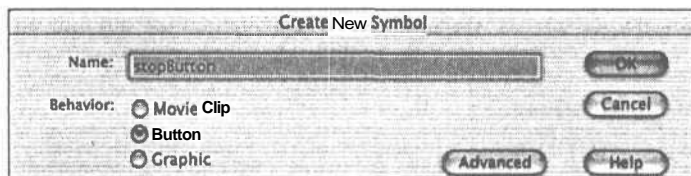


Рис. 9.1. Чтобы создать символ новой кнопки, выполните команду **Insert⇒New Symbol**

3. Щелкните на кнопке ОК. В результате будет создана временная шкала, в первый кадр которой будет помещен ключевой кадр, соответствующий состоянию Up. Это состояние соответствует внешнему виду кнопки в рабочем поле.
4. Воспользуйтесь инструментами рисования и создайте визуальную форму кнопки в кадре Up. Символы кнопки могут состоять из нескольких слоев, в которых можно разместить различные части графического изображения (рис. 9.2).

5. Состояние **Over** позволяет создавать **ролловеры**. Щелкните в кадре, расположенном под надписью **Over**. Если для создания эффекта ролловера необходимо изменить графику в нескольких слоях, **выделите** кадры в нескольких слоях. Для этого **щелкните** мышью, удерживая при этом клавишу <Shift>. В слои добавьте ключевые кадры, выполнив команду **Insert⇒Keyframe** (Вставка⇒Ключевой кадр) либо воспользовавшись комбинацией клавиш <Cmd+F6> (Macintosh) или <Ctrl+F6> (Windows). В рабочем поле появится графическое изображение из первого кадра.
6. Выделите графическое изображение кнопки в рабочем поле и внесите изменения, необходимые для создания состояния ролловера (рис. 9.3).

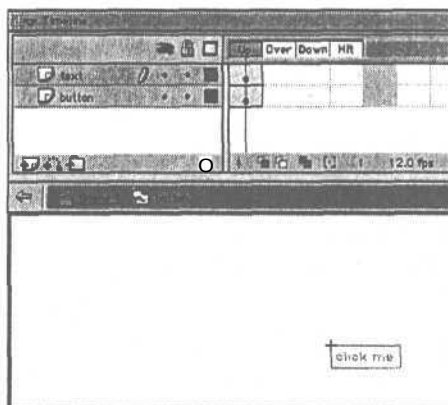


Рис. 9.2. Чтобы создать графическую форму кнопки в состоянии **Up**, воспользуйтесь любым инструментом рисования

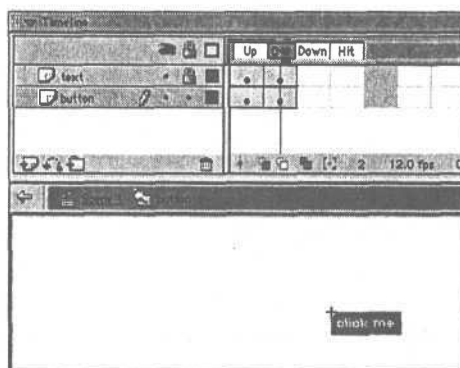


Рис. 9.3. В кадр **Over** вставьте ключевой кадр и внесите необходимые изменения для создания ролловера

7. Чтобы создать состояние нажатой кнопки, выделите кадр, расположенный под надписью **Down**, после чего повторите шаги 5 и 6. Постарайтесь сдвинуть графическое изображение, переместив его в кадре **Down** на несколько пикселей вниз и вправо. Это создаст иллюзию того, что кнопка нажата.
8. Состояние **Hit** определяет область кнопки, на которой производится щелчок мышью. Добавьте ключевой кадр под надписью **Hit** в слой, содержащий форму кнопки (рис. 9.4). Проверьте, чтобы форма охватывала всю область, фиксирующую щелчок на кнопке. Чтобы охватить всю область кнопки, может понадобиться нарисовать в кадре **Hit** отдельную форму.

Состояние **Hit**

Чтобы активную область можно было использовать, необходимо задать вполне определенное состояние **Hit**. Как правило, графическое изображение кнопки определяет область, которая будет **откликаться** на щелчок мышью, например прямоугольная область позади текста кнопки. Такое же графическое изображение можно использовать в кадре **Hit**.

Однако, если кнопка состоит исключительно из текста, очень важно, чтобы в кадре **Hit** был нарисован определенный контур, окружающий текст. В противном случае для получения отклика на щелчок пользователям придется щелкать точно на буквах.

9. Щелкните на кнопке **Back** (Назад), расположенной под пиктограммами папок, или выполните команду **Edit⇒Edit Document** (Правка⇒Редактировать документ). В результате вы выйдете из режима редактирования символов и вернетесь к основной временной шкале.

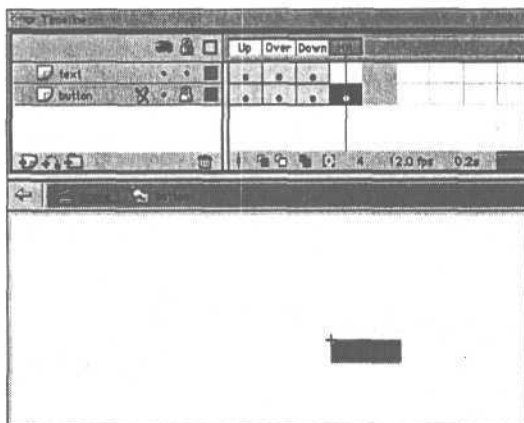


Рис. 9.4. Для задания активной области кнопки в кадр *Hit* вставьте ключевой кадр

10. Откройте панель Library (Библиотека), выполнив команду **Window⇒Library** (Окно⇒Библиотека) либо воспользовавшись комбинацией клавиш <Cmd+L> (Macintosh) или <Ctrl+L> (Windows).
11. Поместите кнопку в библиотеку и перетащите ее экземпляр в рабочее поле.
12. Для предварительного просмотра кнопки в рабочем поле выполните команду **Control⇒Enable Simple Buttons** (Управление⇒Включить простые кнопки). Если вы наведете курсор на кнопку, то увидите состояние *Over*, а если щелкнете на кнопке — состояние *Down*. Состояние *Hit* при просмотре является невидимым, но оно определяет область графического изображения кнопки, откликающегося на движение мыши и инициирующего запуск состояний *Over* и *Down*.

Итак, создание графических изображений кнопки завершено, но пока что кнопка не выполняет никаких функций. Чтобы кнопка не только изменяла свой внешний вид в ответ на перемещение мыши пользователем, но и выполняла какие-либо действия, к ней необходимо добавить сценарий **ActionScript**.

Чтобы для кнопки назначить какое-то простое действие, выполните следующие действия.

1. Выделите экземпляр кнопки в рабочем поле. Если просмотр простых кнопок разрешен, то для выделения кнопки необходимо воспользоваться инструментом **Arrow** (Стрелка).
2. Откройте панель **Actions** (Действия), выполнив команду **Window⇒Actions** (Окно⇒Действия) или нажав клавишу <F9>.
3. В левой части панели **Actions** дважды щелкните на папке **Actions**.
4. Дважды щелкните на папке **Movie Control** (Управление фильмом) и дважды щелкните на элементе **on**, в результате чего действие **on** будет присоединено к кнопке, а сценарий **on** будет отображен в правой части панели **Actions**.
5. Работая в обычном режиме, выберите событие, соответствующее действию **on** для данной кнопки (рис. 9.5). По умолчанию для кнопок назначено событие **release** (Отпустить).
6. Выделив строку кода действия **on**, дважды щелкните на элементе **goto** (перейти), расположенном в папке **Movie Control** в левой части панели **Actions**.

7. Установите переключатель в поле опции Go to and Play (Перейти и воспроизвести) или Go to and Stop (Перейти и остановить).
8. Из раскрывающегося меню Type (Тип) выберите пункт **Frame Number** (Номер кадра), Frame Label (Надпись кадра), Expression (Выражение), Next Frame (Следующий кадр) или Previous Frame (Предыдущий кадр) (рис. 9.6).
9. Протестируйте кнопку, выполнив команду **Control⇒Test Movie** (Управление⇒Тестировать фильм).

Совет

Подробная информация о написании сценариев приведена в главе 11 "Знакомство с ActionScript", а о событиях, назначаемых для кнопок, — в главе 16 "Взаимодействие, события и установление последовательности".

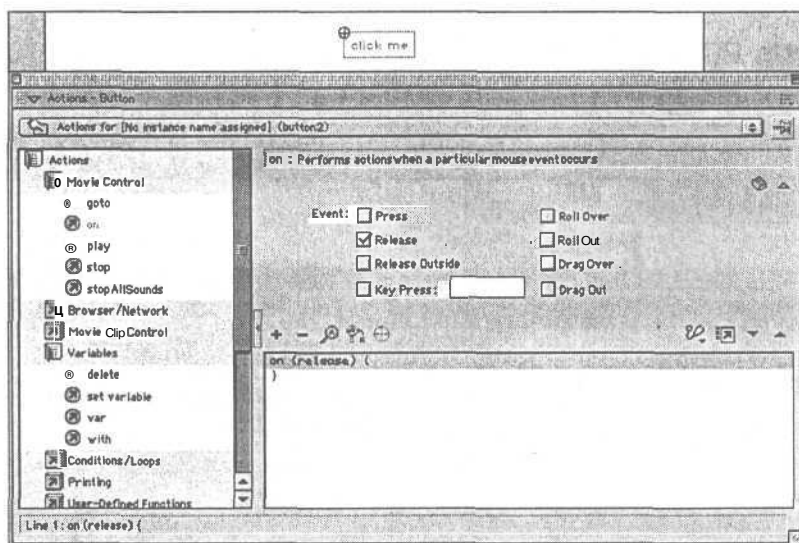


Рис. 9.5. Присоедините действие on к экземпляру кнопки

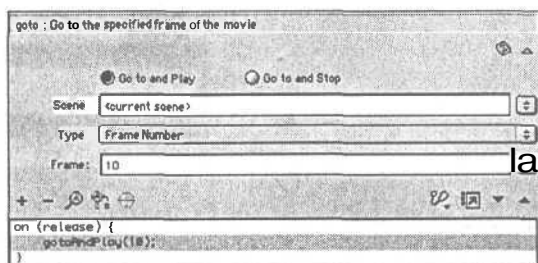


Рис. 9.6. Выберите необходимое действие из раскрывающегося меню Type

СЛОЖНЫЕ кнопки

Кнопки могут реагировать на действия пользователя огромным количеством способов. В сложных кнопках для обозначения их состояний используется не только графика, но и анимированные видеоклипы или звуки. Вместо или дополнительно к простому графическому изображению в любой кадр кнопки можно добавить звук. Как правило, звук добавляют к кадрам **Over** или **Down**. Для добавления звука к кнопке выполните следующие действия.

1. Вставьте в символ кнопки новый слой. Присвойте ему имя **sound** (звук). Лучше всего поместить звук в отдельный слой.
2. В слое звука вставьте ключевой кадр для состояния кнопки, к которому добавляется звук.
3. Выделив слой звука, перетащите экземпляр звука с панели **Library** в рабочее поле. Звук будет добавлен в кадр кнопки (рис. 9.7).
4. На панели инспектора свойств задайте синхронизацию звука с событием.

Совет

Об использовании звука рассказывается в главе 8 "Использование звука".

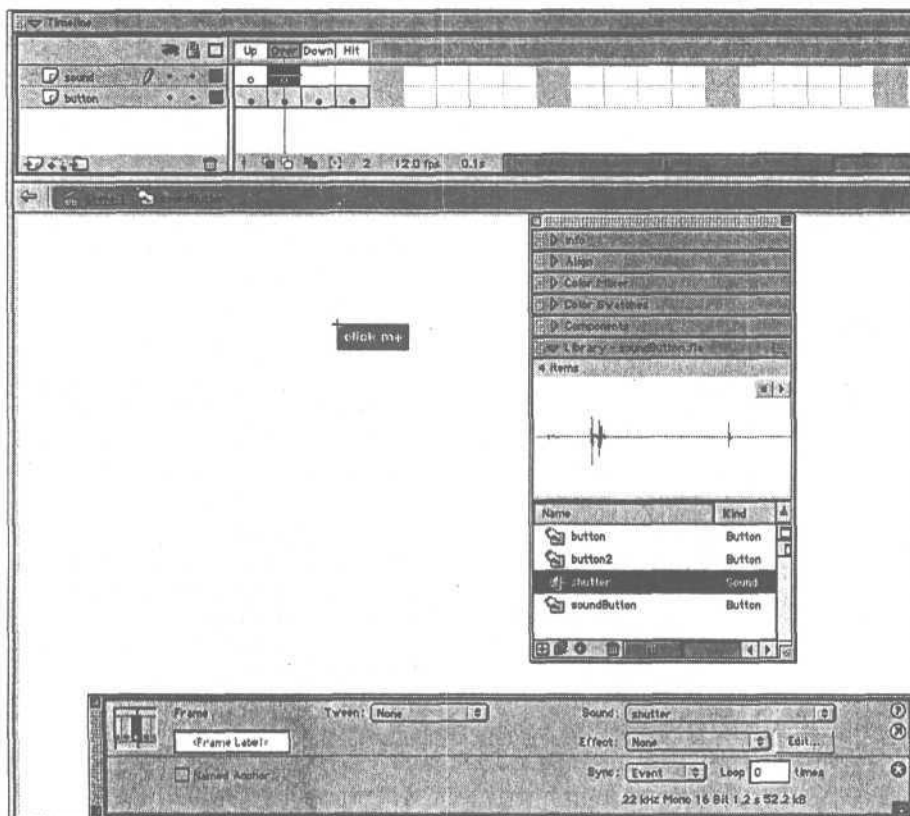


Рис. 9.7. Чтобы добавить звук к кнопке, вставьте ключевой кадр в кадр кнопки и перетащите звук с панели **Library** в рабочее поле

Кроме того, можно создать анимированные кнопки, поместив в кадры кнопки видеоклипы. Лучше всего задавать анимацию только для одного состояния кнопки и использовать с

этой целью короткие анимационные фрагменты. В противном случае анимация будет отвлекать внимание и затруднит процесс распознавания различных состояний кнопки. Чтобы добавить анимацию к состоянию кнопки, выполните следующие действия.

1. Создайте видеоклип состояния кнопки.

Совет

Подробная информация о создании анимированных видеоклипов содержится в главе 7 "Анимация во Flash".

2. Создайте кнопку и добавьте ключевой кадр в тот кадр кнопки, для которого создается анимированное состояние.
3. Перетащите экземпляр видеоклипа с панели Library в рабочее поле. Обязательно выравнивайте видеоклип относительно графики в других кадрах кнопки так, чтобы кнопка имела целостный вид. Чтобы отметить размещение кнопки в кадрах, воспользуйтесь направляющими. Вам совершенно не нужно, чтобы после щелчка или наведения указателя на кнопку она сместилась на несколько сотен пикселей.

Совет

О выравнивании и использовании направляющих рассказывается в главе 2 "Интерфейс Flash".

На заметку

Выравнивание элементов, в особенности текста, в пределах кадров кнопки является очень важным моментом. Если элементы не выровнены, то при переходе из одного состояния кнопки в другое будет казаться, что они "прыгают" по экрану.

4. Завершив работу над остальными кадрами кнопки, вернитесь к основной временной шкале и перетащите экземпляр кнопки в рабочее поле.
5. Протестируйте фильм, выполнив команду **Control**⇒**Test Movie**.

Внимание!

Внедренные видеоклипы нельзя предварительно просмотреть в редакторе Flash, в котором отображается только одна временная шкала. В основной временной шкале будет виден только первый кадр внедренного видеоклипа.

Можно также создать невидимые кнопки, которые состоят только из активной области. Зачем нужны невидимые кнопки? В версии MX таких причин существует гораздо меньше, чем в предыдущих версиях программы. Ранее невидимые кнопки использовались для создания активных областей графических изображений или видеоклипов. Как и при работе с картами изображений в HTML, можно не создавать отдельные кнопки, а поместить поверх графического изображения ряд использующихся для тех же целей невидимых кнопок. Однако улучшенная объектная модель версии MX позволяет интерактивно назначать кнопку непосредственно для видеоклипа. Кроме того, использовать невидимые кнопки для создания карт изображений очень неудобно. Средства считывания информации с экрана и другие вспомогательные технологии не распознают невидимые кнопки как собственно кнопки, поэтому любой текст под такой кнопкой будет проигнорирован.

Совет

Подробная информация об объекте Movie Clip (Видеоклип) будет приведена в главе 17 "Демонстрация мощи видеоклипов", а о доступности Flash — в приложении А.

МЕНЮ

С помощью меню пользователи перемещаются по фильму и решают, когда и в каком порядке отображать содержимое. Меню является основным элементом доступа к содержимому фильма.

Навигация является неотъемлемой частью работы с любым содержимым, размещенным в Web. Общих стандартов для навигации по Web-содержимому (например, перелистывания страниц или четкого разделения содержимого Web на нумерованные разделы, подобные страницам книги) не существует. Однако при работе во Flash очень важно обеспечить возможность навигации по фильму. Гибкость средств, которые предлагаются во Flash для создания новых интерфейсов со сложным вложением временных зависимостей, может запутать пользователей.

Если пользователи не смогут найти нужное содержимое, они никогда не услышат ваших сообщений и вообще вряд ли еще раз зайдут на ваш узел. Возможность вложения временных зависимостей, предлагаемая во Flash, означает, что существует несколько путей перемещения по фильму, и использующиеся в Web соглашения (например, отладочные операторы), которые детализируют путь пользователя к Web-странице, не являются стандартными. Содержимое Flash можно разделить на отдельные SWF-файлы, которые с целью сохранения пропускной способности загружаются по запросу. Например, разработчик узла, посвященного розничной продаже одежды, поступит очень мудро, если разнесет разные линейки продукции в отдельные фильмы, чтобы пользователи ожидали загрузки только информации, например, о женской или детской одежде, а не описание всего имеющегося товара. Необходимо составить четкую структуру фильмов и создать хорошую навигационную систему, которая позволяла бы пользователям легко маневрировать как в пределах одного SWF-файла, так и переходить от одного фильма к другому.

СТРУКТУРИРОВАНИЕ ДОКУМЕНТОВ с помощью ВРЕМЕННОЙ ШКАЛЫ

Самый простой способ систематизации взаимодействий в пределах фильма, практически не требующий написания сценариев, заключается в том, что видеоклипы, содержащие части содержимого, помещаются в ключевые кадры временной шкалы. Этим кадрам присваиваются соответствующие имена, а доступ к ним осуществляется посредством **ActionScript**.

Части содержимого фильма можно поместить в отдельные видеоклипы. Например, на Flash-узле компании можно создать видеоклипы *Home* (Начальная страница), *Services* (Услуги), *Portfolio* (Портфель), *About* (О компании) и др., соответствующие элементам меню. Затем участки узла, размещенные в отдельных слоях, можно собрать в одну временную шкалу, как показано на рис. 9.8.

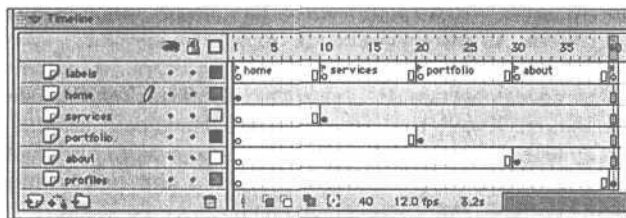


Рис. 9.8. Части фильма, расположенные в разных слоях, могут быть собраны в основной временной шкале

Каждая часть фильма должна быть размещена в отдельном слое, иметь ключевой кадр и метку кадра, т.е. для каждой части фильма необходимо создать слой. Очень важно расположить все разделы в кадрах с разными номерами. Тогда можно будет написать сценарий так, чтобы для доступа к определенной части содержимого воспроизводящая головка останавливалась в определенном кадре. Для этого ключевые кадры необходимо разнести так, чтобы каждый из них был

расположен в кадре с другим номером. Создайте новый слой для меток и добавьте ключевые кадры с метками, соответствующими каждому из ключевых кадров частей фильма. Можно просто назначить определенный сценарий для каждого из номеров кадров, но если на узел потребуется добавить новые части, вам придется изменять сценарии. С помощью меток можно легко добавить новое содержимое без необходимости обновления существующих сценариев.

Ключевые кадры содержимого можно разнести так, чтобы части фильма были помещены в последовательные ключевые кадры, но при этом вы не сможете прочитать на временной шкале соответствующие метки кадров. Для обеспечения возможности быстрой ссылки ключевые кадры лучше разместить с промежутками, чтобы можно было читать и использовать метки на временной шкале.

После создания навигационной системы к кнопкам можно присоединить действия, позволяющие получить доступ к различным частям узла путем перемещения воспроизводящей головки к разным меткам кадров частей фильма.

СВОЙСТВА TRACK AS BUTTON и TRACK AS MENU ITEM

При создании Flash-меню определяющими являются два фактора: размещение и тип данных меню. Сколько места необходимо для макетирования элементов меню? Будут ли меняться элементы меню? Если да, то как часто? Будут ли данные меню динамически загружаться из внешних источников?

Классическое Web-меню, т.е. просто ряд кнопок меню, можно использовать на небольших узлах, содержащих несколько элементов меню, не требующих частого обновления (рис. 9.9).



Рис. 9.9. Простое статическое меню представляет собой ряд кнопок

При создании экземпляра кнопки в качестве свойств можно выбрать пункт Track as Button (Отслеживать как кнопку) или Track as Menu Item (Отслеживать как элемент меню), как показано на рис. 9.10. Свойство Track as Button, заданное по умолчанию, определяет события, назначаемые для кнопки, независимо от других кнопок. Если пользователь щелкнет мышью на кнопке и перетащит курсор на другую кнопку, то инициирован будет только запуск первого события. Вторая кнопка является независимой от первой и не отслеживает действия мыши, примененные к другим кнопкам. Что касается свойства Track as Menu Item, то оно позволяет инициировать события кнопки посредством действий мыши, начатых на других кнопках. Если пользователь щелкнет мышью на одной кнопке, а затем перетащит курсор и отпустит его, находясь на другой кнопке, то это приведет к запуску ролловера и состояния щелчка второй кнопки. Для кнопок, являющихся элементами меню, необходимо задать свойство Track as Menu Item.

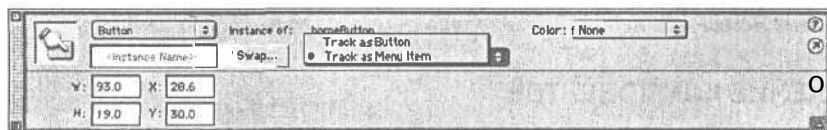


Рис. 9.10. Экземпляры кнопок можно отслеживать как кнопки или как элементы меню

Меню могут быть очень сложными, инициирующими выполнение сложных сценариев и даже позволяющими выполнять загрузку внешних и XML-данных. Динамические меню можно создавать по запросу.

Совет

Подробная информация об интеграции во Flash-фильмы XML-данных представлена в главе 28 "XML-данные".

ИСПОЛЬЗОВАНИЕ КОМПОНЕНТОВ

Компоненты — потрясающая новая функция Flash MX, берущая свое начало из элемента Smart Clip (Интеллектуальный клип) версии Flash 5. Компоненты представляют собой различные встроенные элементы интерфейса. В программе Flash MX содержится семь компонентов, представляющих собой наиболее распространенные элементы формы. Очень много компонентов можно загрузить с Web-узла компании Macromedia, а наибольший потенциал заложен в возможности самостоятельного создания пользовательских компонентов.

Совет

О создании пользовательских компонентов вы узнаете в главе 22 "Компоненты".

Компоненты размещаются на панели Components (Компоненты) (рис. 9.11). Чтобы открыть панель, выполните команду **Window⇒Components** (Окно⇒Компоненты) либо воспользуйтесь комбинацией клавиш <Cmd+F7> (Macintosh) или <Ctrl+F7> (Windows).

Во Flash MX содержится семь компонентов.

- **PushButton** (Нажимная кнопка). Простая кнопка с предварительно заданными состояниями.
- **CheckBox** (Флаговое поле). Традиционный элемент формы, позволяющий выбрать несколько пунктов из ряда предложенных.
- **RadioButton** (Переключатель). Другой традиционный элемент формы, позволяющий выбрать один или несколько пунктов из ряда предложенных.
- **ComboBox** (Комбинированный управляющий элемент). Прокручивающийся раскрывающийся список, позволяющий выбрать один элемент.
- **ListBox** (Окно списка). Прокручивающийся раскрывающийся список, позволяющий выбрать один или несколько элементов.
- **ScrollBar** (Полоса прокрутки). Вертикальный или горизонтальный элемент полосы прокрутки, который можно добавить к динамическим полям и полям ввода данных, что позволяет получить доступ к большим массивам текста в небольших текстовых полях.
- **ScrollPane** (Панель прокрутки). Прокручиваемая область окна, добавляющая вертикальную и горизонтальную полосы прокрутки для отображения видеоклипов. Компонент ScrollPane используется только с видеоклипами.

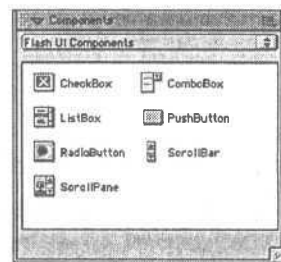


Рис. 9.11. Во Flash MX имеются встроенные элементы интерфейса, размещенные на панели Components

ДОБАВЛЕНИЕ КОМПОНЕНТОВ

Компоненты добавляются путем их перетаскивания в рабочее поле. Можно также дважды щелкнуть на компоненте на панели **Components**, в результате чего экземпляр компонента будет помещен в рабочее поле, где при необходимости его местоположение можно изменить.

Чтобы добавить компонент ComboBox, выполните следующие действия.

1. Щелкните на компоненте ComboBox на панели Components и перетащите его экземпляр в рабочее поле. На рис. 9.12 показано, что на панели инспектора свойств будут указаны параметры данного компонента.
2. Присвойте экземпляру компонента ComboBox имя month (месяц).
3. По умолчанию параметр Editable (Доступный для редактирования) установлен в значение `false` (ложь). Это означает, что элементы списка ComboBox являются статичными. Если данный параметр имеет значение `true` (истина), пользователи смогут выводить элементы списка и ввести термины, поиск которых будет осуществляться в пределах списка. Для параметра Editable оставьте значение `false`.
4. Дважды щелкните в столбце, который следует за Labels (Метки), в результате чего откроется диалоговое окно Values (Значения). Щелкните на кнопке со знаком "плюс" 12 раз, в результате чего будет создано 12 значений для 12 месяцев (рис. 9.13). Щелкните в каждой строке `defaultValue` (Значение по умолчанию) и введите имя элемента списка, т.е. названия месяцев с января (January) по декабрь (December).

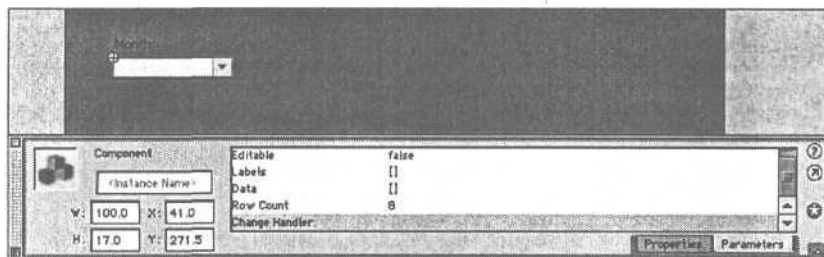


Рис. 9.12. Атрибуты компонента указываются на панели инспектора свойств

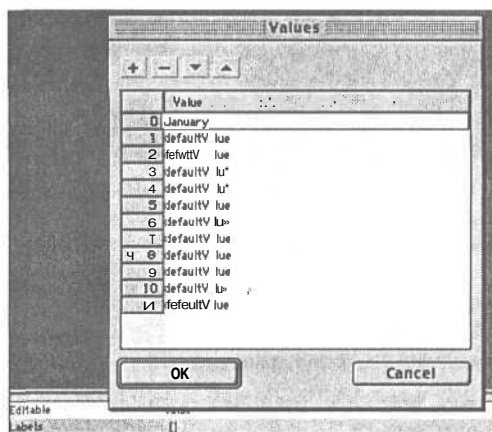


Рис. 9.13. В диалоговом окне Values введите элементы списка компонента ComboBox

5. Сохраните фильм и протестируйте его, выполнив команду **Control**⇒**Test Movie**. На рис. 9.14 показано, что значения, добавленные для компонента ComboBox, будут отображены в виде раскрывающегося списка.

РЕДАКТИРОВАНИЕ КОМПОНЕНТОВ

При добавлении компонента к фильму на панель Library добавляется папка Flash UI Components (Компоненты пользовательского интерфейса Flash), как показано на рис. 9.15. В этой папке содержатся символы, составляющие каждый компонент.

В папке Flash UI Components содержится выбранный компонент, папка Component Skins (Оболочки компонент) с графическими символами, составляющими выбранный компонент, папка Global Skins (Общие оболочки) с графическими элементами, общими для всех компонентов, и папка Core Assets (Основные средства), содержащая сложные элементы сценариев.

Внешний вид компонентов можно отредактировать, воспользовавшись папкой Component Skins. Откройте ее, а затем дважды щелкните на папке Skins компонента, который хотите отредактировать. Чтобы отредактировать видеоклип, дважды щелкните на его оболочке. Кроме того, можно удалить видеоклип и заменить его новым. После внесения необходимых правок вернуться к основной временной шкале и перетащите экземпляр отредактированного компонента в рабочее поле. Чтобы увидеть, как будет выглядеть новая оболочка, протестируйте фильм.

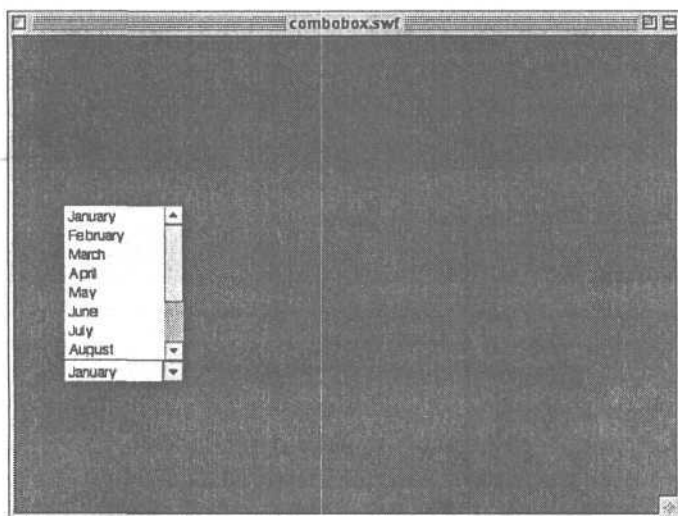


Рис. 9.14. Из компонента *ComboBox* в экспортированном фильме создан раскрывающийся список

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Можно ли редактировать стили текста компонента?

Да, доступ к цвету и стилю текстов компонента осуществляется посредством ActionScript. Подробно об использовании метода `setPropertyStyle` применительно к компонентам речь пойдет в главе 22 "Компоненты".

Если изменить размеры экземпляра компонента, он выглядит вытянутым. Как можно масштабировать компоненты?

При изменении размеров экземпляра компонента производится его масштабирование. При просмотре экземпляра в рабочем поле это может быть не заметно. Фактически компоненты не будут созданы до тех пор, пока фильм не экспортирован. После этого применяются параметры, в том числе и размеры. Протестировав фильм, вы увидите, что компонент масштабирован так, как вы указали.

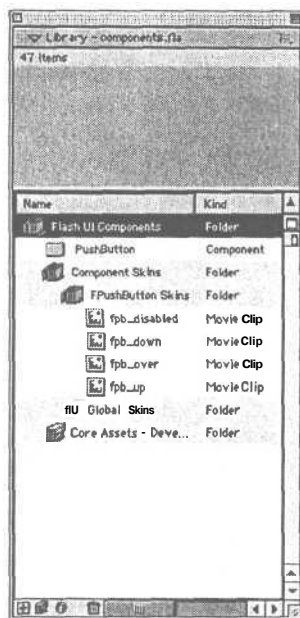


Рис. 9.15. При добавлении компонента к фильму на панель *Library* добавляется папка *Flash UIComponents*

FLASH ЗА РАБОТОЙ: КОМАНДА OPEN AS LIBRARY

Самый простой способ обеспечить коллективный доступ к элементам всем членам коллектива разработчиков заключается в создании проекта .fla, содержащего основные используемые в нем элементы. Обязательно включите в проект отредактированные компоненты, элементы дизайна и меню. Обновленный файл элементов можно раздать всем членам коллектива, которые могут открыть его как библиотеку, выполнив команду **File⇒Open as Library** (Файл⇒Открыть как библиотеку). После чего содержимое файла .fla откроется в виде библиотеки. Из открытой библиотеки разработчики могут перетащить элементы в свои рабочие файлы. Этот метод позволяет предоставить в распоряжение членов коллектива одни и те же элементы, что позволит избежать дублирования работы и может существенно повысить ее эффективность, снижая количество времени, затрачиваемого на разработку.

ИНТЕГРАЦИЯ ВИДЕО

В ЭТОЙ ГЛАВЕ...

Видео во Flash	187
Импортирование видео	188
Работа с импортированным видео	191
Управление воспроизведением видео	192
Возможные проблемы	195
Flash за работой: интерактивность видео	195

ВИДЕО во FLASH

В версии Flash MX впервые обеспечивается полноценная поддержка видео — возможность внедрять и воспроизводить видео посредством Flash Player без необходимости установки каких-либо внешних надстроек. Теперь видео можно импортировать во Flash и преобразовывать практически так же, как и любой другой объект. Во Flash-фильмы можно включать такое содержимое, как корпоративные объявления, обучающее видео, новости и спортивные репортажи.

Как и в случае с другими импортируемыми объектами, качество видео в опубликованном Flash-фильме зависит от качества импортируемого видео. Спланируйте, как вы будете использовать видео во Flash, и еще до импортирования отредактируйте его в предназначенной для этого программе. Для достижения наилучших результатов не пытайтесь редактировать видео во Flash. Размер файла импортированного видео в опубликованном фильме останется тем же самым, независимо от того, выполнялись ли какие-то операции по удалению кадров или содержимого. (Это похоже на масштабирование во Flash растровых изображений.) Как всегда, целью является достижение максимально возможного качества при наименьшем размере файла.

ИМПОРТИРОВАНИЕ ВИДЕО

Если на компьютере установлен модуль QuickTime 4 (QT) либо DirectX 7 или последующей версии (Windows), то Flash поддерживает импортируемые видеоформаты, перечисленные в табл. 10.1. Формат Macromedia Flash Video (.flv), созданный с помощью приложения Sorenson Spark, может быть импортирован во Flash без необходимости установки QT либо DirectX.

ТАБЛИЦА 10.1. ФОРМАТЫ ИМПОРТИРУЕМЫХ ВИДЕОФАЙЛОВ

Тип ФАЙЛА	ТРЕБУЕМЫЕ ДРАЙВЕРЫ	ПЛАТФОРМА
Audio Video Interleaved (. avi)	QuickTime 4, DirectX 7 или выше	Macintosh, Windows
Digital Video (. dv)	QuickTime 4	Macintosh, Windows
Motion Picture Experts Group (. mpg , . mpeg)	QuickTime 4, DirectX 7 или выше	Macintosh, Windows
QuickTime Movie (. mov)	QuickTime 4	Macintosh, Windows
Windows Media File (. wmv , . asf)	DirectX 7 или выше	Windows

Импортирование и экспортирование видео во Flash осуществляется с помощью кодека *Sorenson Spark*. Кодек расшифровывается как *кодек-декодек*, т.е. компрессор-декомпрессор данных, в данном случае видео. Сжатие видео реализуется посредством уменьшения числа битов, необходимого для представления видеоданных и последующей распаковки, т.е. развертывании исходных данных при воспроизведении. Видеофайлы печально известны необходимостью обработки большого количества данных, поэтому процедура сжатия для них является жизненно важной.

Видеоданные можно сжимать двумя способами: в пространственном и во временном отношении. При пространственном, или *внутрикадровом*, сжатии данные в каждом кадре сжимаются независимо от остальных кадров. При временном, или *межкадровом*, сжатии производится сравнение данных в последовательных кадрах и при этом сохраняются только различия между кадрами. В кодеке Sorenson Spark используются преимущества обоих типов сжатия. По возможности Sorenson Spark использует временное сжатие с целью создания файлов минимально возможных размеров. Однако, если видеокادر существенно изменяется, то с помощью пространственного сжатия создается ключевой видеокادر. Ключевые видеокadres отмечают точку изменения и являются аналогичными ключевым кадрам Flash. Ключевой видеокادر становится контрольной точкой последовательного сжатия с переменной битовой частотой. Ключевые видеокadres создаются при импортировании во Flash.

Sorenson Spark

Sorenson Spark — это мощная программа сжатия. Размеры видеофайлов огромны, и без применения интенсивного сжатия они будут слишком большими для передачи через Internet. Кодеки сжимают данные для ускорения загрузки и затем распаковывают их при воспроизведении файла. Кодек Spark интегрирован во Flash Player 6, поэтому для отображения видеосодержимого не нужно устанавливать никаких внешних плееров или надстроек.

Во Flash MX интегрирована стандартная версия Sorenson Spark, использующаяся для импортирования и сжатия видеофайлов. Имеется также профессиональная версия кодека с более надежными параметрами сжатия, в том числе двухпроходное сжатие с переменной битовой частотой. Приобрести кодек Sorenson Spark можно на узле <http://www.sorenson.com>.

ВНЕДРЕННОЕ и СВЯЗАННОЕ ВИДЕО

При работе во Flash MX видео можно либо непосредственно внедрить во Flash-фильм, либо связать его с Flash-фильмом. Внедренное видео становится частью документа Flash. При установлении связи с видео формата QuickTime Flash-фильм придется публиковать как фильм QuickTime. Затем видеосодержимое отображается с помощью надстройки QuickTime, вне модуля Flash Player. При внедрении видео интегрируется непосредственно во Flash-фильм, и для его просмотра не нужны никакие внешние плееры.

Совет

Подробная информация о параметрах публикации приведена в главе 30 "Оптимизация, публикация и экспортирование фильмов".

ПАРАМЕТРЫ СЖАТИЯ КОДЕКА SORENSON SPARK

Чтобы импортировать и внедрить видеоролик во Flash, выполните следующие действия.

1. Выполните команду **File⇒Import (Файл⇒Импорт)** либо воспользуйтесь комбинацией клавиш **<Cmd+R>** (Macintosh) или **<Ctrl+R>** (Windows). В результате откроется диалоговое окно **Import (Импорт)**.
2. Укажите местонахождение файла, подлежащего импортированию.
3. Выберите нужный файл и щелкните на кнопке **Open (Открыть)**. Если импортируемый файл имеет формат QuickTime (.mov), откроется диалоговое окно **Import Video (Импорт видео)**, изображенное на рис. 10.1. Установите переключатель в поле опции **Embed Video in Macromedia Flash Document (Внедрить видео в документ Macromedia Flash)**.
4. Откроется диалоговое окно **Import Video Settings (Настройки импортирования видео)** (рис. 10.2).

В верхней части диалогового окна **Import Video Settings** указаны свойства исходного видео, в том числе его размеры, объем файла, длительность и частота смены кадров. Ниже следуют параметры, которые в этом диалоговом окне можно откорректировать.

- Ползунок **Quality (Качество)** определяет качество, с которым будет экспортироваться видео. Он аналогичен параметрам экспортирования JPEG. Значение качества можно установить в пределах от 0 до 100, но минимальным приемлемым значением является 60. По умолчанию ползунок установлен в значение 50, но его обязательно нужно сместить хотя бы до 60.
- Ползунок **Keyframe interval (Интервал между ключевыми кадрами)** определяет, насколько часто кодек Sorenson Spark создает ключевые кадры. Помните, что полные данные кадра хранятся только там, где встречаются ключевые кадры. Чем больше ключевых кадров, тем больше будет размер файла. Если ползунок установлен в значение 0, вставка ключевых кадров не производится. Чем в более низкое значение установлен ползунок, тем чаще будут вставлены ключевые кадры (за исключением

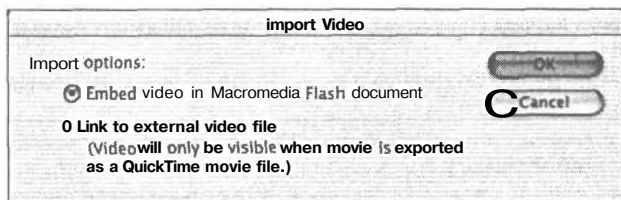


Рис. 10.1. При импортировании фильма QuickTime вам предложат либо внедрить видео, либо установить с ним связь

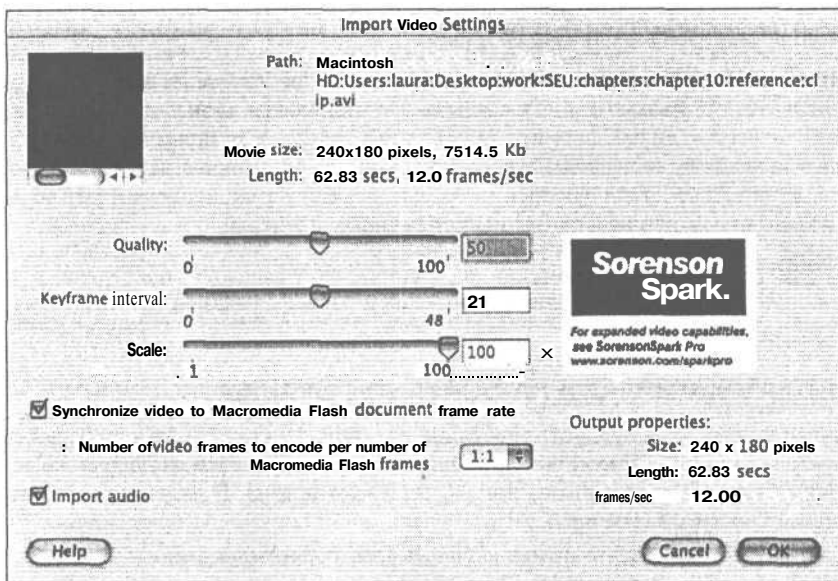


Рис. 10.2. С помощью кодека Sorenson Spark при импортировании можно указать параметры сжатия видео

значения 0). Поэкспериментируйте и найдите компромиссный вариант между качеством изображения и размером файла.

- Ползунок Scale (Масштаб) позволяет уменьшить пиксельный размер изображения импортированного видео. При перемещении ползунка автоматически обновляются параметры Output properties (Выходные свойства), отображаемые в правом нижнем углу диалогового окна Import Video Settings. По умолчанию ползунок установлен в значение 100%.
- Опция Synchronize Video to Macromedia Flash Document (Синхронизировать видео с документом Macromedia Flash), в поле которой автоматически установлен флажок, корректирует частоту смены кадров исходного видео так, чтобы она совпадала с заданной частотой смены кадров Flash-фильма. Если флажок с поля данной опции снят, то каждый кадр импортированного видео занимает один кадр во временной шкале Flash-фильма.
- Задайте нужное значение в поле опции Number of Video Frames to Encode Per Number of Macromedia Flash Frames (Число видеок кадров, подлежащих кодированию, к числу кадров Macromedia Flash). Этот параметр задает отношение числа кадров видео к числу кадров временной шкалы Flash. По умолчанию выбрано значение 1:1, при котором сохраняется исходная частота смены кадров импортированного видео. Постарайтесь не опускаться ниже частоты смены кадров, равной 12 кадрам в секунду, иначе видео будет казаться прерывистым.
- Поле опции Import Audio (Импортирование аудио) позволяет включать или исключать звуковые дорожки в импортируемом видео. По умолчанию флажок в этом поле установлен.

5. Щелкните на кнопке OK. Откроется диалоговое окно Import со строкой состояния.

Внимание!

При работе с видео задействуется большое количество памяти, особенно на платформе Macintosh. В операционных системах версий, предшествующих OS X, высвободите для Flash как можно больше оперативной памяти. В противном случае вы не сможете импортировать видео.

6. Если число кадров временной шкалы не достаточно для отображения импортированного видео, на экране появится подсказка с предложением расширить временную шкалу (рис. 10.3). Чтобы расширить временную шкалу, щелкните на кнопке Yes (Да), а для сохранения текущего количества кадров щелкните на кнопке No. Кадры импортированного видео с номером, превышающим число кадров временной шкалы, при воспроизведении видео отображены не будут.

На заметку

Число кадров, требующееся для отображения видео, равнодлительности видео в секундах, умноженной на частоту смены кадров Flash-фильма.

Видео импортируется в библиотеку, а его экземпляр помещается в рабочее поле. Чтобы импортировать видео непосредственно в библиотеку, выполните команду **File⇒Import to Library** (Файл⇒Импортировать в библиотеку) и повторите шаги 2-5 приведенной выше инструкции. При импортировании непосредственно в библиотеку подсказка о том, что во временной шкале содержится меньше кадров, чем в импортированном видео, на экран выводиться не будет.

Чтобы проверить размер импортированного видео, сохраните фильм и протестируйте его, выполнив команду **Control⇒Test Movie**. Для предварительного просмотра фильма выполните команду **View⇒Show Bandwidth Profiler** (Вид⇒Показать профайлер полосы пропускания). Чтобы узнать, как долго пользователи будут ожидать выполнения загрузки содержимого, выполните команду **View⇒Show Streaming** (Вид⇒Показать поток).

Совет

Подробная информация о тестировании и оптимизации фильма содержится в главе 30 "Оптимизация, публикация и экспортирование фильмов".

РАБОТА С ИМПОРТИРОВАННЫМ ВИДЕО

Экземпляры внедренного видео можно преобразовать так же, как и другие объекты. Его можно вращать, наклонять, применять цветовые эффекты, маскировать и даже создать промежуточные отображения. Для применения преобразований выделите экземпляр видео и воспользуйтесь инструментом Free Transform (Свободное преобразование). Для маскирования видео переместите слой, содержащий его, под слой маски. Для анимации или применения цветовых эффектов к экземпляру видео внедрите видео в символ видеоклипа и примените цветовые эффекты или создайте промежуточное отображение символа. Комбинируйте анимацию и видео очень осторожно, поскольку размер файла может очень сильно увеличиться.

Совет

Подробная информация о преобразовании объектов приведена в главе 2 "Интерфейс Flash", о создании масок — в главе 3 "Рисование во Flash", а о создании промежуточных отображений и цветовых эффектах — в главе 7 "Анимация во Flash".

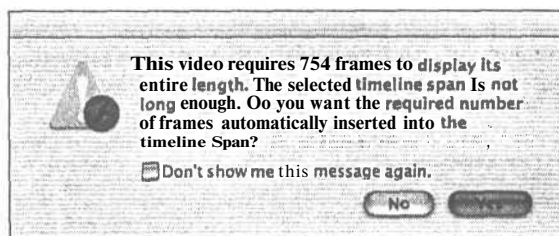


Рис. 10.3. Если число кадров временной шкалы не достаточно для отображения импортированного видео, на экране появится предложение расширить временную шкалу

После импортирования видео его можно заменить или обновить, как и другие символы. Чтобы заменить внедренный видеоклип, выполните следующие действия.

1. На панели Library (Библиотека) выделите внедренный видеоклип.
2. Щелкните в правом верхнем углу панели, в результате чего откроется меню Library Options (Параметры библиотеки). Выберите из него пункт Properties (Свойства). Откроется диалоговое окно Embedded Video Properties (Свойства внедренного видео), изображенное на рис. 10.4.
3. Щелкните на кнопке Import и в открывшемся диалоговом окне найдите видеофайл, на который будет заменен внедренный видеоклип. Затем выполните действия, описанные выше, в разделе "Параметры сжатия кодека Sorenson Spark".

Чтобы обновить клип, который после импортирования во Flash был отредактирован, выделите клип на панели Library и откройте диалоговое окно Embedded Video Properties в соответствии с приведенной выше инструкцией. Вместо кнопки Import щелкните на кнопке Update (Обновить).

Видеоклипы, помещенные в рабочее поле, можно перемещать подобно любым другим объектам. На рис. 10.5 показано, что на панели инспектора свойств экземпляра клипа можно присвоить имя и изменить его координаты и размеры.

УПРАВЛЕНИЕ ВОСПРОИЗВЕДЕНИЕМ ВИДЕО

После внедрения видео им можно управлять практически так же, как и любым другим содержимым, помещенным в рабочее поле. Кадры видео преобразовываются в кадры Flash, и для предварительного просмотра содержимого видеофайла можно перетащить воспроизводящую головку по временной шкале. Если содержимое видеофайла слишком велико (например, видео является презентацией), имеет смысл добавить кнопки, которые позволят пользователям управлять воспроизведением видео. Управление воспроизведением видео осуществляется с помощью ActionScript. При этом осуществляется управление воспроизведением временной шкалы, в которую было помещено данное видео. Чтобы перемотать видео в начало, нужно просто вернуться к кадру 1 на панели Timeline (Временная шкала). Чтобы выполнить быструю перемотку вперед или назад, задайте положительное или отрицательное приращение количества кадров.

Для создания простых кнопок управления видеоклипом выполните следующие действия.

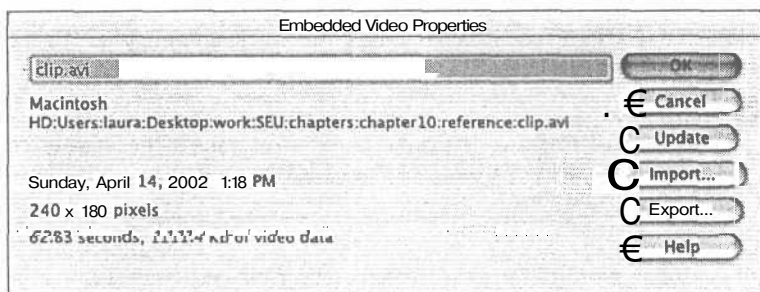


Рис. 10.4. В диалоговом окне Embedded Video Properties можно заменить или обновить видеоклип



Рис. 10.5. На панели инспектора свойств указаны свойства видеоклипа и содержится кнопка для замены экземпляра символа

1. Откройте документ Flash и присвойте существующему слою имя Video.
2. Импортируйте видеоклип, выполнив команду **File⇒Import**. На прилагаемом к данной книге компакт-диске имеется файл **video.avi** с образцом видео. Выделите видеоклип и импортируйте его в рабочее поле. Обязательно проверьте, чтобы экземпляр фильма был помещен в основную временную шкалу.
3. В документ добавьте новый слой и присвойте ему имя **Controls** (Элементы управления).
4. Выполните команду **Window⇒Common Libraries⇒Buttons** (Окно⇒Общие библиотеки⇒Кнопки), как показано на рис. 10.6. Откроется библиотека, содержащая образцы кнопок различных графических стилей. Просмотрите кнопки и выберите стиль, который лучше всего подходит для вашего фильма.



Рис. 10.6, Просмотрите образцы кнопок, выполнив команду **Window⇒Common Libraries⇒Buttons**

5. Выделив на основной временной шкале слой Controls, перетащите кнопки Play (Воспроизведение), Stop (Остановка), Fast Forward (Быстрая перемотка вперед) и Re-wind (Перемотка назад) из библиотеки Buttons в рабочее поле (рис. 10.7).
6. Выделите кнопку Play (с одной стрелкой, указывающей вправо) и выполните команду **Window⇒Actions** (Окно⇒Действия) либо нажмите клавишу <F9>, чтобы открыть диалоговое окно ActionScript Editor (Редактор ActionScript).

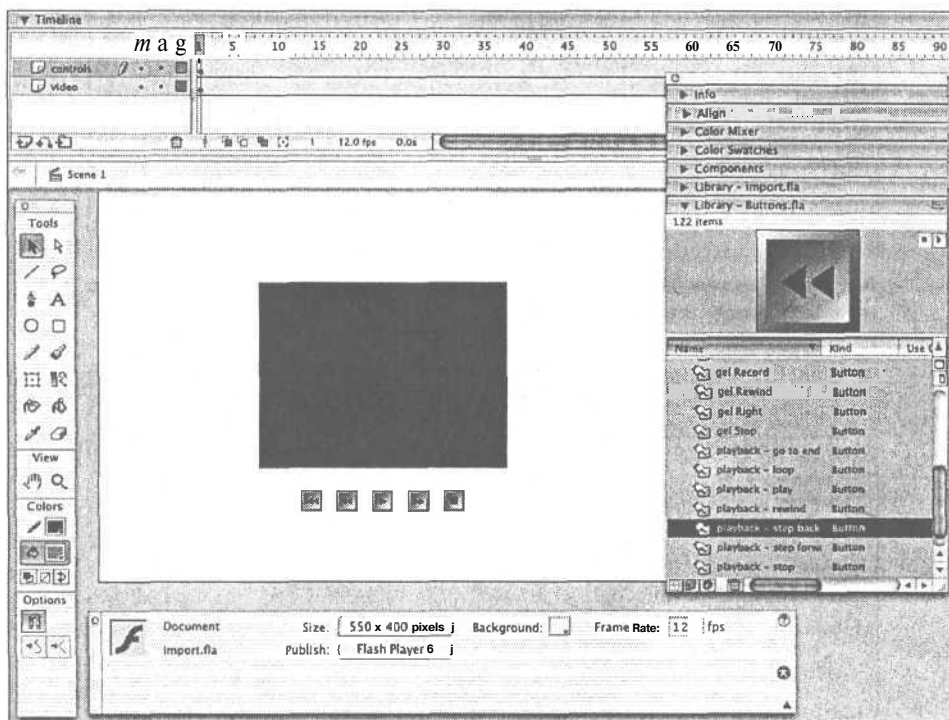


Рис. 10.7. Поместите кнопки управления воспроизведением в слой Controls

7. Откройте меню параметров панели Actions (щелкнув на кнопке в правом верхнем углу) и проверьте, чтобы была выбрана опция **Expert Mode** (Экспертный режим).
 8. В окне сценария введите код (рис. 10.8):


```
on (release) {
    play O;
```
 9. Выделите в рабочем поле кнопку Stop (с изображением квадрата) и снова запустите редактор **ActionScript**, щелкнув на панели Actions или нажав клавишу <F9>.
 10. Введите следующий код (рис. 10.9):


```
on (release) {
    stop();
```
 11. Сохраните и протестируйте свой фильм, нажав комбинации клавиш <Cmd+Return> (Macintosh) или <Ctrl+Enter> (Windows).
 12. Flash-фильмы запускаются автоматически, т.е. воспроизведение видео должно начаться сразу же после его загрузки, без щелчка на кнопке Play. (Чтобы запретить автоматическое воспроизведения фильма, присоедините к кадру 1 действие **stop()** ;.) Протестируйте кнопки. Щелкните на кнопке Stop, чтобы остановить воспроизведение видео. После щелчка на кнопке Play видео начнет воспроизводиться с точки, в которой было остановлено.
- Чтобы завершить работу над элементами управления воспроизведением, выполните следующее.

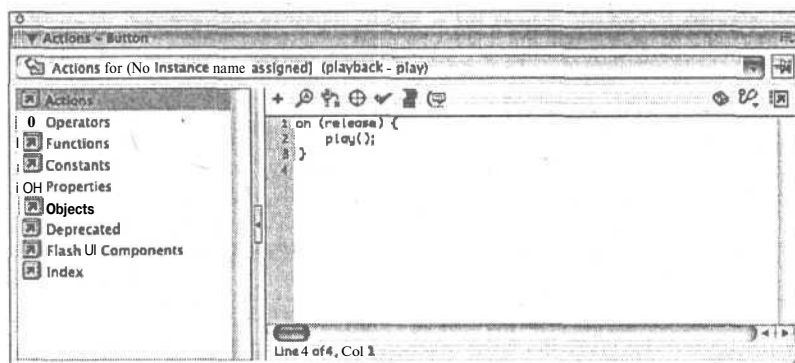


Рис. 10.8. Простое действие воспроизведения, добавляемое к кнопке, позволит управлять воспроизведением видеоклипа

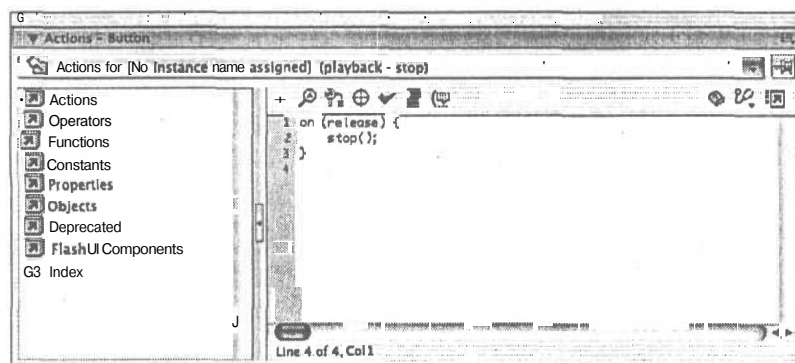


Рис. 10.9. Добавьте действие остановки воспроизведения к кнопке Stop

13. Вернитесь к сохраненному фильму в режиме редактирования. Щелкните на кнопке **Rewind-to-the-Beginning** (Перемотать в начало) (с двумя треугольниками, указывающими влево на вертикальную линию) и в окне редактора **ActionScript** введите следующий код:

```
on (release) {  
    gotoAndStop(1);  
}
```
14. Вернитесь в рабочее поле и выделите кнопку **Rewind** (Перемотка назад) (с двумя стрелками, указывающими влево) и в окне редактора **ActionScript** введите код

```
on (release) {  
    prevFrame();  
}
```
15. Вернитесь в рабочее поле и выделите кнопку **Fast Forward** (Быстрая перемотка вперед) (с двумя стрелками, указывающими вправо) и в окне редактора **ActionScript** введите следующий код:

```
on (release) {  
    nextFrame();  
}
```
16. Сохраните и протестируйте фильм. Щелкните на каждой кнопке, чтобы изменить воспроизведение фильма.

Совет

Подробная информация о кнопках приведена в главе 9 "Кнопки, меню и поля ввода данных", а о написании сценариев — в главе 11 "Знакомство с ActionScript".

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Может ли видео иметь свои собственные звуковые свойства?

К сожалению, нет. Видео не поддерживает событийные звуки, и в каждом SWF-файле имеется только одна дорожка потокового аудио.

При попытке экспортирования в QuickTime на экран выводится следующее сообщение об ошибке: *The installed version of QuickTime does not have a handler for this type of Macromedia Flash movie* (Установленная версия QuickTime не имеет обработчика для данного типа Flash-фильма).

Это сообщение означает, что установленная версия QuickTime не поддерживает Flash Player 6. До выхода QuickTime 6 содержимое Flash Player 6 не поддерживалось. Для устранения данной проблемы перейдите к настройкам публикации. Для этого выполните команду **File⇒Publish Settings** (Файл⇒Параметры публикации). В открывшемся диалоговом окне перейдите на вкладку **Flash** и из раскрывающегося меню **Version** (Версия) выберите **Flash Player 5**.

FLASH ЗА РАБОТОЙ: ИНТЕРАКТИВНОСТЬ ВИДЕО

Возможность внедрять видео во **Flash MX** означает, что пользователи могут управлять передачей содержимого видео, причем не только с помощью элементов управления воспроизведением. Например, выполняя тщательное редактирование и сжатие с целью оптимизации размера файла, можно создать комментарии к нескольким видеоклипам, работающие под управлением пользователя. Кроме того, для создания вспомогательных субтитров на разных языках можно задействовать Flash-текст. Комбинируя возможности видео и Flash, можно добиться замечательных эффектов.

РАСШИРЕНИЕ ИНТЕРАКТИВНОСТИ с ПОМОЩЬЮ ACTIONSCRIPT

В ЭТОЙ ЧАСТИ...

Глава 11. Знакомство с ActionScript

Глава 12. Управление переменными, данными и типами данных

Глава 13. Использование операторов

Глава 14. Работа с данными: использование предложений

Глава 15. Объединение предложений в функции

Глава 16. Взаимодействие, события и установление последовательности

Глава 17. Демонстрация моши видеоклипов

Глава 18. Процедура рисования с помощью ActionScript

ЗНАКОМСТВО с ACTIONSCRIPT

В ЭТОЙ ГЛАВЕ...

Добавление интерактивности с помощью ActionScript	199
Использование панели Actions	200
Знакомство с объектно-ориентированными языками	204
Три кита: удобочитаемость, повторное использование и достижимость	206
Возможные проблемы	213
Flash за работой: удобочитаемый, многократно используемый, достижимый код	214

ДОБАВЛЕНИЕ ИНТЕРАКТИВНОСТИ с помощью ACTIONSCRIPT

ActionScript — это язык программирования, посредством которого во Flash осуществляется отправка команд и запросов о временных зависимостях, видеоклипах, кнопках и других объектах.

На заметку

Язык ActionScript создан на основе стандарта ECMA-262, который, в свою очередь, составлен на базе языка JavaScript. По сути, ActionScript является разновидностью JavaScript, адаптированного и оптимизированного для работы в среде Flash.

Очень часто с помощью ActionScript можно легко достичь целей, которые в противном случае представляли бы собой труднодостижимую или, вообще, невозможную задачу. Без ActionScript можно реализовать только малую часть возможностей Flash. Например, ActionScript необходим для реализации любого рода интерактивности, например отклика после щелчка пользователем кнопкой мыши или нажатия клавиши на клавиатуре. Кроме того, ActionScript является единственным способом реализации перехода к определенному кадру временной шкалы либо начала или остановки воспроизведения видеоклипа. Однако эти про-

стые примеры не раскрывают богатство, гибкость и бесконечность возможностей, которые ActionScript открывает перед разработчиками, использующими среду Flash. Язык ActionScript для Flash является "родным", и без хотя бы минимального овладения им вы будете подобны человеку, который пребывает в чужой стране и вынужден общаться с помощью жестикологии. В этом случае придется довольствоваться меньшим по сравнению с тем, что в действительности вы хотели бы получить.

Как правило, при использовании ActionScript удастся получить SWF-файлы меньших размеров и с лучшим качеством, чем при создании промежуточных отображений. Еще одним преимуществом является то, что различные задачи удастся выполнять с большей точностью, например перемещать видеоклип в точно заданное место рабочей области.

Возможности языка ActionScript безграничны, а использовать его очень легко. Знакомство с ActionScript лучше всего начать с панели Actions (Действия).

ДОСТУП К ПАНЕЛИ ACTIONS

Чтобы открыть или развернуть панель Actions, выполните одно из следующих действий:

- из меню Window (Окно) выберите пункт Actions (Действия);
- нажмите клавишу <F9>.

После этого на экране появится или будет развернута панель Actions (рис. 11.1). Если эта панель была открыта, но скрыта другим окном, она переместится на передний план. При заданной по умолчанию компоновке панель Actions связана с панелью инспектора свойств. (Если вы хотите восстановить компоновку, заданную по умолчанию, выполните команду Window⇒Panel Sets⇒Default Layout (Окно⇒Наборы панелей⇒Размещение по умолчанию).)

Панель Actions является встроенным во Flash редактором ActionScript. При соответствующем использовании суффиксов экземпляров объектов (например, суффикс _mc для видеоклипов) в ней будет открываться вспомогательное окно с советами по применению кода. Воспользовавшись кнопкой Pin current script (Прикрепить текущий сценарий), вы сможете "закрепить" текущий сценарий, т.е. сохранить его на панели Actions.

ИСПОЛЬЗОВАНИЕ ПАНЕЛИ ACTIONS

Весь код на языке ActionScript вводится на панели Actions. Щелкните в кадре, на кнопке или видеоклипе, к которому требуется присоединить код ActionScript, а затем перейдите к панели Actions и в ее правой части введите код соответствующего действия.

ВЫБОР ДЕЙСТВИЙ для КАДРА или ОБЪЕКТА

Заданный блок действий можно присоединить либо к объекту (кнопке или видеоклипу), либо к кадру. Тип вводимого действия указывается в строке заголовка серого цвета, распо-

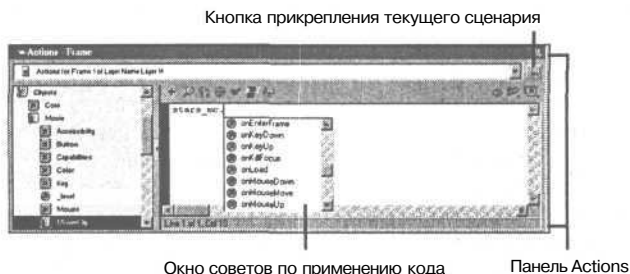


Рис. 11.1. Панель Actions является встроенным во Flash редактором ActionScript

женной в верхней части панели Actions. Это может быть Actions — Frame (Действия, присоединяемые к кадру), Actions — Movie Clip (Действия, присоединяемые к видеоклипу) или Actions — Button (Действия, присоединяемые к кнопке). Соответствующий режим выбирается щелчком на объекте или кадре.

Действия, присоединенные к кадру, выполняются при достижении воспроизводящей головкой этого кадра. Действия, присоединенные к объекту, выполняются при наступлении события, назначенного для данного объекта.

- **Действие, присоединенное к кадру.** Предположим, что к кадру присоединено действие `trace`, а именно

```
trace ("hello");
```

- Всякий раз при достижении воспроизводящей головкой этого кадра в выходном окне будет отображаться слово `hello`.
- **Действие, присоединенное к объекту.** Видеоклипы обладают событием "вступительного кадра", которое наступает всякий раз по достижении кадра. Если фильм воспроизводится при частоте 12 кадров в секунду, это событие наступает 12 раз в секунду. Чтобы сообщить Flash, какое событие должно запускать тот или иной код, этот код помещают внутрь обработчика события. Традиционный обработчик события "вступительный кадр" выглядит следующим образом:

```
onClipEvent (enterFrame) {  
    // сюда вводится код!  
}
```

Так работает методика в целом. Ниже приведены два существенных исключения.

- **Функция** — это именованный блок кода, внедряющий определенную процедуру. Например, `gotoAndPlay()` — это встроенная функция видеоклипа, вызывающая переход фильма к определенному кадру на временной шкале и начало его воспроизведения.
- Везде, где можно писать код, будь то кадр или обработчик события, можно задавать свои собственные пользовательские функции. Когда воспроизводящая головка достигает кадра или выполняется обработчик события, содержащий определение пользовательской функции, функция становится доступной, но фактически не выполняется. Чтобы инициализировать это действие, к функции *следует обратиться*.

Совет

Подробная информация о функциях приводится в главе 15 "Объединение предложений в функции".



Во Flash MX имеется новый формат обработчиков событий, позволяющий вводить их код в кадр. Например, новый обработчик события "вступительный кадр" выглядит следующим образом:

```
myClip.onEnterFrame = function () {  
    // сюда вводится код!  
};
```

Когда воспроизводящая головка достигает кадра с кодом обработчика, Flash начинает "слушать" событие. Однако "срабатывает" обработчик (выполняет содержащийся в нем код) только при наступлении события.

Совет

В приведенном выше примере `myClip` — это имя экземпляра видеоклипа. Имена экземпляров видеоклипов были рассмотрены в главе 6 "Символы, экземпляры и элементы библиотек".

ОБЫЧНЫЙ И ЭКСПЕРТНЫЙ РЕЖИМЫ

Создавать и редактировать код можно в двух режимах: обычном и экспертном.

За исключением многострочных комментариев (с которыми можно работать только в экспертном режиме), в любом из режимов можно писать одни и те же выражения. Обычный режим больше подходит для коротких простых блоков кода. Экспертный режим хорош для кода, имеющего более сложную структуру.

Flash помогает писать код



Описание инструкций по созданию кода в обычном режиме может занять достаточно много места. Поэтому, чтобы пояснения оставались лаконичными, в данной книге описывается экспертный режим. Однако, пытаясь запомнить формат действия, не забывайте о том, что вы всегда можете перейти в обычный режим и дать возможность Flash "потренировать" вас.

Во Flash имеется новая функция, *Code Hints* (Советы по использованию кода), которая существенно облегчает работу в экспертном режиме. Функция *Code Hints* работает в двух формах: *всплывающие подсказки* и *завершение* кода в режиме меню.

В режиме всплывающих подсказок после ввода части кода будут указаны форматы.

Режим завершения кода позволяет уменьшить ввод с клавиатуры путем отображения меню возможных вариантов кода (рис. 11.1).

Во Flash имеется новая функция, *Code Hints* (Советы по использованию кода), которая существенно облегчает работу в экспертном режиме. Функция *Code Hints* работает в двух формах: *всплывающие подсказки* и *завершение кода* в режиме меню. В режиме всплывающих подсказок после ввода части кода будут указаны форматы. Режим завершения кода позволяет уменьшить ввод с клавиатуры путем отображения меню возможных вариантов кода (рис. 11.1).

Для доступа к меню советов по использованию кода для экземпляров различных классов объектов (таких как видеоклипы, кнопки или текстовые поля) к каждому имени экземпляра необходимо добавить суффикс нужного класса. Например, на рис. 11.1 показано, что к имени экземпляра `stars` добавляется суффикс `_mc`, в результате чего образовывается имя `stars_mc`. (Все возможные суффиксы перечислены в табл. 11.1)

Обе формы советов по использованию кода по умолчанию являются включенными. Чтобы их отключить, выполните команду *Edit⇒Preferences* (Правка⇒Настройка). Затем в диалоговом окне *Preferences* (Настройка) перейдите к вкладке *ActionScript Editor* (Редактор ActionScript) и снимите флажок с поля опции *Code Hints* (Советы по использованию кода). Если функция советов по использованию кода включена, то можно отдельно включать и отключать режим всплывающих подсказок. Для этого в диалоговом окне *Preferences* перейдите к вкладке *General* (Общее) и либо установите, либо снимите флажок с поля опции *Show Tooltips* (Отображать всплывающие подсказки).

ТАБЛИЦА 11.1. СУФФИКСЫ CODE HINTS ПАНЕЛИ ACTIONS

КЛАСС ОБЪЕКТА	СУФФИКС
Массив	<code>_array</code>
Кнопка	<code>_btn</code>
Камера	<code>_camera (*)</code>
Цвет	<code>_color</code>

КЛАСС ОБЪЕКТА	СУФФИКС
Дата	<code>_date</code>
Микрофон	<code>_mic (*)</code>
Видеоклип	<code>_mc</code>
Сетевое соединение	<code>_connection (*)</code>
Сетевой поток	<code>_stream (*)</code>
Объект коллективного использования	<code>_so (*)</code>
Звук	<code>_sound (</code>
Строка	<code>_str</code>
Текстовое поле	<code>_txt</code>
Текстовый формат	<code>_fmt</code>
Видео	<code>_video (*)</code>
XML	<code>_xml</code>
Сокет XML	<code>_xmlsocket</code>

(*) Несмотря на то что данные суффиксы указаны в документации Macromedia, в настоящее время их ввод не приводит к открытию подсказок *Code Hints*.

При работе в обычном режиме панель Actions принимает вид "заполненной анкеты". После щелчка на элементе в левой части панели Actions в правой верхней части панели появляется его описание. После двойного щелчка на элементе в правой нижней части панели появится та часть действия, которая всегда остается неизменной. Это позволяет вводить с клавиатуры меньшее количество кода и дает возможность познакомиться с синтаксисом действия.

В текстовые поля в верхней части панели вводится та часть кода действия, которая не всегда остается неизменной. Например, при использовании действия `gotoAndPlay()` воспроизводящая головка переходит к определенному кадру видеоклипа, и с него начинается воспроизведение. После двойного щелчка на элементе `gotoAndPlay()` на панели Actions появится следующая строка:

```
<not set yet>. gotoAndPlay();
```

Фрагмент `<not set yet>` является заполнителем имени видеоклипа, которое необходимо указать в текстовом поле сверху панели Actions. В другом текстовом поле необходимо указать номер кадра. В целом код в окне действий будет выглядеть следующим образом:

```
myClip.gotoAndPlay(4);
```

Непосредственно в окно действий не вводится никакой код. Оно используется только для просмотра. Кроме того, в обычном режиме можно добавлять, удалять выражения или изменять их порядок. Пользуйтесь обычным режимом, когда вы не уверены в синтаксисе используемых выражений и хотите освежить свою память или избежать ошибок.

В экспертном режиме можно выделить элемент действия (дважды щелкнув на нем в левой части панели), но сверху окна действий не будет текстовых полей. Код можно непосредственно вводить в окно действий или копировать и вставлять его из другого документа (в обычном режиме этих возможностей нет). Как правило, при работе в экспертном режиме, в отличие от обычного, тот же самый код можно получить путем меньшего количества нажатий клавиши или щелчков мышью.

Чтобы выбрать обычный или экспертный режим, выполните одно из следующих действий.

- Щелкните в меню параметров, расположенном в правом верхнем углу панели Actions, и выберите пункт Normal Mode (Обычный режим) или Expert Mode (Экспертный режим).
- Для выбора экспертного режима воспользуйтесь комбинацией клавиш <Cmd+Shift+E> (Macintosh) или <Ctrl+Shift+E> (Windows), а для выбора обычного режима — <Cmd+Shift+N> (Macintosh) или <Ctrl+Shift+N> (Windows).

Какой бы режим **вы** ни выбрали, он будет использоваться для редактирования кода во всех фильмах, начиная с момента выбора и до тех пор, пока вы не перейдете в другой режим. Так будет, даже если вы выйдете из Flash и перезапустите программу.

СОКРЫТИЕ И ОТОБРАЖЕНИЕ СПИСКА ДЕЙСТВИЙ

Чтобы скрыть список действий (левую часть панели Actions) и увеличить место, занимаемое окном действий, щелкните на треугольнике, расположенном между левой и правой частью панели.

Чтобы снова отобразить список действий, щелкните на треугольнике слева от окна действий.

ЗНАКОМСТВО с ОБЪЕКТНО-ОРИЕНТИРОВАННЫМИ ЯЗЫКАМИ

ActionScript является объектно-ориентированным языком программирования. За последние 40 лет объектно-ориентированное программирование (ООП) превратилось в ведущий принцип, используемый при написании кода. В данной главе мы укажем причины этого события и рассмотрим простые примеры. Подробно данный вопрос будет освещаться далее в этой книге, начиная с главы 15 "Объединение предложений в функции".

Совет

В главе 15 "Объединение предложений в функции" описывается основная концепция классов в объектно-ориентированном программировании. В большинстве следующих глав речь пойдет о том, как при создании пользовательских объектов используются встроенные объекты и/или классы.

Объектно-ориентированный означает *основанный на объектах*. *Объект* — это структура данных, позволяющая выполнять в программах те же операции, которые вы выполняете в повседневной жизни: группировать сложные явления под простыми заголовками.

Предположим, я хочу пригласить своего друга Джо прийти на вечеринку и принести с собой гитару. И Джо, и его гитара обладают множеством функций и характеристик, но я просто позволю ему и скажу: "Эй, приятель, ты свободен в субботу вечером? Прекрасно! Приходи ко мне на вечеринку и приноси с собой гитару". Я не буду рассказывать ему, по какой улице он должен идти, чтобы попасть на вечеринку, или какая должна быть гитара, — Джо все это уже знает.

В данном случае Джо является аналогом объекта программирования, и речь идет о двух его свойствах. Одним из свойств является функция, а именно приход на вечеринку. Другим свойством является объект: гитара Джо. (В этой аналогии Джо тоже является объектом, т.е. один объект выступает в качестве свойства другого объекта.)

В ActionScript объекты можно использовать для систематизации сложных поведений и характеристик программ и обеспечения простого взаимодействия с ними. Подобно примеру с Джо, объектом является нечто, к чему можно обратиться по имени, а это нечто может обладать собственными функциями и поведением. Сводя связанные функции и свойства данных под именами объектов, вы облегчаете процедуру работы с программами и их понимание.

Каждый видеоклип во Flash является объектом. Таким образом, даже для тех, кто очень мало знаком с Flash, терминология объектов может быть новой, но сами объекты таковыми не являются.

Рассмотрим три основных компоновочных блока объектов: элементарные данные, функции и структуры данных. Их рассмотрение представляет собой краткий обзор элементов, составляющих программу **ActionScript**. Оставшаяся часть этой книги посвящена детализации этих понятий.

Простейшая форма данных (элементарные данные) представляет собой строки цифр и символов. Например, и число 6, и строка "Hello, world!" являются элементарными данными (частью данных). Еще одним типом являются булевы данные, которые могут принимать только одно из двух значений: **true** (истина) или **false** (ложь).

Совет

Подробная информация об элементарных данных приведена в главе 12 "Управление переменными, данными и типами данных".

Функцией называется именованный блок кода, заключающий в себе процедуру.

Элементарные данные и функции можно сгруппировать в структуры данных. Одним из типов структуры данных является *массив*, который, как правило, представляет собой нумерованный список, начинающийся с нуля. Таким образом, `myArray[0]` является первым элементом массива с именем `myArray`. В массиве, содержащем имена месяцев, будет содержаться элемент `myArray[0] = "January"`.

Совет

Подробно о массивах рассказывается в главе 19 "Использование встроенных базовых объектов".

Объекты являются еще одним способом группирования данных и функций в структуры. Вместо того чтобы доступ к данным осуществлялся с помощью чисел, как это происходит в массивах, в объектах доступ к элементам (называемым *свойствами*) осуществляется с помощью имен.

Ниже приведены примеры некоторых свойств видеоклипов (типа "считывание—запись"). Для определения текущего состояния видеоклипа свойства можно считывать. Их также можно изменять, управляя, таким образом, клипом.

- `_currentFrame` (Текущий кадр). Число, указывающее текущий кадр видеоклипа на временной шкале.
- `_rotation` (Вращение). Число градусов, на которое повернут видеоклип относительно своего исходного положения.
- `_visible` (Видимый). Булево значение, управляющее состоянием отображения видеоклипа.
- `_name` (Имя). Строка, являющаяся именем экземпляра видеоклипа.

Доступ к элементам массива осуществляется посредством записи в квадратных скобках: `myArray[0]`. Доступ к свойствам объекта, как правило, осуществляется посредством "точечного синтаксиса", при котором имя объекта и имя свойства отделяются друг от друга точками, как в приведенных ниже примерах.

```
myClip._currentFrame
myClip._visible
myClip._name
myClip._rotation
myClip._gotoAndPlay()
```

Подводя итоги, можно сказать, что объекты являются наборами свойств. Каждый объект имеет имя, и каждое свойство в пределах объекта тоже имеет свое собственное имя. Имя

свойства ссылаются либо на функцию, либо на данные. Данные могут быть элементарными величинами или структурами данных, как, например, массив или другой объект.

Данные также могут храниться в *переменных*, представляющих собой контейнер, находящийся за пределами любой структуры данных. В приведенном ниже примере выражения число 10 хранится в переменной с именем x.

```
x = 10;
```

Функции в объектах являются методами

И массивы, и объекты могут включать в себя функции. Однако свойства объекта, обращающиеся к функциям или содержащие их, называются *методами*. Функция, являющаяся элементом массива, так и называется *функцией*.

Методы относятся к более широкому понятию *действия*, которое также включает в себя глобальные функции (например, `eval`), которые не связаны ни с каким объектом, операторы (например, `break`), управляющие ходом выполнения программы, а также директивы (например, `include`), передающие инструкции компилятору ActionScript. *Методы* — это наиболее точный термин.

ТРИ КИТА: УДОБОЧИТАЕМОСТЬ, ПОВТОРНОЕ ИСПОЛЬЗОВАНИЕ И ДОСТИЖИМОСТЬ

Каждый программист, работающий во Flash, стремится разработать свой собственный стиль написания, систематизации кода и создания комментариев к программам. Компании, с которыми сотрудничают программисты Flash, работают согласно разным стандартам. Однако, каким бы ни был стиль работы программиста, он всегда стремится, чтобы написанный им код был удобочитаемым, достижимым и чтобы его можно было использовать неоднократно. Чем длиннее и сложнее программа, тем более важно стремиться к достижению этих целей как до начала работы, так и в процессе написания кода.

УДОБОЧИТАЕМОСТЬ

Когда приходится возвращаться к редактированию, усовершенствованию или отладке программы, всегда хочется, чтобы код можно было легко расшифровать, не слишком напрягая при этом ни зрение, ни мозги. Удобочитаемость отчасти заключается в установке интервалов, отступов и применении прописных букв в коде. Удобочитаемость могут улучшить комментарии к коду, а также использовании имен функций, переменных, объектов и массивов, несущих определенную смысловую нагрузку. И наконец, самым лучшим (и самым тяжелым) способом повышения удобочитаемости кода является его написание в более элегантной и простой манере.

УСТАНОВКА ИНТЕРВАЛОВ, ОТСТУПОВ и ПРИМЕНЕНИЕ ПРОПИСНЫХ БУКВ

За исключением лишь некоторых ситуаций, язык ActionScript достаточно гибок к заданию интервалов, отступов и применению прописных букв.

Совет

Некоторые необычные, но важные примеры, касающиеся строгого подхода Flash к применению прописных букв, приведены в главе 16 "Взаимодействие, события и установление последовательности".

Две следующие функции, например, будут одинаково трактоваться интерпретатором ActionScript (программой, интерпретирующей SWF-файлы):

```
function addonetobasescore (basescore){return basescore+1;}
function addOneToBaseScore (baseScore){
    return basescore+1;
}
```

Использование пробелов, отступов и прописных букв делает вторую функцию гораздо более удобочитаемой и, следовательно, более легкой для понимания и редактирования программы.

ДОБАВЛЕНИЕ КОММЕНТАРИЕВ к СЦЕНАРИЮ

Только лишь форматирование кода редко помогает понять, что же происходит в сложной программе. Чтобы элементы программы были более понятными, необходимо добавлять комментарии. Однострочные комментарии помечаются двумя косыми чертами:

```
//ввод: базовый счет, возвращает: базовый счет на единицу больше
```

Кроме того, комментарий может следовать за кодом:

```
baseScore +=1; // baseScore объявляет кадр 1 основной временной шкалы
```

Комментарий может быть и многострочным, например, таким:

```
/* ДОБАВЛЕНИЕ ЕДИНИЦЫ К БАЗОВОМУ СЧЕТУ
Michael Hurwicz - January 2002 - www.maximpulse.com
baseScore является целым числом */
```

Некоторые программисты предпочитают ставить двойную косую черту в каждой строке, поскольку так будет понятнее, что к комментарию относится каждая выделенная строка:

```
////////////////////
// ДОБАВЛЕНИЕ ЕДИНИЦЫ К БАЗОВОМУ СЧЕТУ
// Michael Hurwicz - January 2002 - www.maximpulse.com
// baseScore является целым числом
////////////////////
```

В окне действий комментарий выделяется цветом, поэтому вы всегда будете знать, где он начинается и где заканчивается. Чтобы задать цвет комментария, выполните команду **Edit⇒Preferences**. В диалоговом окне Preferences перейдите к вкладке **ActionScript Editor** и в группе опций Syntax Coloring **щелкните** на кнопке цветовой палитры Comments. Затем в открывшейся цветовой палитре выберите нужный цвет. Точно таким же образом можно задать цвет кода, располагающегося на переднем и заднем плане, ключевых слов, идентификаторов и строк.

Кроме того, Flash допускает хранение кода во внешних файлах.

Совет

Подробная информация о хранении кода во внешних файлах приведена в главе 24 "Коллективное использование ActionScript".

Не все внешние редакторы "понимают" ActionScript, поэтому не всегда с их помощью можно отличить комментарии от кода. В этом случае использование двойной косой черты в каждой строке будет более эффективным, особенно если комментарии длинные. Символы двойной косой черты будут гораздо понятнее и при вставке кода, например, в сообщение электронной почты.

Совет

Возникли проблемы с комментариями? Обратитесь к разделу "Возможные проблемы" в конце этой главы.

СОЗДАНИЕ УДОБОЧИТАЕМЫХ ИМЕН

Вы уже знаете, что в программах **ActionScript** используются элементы нескольких типов (таких как функции, массивы, объекты и переменные), требующие присвоения имен. Как сделать имена удобочитаемыми?

Мы уже рассмотрели один из факторов повышения удобочитаемости имен: применение прописных букв. Прочитать строку `addOneToBaseScore` гораздо легче, чем строку `addonetobasescore`. Как правило, большинство имен во Flash начинаются со строчной буквы. Начальная прописная буква обозначает **класс**.

Еще одним способом повышения удобочитаемости является присвоение имен, которые будут интуитивно понятны вам и, если нужно, то и другим людям. К примеру, вам вряд ли захочется проверять данные функции `addOneToBaseScore`, чтобы запомнить, как она работает.

УПРОЩЕНИЕ, СТАНДАРТИЗАЦИЯ и ОПТИМИЗАЦИЯ КОДА

Самый лучший способ повышения удобочитаемости кода заключается в разработке простых способов достижения целей написания программы. Простой код делает жизнь программиста гораздо проще. Естественно, процесс написания изящного кода гораздо проще выглядит в теории, чем на практике, однако это объективный фактор.

Полезной будет попытка стандартизировать код таким образом, чтобы одни и те же методы или синтаксис можно было использовать снова и снова. В этом случае блоки кода будут понятны с первого же взгляда, если программист хорошо знаком с методикой его написания и используемым синтаксисом.

Простой код обычно выполняется гораздо лучше сложного. Поскольку стандартизируются лучшие примеры кода, то применительно ко всей программе это позволит добиться улучшения ее выполнения в целом. Создание более лаконичного кода будет также способствовать ускорению отладки и облегчению редактирования программы.

Однако в некоторых ситуациях возникает противоречие между быстроедействующим и легко читаемым кодом. В таких случаях лучше использовать быстрый код в программе, а легко читаемый код — в качестве комментария.

ПОВТОРНОЕ ИСПОЛЬЗОВАНИЕ: РАСЧЛЕНЕНИЕ КОДА НА МОДУЛИ

Большинство программистов стараются написать такой код, который можно было бы использовать неоднократно. Все-таки программирование, по сути своей, является процедурой решения той или иной задачи. Если какую-то задачу вы уже решили, зачем изобретать велосипед в дальнейшем? Кроме того, методики, используемые для написания кода, подлежащего повторному использованию, способствуют улучшению его удобочитаемости и повышению опыта программистов.

Возможность повторного использования кода определяется функциями и объектами. Оба этих средства позволяют объединять код в функциональные модули, которые затем можно будет использовать как в данной, так и в других программах. Ту же роль, в случае, если доступ к данным осуществляется посредством цифровых индексов, могут выполнять и массивы.

Кроме того, объединение функциональных возможностей в объекты и функции делает код более удобочитаемым. Пусть, например, приведенный ниже код содержится внутри обработчика события вступительного кадра, присоединенного к левой ноге фигуры человека.

```
degrees++;  
if (degrees < 30) {  
    _rotation++;  
}
```

Вам известно, что всякий раз при достижении видеоклипом вступительного кадра интерпретирующая программа **ActionScript** должна выполнить то, о чем говорится в коде. Но что именно?

Сравним этот пример с приведенной ниже строкой, которая активизирует функцию `lift()`, передавая ей аргумент 30.

```
Lift (30);
```

На заметку

Аргумент — это данные, передаваемые функции. Аргумент заключается в круглые скобки после имени функции.

Не нужно быть гением, чтобы догадаться, что данная функция задает поднятие левой ноги и что аргумент 30 имеет какое-то отношение к тому, насколько высоко будет поднята нога. В данном случае удобочитаемость кода реализуется за счет применения менее интуитивно понятного кода в функции с интуитивно понятным именем.

Теперь скажем, что левая нога представляет собой часть объекта с именем `guy` (человек) и свойством `leftLeg`. Кроме того, свойство `leftLeg` является объектом с методом `lift()`:

```
guy.leftLeg.lift(30);
```

Практически данный код представляет собой полное предложение, имеющее подлежащее, сказуемое, дополнение и определение: "Человек поднимает левую ногу".

Как для объекта с именем `guy` создать часть, являющуюся ногой? Воспользуемся тем фактом, что видеоклипы являются объектами. В данном случае `guy` — это видеоклип, а `leftLeg` — его фрагмент. Функция `lift()` становится методом фрагмента `leftLeg` при ее определении в обработчике события, связанного с данным фрагментом.

И метод `lift()`, и строка `guy.leftLeg.lift()` предполагают, что где-то в другом месте программы задана функция `lift`, выполняющая какую-то операцию, похожую на ту, что задана в виде менее понятного кода, представленного в самом начале данного подраздела. Если взглянуть на определение функции, вы получите более точное представление о том, что означает данная процедура, а именно, что она означает вращение видеоклипа:

```
function lift (degrees) {  
    if (_rotation > degrees) {  
        _rotation--;  
    }  
}
```

Добавив комментарий к определению функции ("используется для поднятия левой ноги"), мы получим вполне удобочитаемый код.

В приведенной ниже простой программе `leglift.fl` показано небольшое уточнение описанной схемы. К функции `lift()` добавлена вторая часть, так что при ее выполнении нога опускается, если величина `_rotation` меньше величины `degrees`, и поднимается, если величина `_rotation` больше величины `degrees`. Если величина `_rotation` равна величине `degrees`, функция не выполняет никакой задачи; это является точкой для остановки движения ноги.

```
// используется для подъема, опускания и остановки движения ноги  
function lift(degrees) {  
    if (_rotation < degrees) {  
        _rotation++; //опустить ногу  
    }  
    if (_rotation > degrees) {  
        _rotation--; //поднять ногу  
    }  
}
```

В приведенном ниже примере даны определения трех функций, изменяющих значение переменной `degrees`, задавая тем самым подъем, опускание или остановку движения ноги.

```
function raise() { degrees = 20; }
function lower() { degrees = 68; }
function halt() { degrees = _rotation; } // lift() ничего не делает
```

Запуск этих трех функций иницируют три кнопки в рабочем поле. Обработчик события вступительного кадра фрагмента ноги постоянно выполняет функцию `lift()`, поэтому соответствующее действие запускается сразу же при изменении переменной `degrees`.

Использование функций и объектов является признаком работы программиста высокого профессионального уровня. Основой этого утверждения является уже упоминавшийся факт, что программирование является решением тех или иных задач, а первым шагом на пути к решению является определение предметной области и самой задачи.

В процессе задания объектов точно определяется предметная область. Скажем, в примере с левой ногой предметной областью является объект `guy.leftLeg`. Аналогично, в процессе задания функции определяется задача. Например, предыдущая функция определяет задачу, как поднятие ноги на определенное число градусов.

Кроме того, функции и объекты помогают разбить большие задачи на более мелкие. Предположим, вы хотите, чтобы нарисованная фигурка прошла по комнате. Функция `walkAcrossTheRoom()` может быть достаточно сложной. Вместо этого можно написать функцию `takeOneStep()` (сделать один шаг) и повторять ее до тех пор, пока фигурка не пересечет всю комнату. Что же будет включено в функцию выполнения одного шага? Фигурка человека должна взмахнуть рукой, передвинуть ногу и переместиться на какое-то расстояние вперед.

Если продолжать действовать таким образом, то в конце концов большая задача будет поделена на несколько небольших частей, достаточно точных и достижимых для решения. Начать работу можно с реализации решений в виде функций. Затем эти функции можно объединить в более сложные функции, объекты или участки кода. Объекты и функции являются великолепным средством решения задач, поскольку помогают создавать простые шаги решения сложных многоэтапных задач.

Совет

Вы готовы писать программу, но не знаете, с чего начать? Обратитесь к разделу "Возможные проблемы" в конце этой главы.

Попытка решить проблему сложных функций и объектов сродни упорядочиванию и структурированию папок при работе с электронной почтой или файлами на жестком диске. По мере увеличения количества почтовых сообщений и файлов их недостаточная систематизация снижает эффективность работы. Аналогичным образом, объекты и функции являются основой организации **ActionScript**. Они не усложняют жизнь программиста, а облегчают ее.

Достижимость: СИСТЕМАТИЗАЦИЯ и ЦЕНТРАЛИЗАЦИЯ КОДА

Существуют два аспекта достижимости: способность интерпретирующей программы находить нужные функции и переменные в течение рабочего цикла (при выполнении программы) и способность программиста находить физические строки кода при необходимости их редактирования.

В объектно-ориентированном программировании инфраструктура, позволяющая интерпретатору **ActionScript** находить при выполнении программы нужные функции, реализуется путем встраивания функций и переменных в объекты и разделения объектов на различные классы. Например, есть метод (функция) `concat()` для массивов и метод `concat()` — для текстовых строк. В каждом случае определение метода хранится в определении класса, и поэтому интерпретирующая программа знает, где искать тот или иной метод и не путает их друг с другом.

Совет

Обратитесь к разделу "Функции как классы: конструктор функций" в главе 15 "Объединение предложений в функции".

Существуют также *правила обзора данных*, которые определяют, является ли переменная, функция или объект одной части программы непосредственно "видимыми" из другой части программы. Проблем обзора данных можно избежать путем централизации кода — сосредоточения как можно большей его части в одном месте.



Например, если весь код находится в пределах одного кадра, то о проблемах обзора данных волноваться не стоит. Во Flash 5 помещение всего кода в один кадр иногда было затруднительным или вообще нежелательным. Что касается Flash MX, то благодаря встроенным обработчикам событий эта процедура значительно облегчена и в целом имеет гораздо меньше отрицательных сторон. (Встроенные, или *динамические*, обработчики событий задаются в кадрах, а не путем присоединения к видеоклипам.)

Совет

Встроенные обработчики событий были описаны выше, в разделе "Выбор действий для кадра или объекта". Более подробно они будут рассмотрены в главе 16 "Взаимодействие, события и установление последовательности".

Совет

Возникли проблемы при преобразовании старого Flash-файла для использования обработчиков событий нового типа? Обратитесь к разделу "Возможные проблемы" в конце этой главы.

Еще одним важным фактором является способность программиста находить и без труда редактировать код. В любой сколько-нибудь сложной программе найдется множество укромных мест и потайных уголков, в которых можно "спрятать" код. Если код разбросан слишком во многих местах, то работать с программой будет достаточно тяжело.

Предположим, что к функции необходимо добавить второй аргумент. Сама функция определена в одном месте, так что изменить ее довольно легко. Однако необходимо по всей программе найти все ссылки на данную функцию и добавить второй аргумент. Если весь код сосредоточен в одном месте, то найти и функцию, и все ссылки на нее будет делом нескольких минут. Если же код разбросан в 20 разных местах, то перед вами стоит уже гораздо более сложная и утомительная задача. Конечно, если внести изменения нужно только один или два раза, то и эта задача не представляет собой большой сложности. Но если на выполнение подобных задач вам придется тратить много времени, то программу нужно будет усовершенствовать.

Централизация кода, вероятно, является наиболее важным фактором реализации легко поддерживаемой программы. За счет встроенных обработчиков событий во Flash MX можно легче избежать эффекта "рассеяния кода", делающего программы тяжелыми для поддержания.

Совет

Вы уже централизовали весь код ActionScript и хотите держать его на панели Actions при работе над графикой других кадров? Обратитесь к разделу "Возможные проблемы" в конце этой главы.

СИСТЕМАТИЗАЦИЯ ДАННЫХ И ФУНКЦИЙ С ПОМОЩЬЮ ВИДЕОКЛИПОВ

До появления Flash MX практически в каждой Flash-программе, в которой применялись сценарии, для систематизации кода использовались видеоклипы. Часто клипы, содержащие код, были "фиктивными", т.е. без графического содержимого. Хотя время от времени бывает удобно присоединить код к видеоклипам, этот подход во Flash MX нужен гораздо меньше, чем во Flash 5. Все, что раньше помещалось в обработчик события `onClipEvent` (во вступительный кадр), присоединяемый к видеоклипу, теперь можно поместить во встроенный обработчик события `onEnterFrame` (в кадре). Существуют также аналогичные встроенные форматы других обработчиков событий.

Принимая решение о необходимости присоединения кода к видеоклипам, подумайте о следующем. Если в кадре 1 данного фильма в рабочем поле есть 20 видеоклипов, то можно поместить действие в обработчик события вступительного кадра каждого видеоклипа и, прежде чем перейти к кадру 2, интерпретирующая Flash-программа будет выполнять каждое из этих действий. Либо можно поместить одно встроенное событие `onEnterFrame` в первый кадр основной временной шкалы, а все действия поместить в один клип. Прежде чем перейти к кадру 2, Flash будет выполнять все действия во встроенном обработчике события. Иными словами, выполняется одна и та же задача, но программа будет поддерживать в 20 раз легче.

ИСПОЛЬЗОВАНИЕ ВРЕМЕННОЙ ШКАЛЫ для СИСТЕМАТИЗАЦИИ КОДА

Для систематизации кода можно использовать **временную** шкалу. Каждый кадр может выступать в роли функции и содержать блок кода, предназначенный для выполнения определенной задачи. Функция запускается посредством действия `play O, gotoAndPlay()` или `gotoAndStop()`, которое направляет воспроизводящую головку к определенному кадру.

В то же время в этом кадре можно анимировать, добавлять или удалять видеоклипы.

Еще лучше будет поместить каждый блок кода в функцию и просто обратиться к этой функции кадра. Это сделает каждый кадр более удобочитаемым, а также позволит централизовать большое количество кода. Например, все функции можно поместить в первый кадр одного слоя.

Систематизация кода на временной шкале является более наглядным подходом, чем реализация в функциях одних и тех же поведений. Однако, в связи с тем, что код в каждом кадре приходится редактировать отдельно, систематизация кода на временной шкале не приводит к его реальной централизации. "Заглянуть" в несколько кадров на временной шкале практически так же тяжело, как и в несколько видеоклипов.

Несмотря на то что данная методика все еще используется, она является "пережитком" более ранних версий Flash, в которых не поддерживались функции. Эта методика достаточно жизнеспособна, особенно если речь идет о простых фильмах, однако в сложных фильмах она становится реальной помехой.

ПОИСК КОДА С ПОМОЩЬЮ ПАНЕЛИ MOVIE EXPLORER

Панель Movie Explorer (Проводник фильма) может помочь в поиске кода **ActionScript** в исходных (`.fla`) файлах Flash. Начните с того, что на панели Movie Explorer (рис. 11.2), в разделе Show (Показать), оставьте активной только кнопку ActionScript. На панели Movie Explorer остаются отображенными только видеоклипы и кнопки, содержащие код ActionScript.

Совет

Подробная информация о панели Movie Explorer представлена в главе 2 "Интерфейс Flash".

Кнопки раздела Show панели Movie Explorer определяют типы программных элементов, отображаемых на данной панели. Перечисляя слева направо, они управляют отображением элементов текста, кнопок, видеоклипов и графики, кода ActionScript, видео, звуков и растровых изображений, а также кадров и слоев. После щелчка на самой правой кнопке откроется отдельное окно для настройки всех параметров отображения панели Movie Explorer.

Чтобы отобразить код ActionScript определенной кнопки или видеоклипа, выделите этот элемент, а затем откройте меню параметров панели (щелкните на кнопке в правом верхнем углу) и выберите из него пункт **Expand Branch** (Развернуть ветвь). Либо щелкните на знаке "плюс", расположенном слева от видеоклипа или кнопки. Однако, чтобы добраться до нужного кода, вам, возможно, придется выполнить эти действия несколько раз.

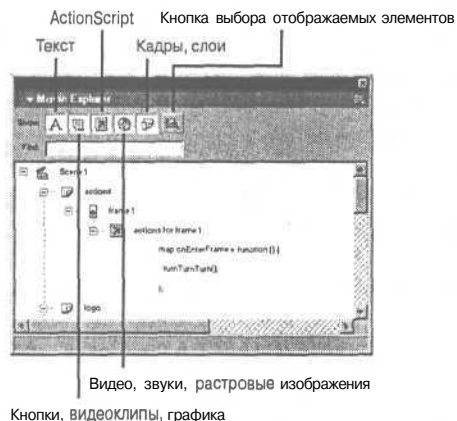


Рис. 11.2. Кнопки раздела Show панели Movie Explorer определяют типы программных элементов, отображаемых на панели

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Я знаю, какие операции должна выполнять программа, но не ЗНАЮ, с чего начать.

Если вы полностью запутались в схеме решения задачи с помощью ActionScript (или любого другого языка программирования), то вероятнее всего, вы еще не разбили одну большую задачу на несколько маленьких. Определите подзадачи и разбейте их на еще более мелкие. Если вы сможете разбить задачу на такие, решение которых реализуется посредством нескольких строк кода, вы на правильном пути.

Можно также попытаться определить упрощенную версию задачи, подлежащей решению. Затем путем последовательных приближений перейдите к решению более сложной задачи.

Мне удалось централизовать весь код ActionScript. Как сохранить его на панели Actions при работе над графикой других кадров ?

Обычно после щелчка на кадре на панели Actions отображается код именно этого кадра, даже если он не содержит ActionScript. Поэтому нужно вернуться назад и щелкнуть на кадре, содержащем код ActionScript. Ужасно утомительное занятие.

Решение заключается в том, что следует воспользоваться кнопкой "прикрепления" текущего сценария, которая расположена в правой части панели Actions, прямо под меню параметров (см. рис. 11.1). Отобразив код ActionScript на панели Actions, щелкните на данной кнопке. Выбранный код будет "прикреплен" к панели Actions и будет оставаться на ней, независимо от того, с каким кадром вы работаете, до тех пор, пока вы его не "открепите", снова щелкнув на этой кнопке.

Я преобразовываю старый проект Flash с тем, чтобы использовать новые обработчики событий. Но они просто не работают! Что сделано неправильно?

Вероятнее всего, Flash MX распознала файл FLA версии Flash 5 и автоматически установила параметры публикации, соответствующие выходному файлу версии Flash 5 (SWF). Обработчики событий нового типа в версии Flash 5 не поддерживаются, и, естественно, возникает проблема. Для ее решения выполните команду File⇒Publish Settings (Файл⇒Параметры публикации). В открывшемся диалоговом окне перейдите к вкладке Flash и из раскрывающегося меню Version (Версия) выберите пункт Flash Player 6.

Могут ли комментарии вызывать появление ошибок?

Да! Есть одно место, где наличие комментариев с завидным постоянством вызывает появление ошибки: это место перед частью else оператора if-else. Рассмотрим следующий код:


```

if (myVar) trace("if");
    //любой комментарий в этой строке приведет к появлению ОШИБКИ!!!
else {
    trace("else");
}

```

Если вы считаете, что возникшая проблема может быть связана с наличием комментария, попробуйте переместить комментарий в новое место или временно его удалить.

Кроме того, рассмотрите возможность, что комментарий мог быть случайно вставлен в каком-то фрагмента кода или что вы забыли убрать комментарий из того места, где он был внедрен при отладке программы.

FLASH ЗА РАБОТОЙ: УДОБОЧИТАЕМЫЙ, МНОГОКРАТНО ИСПОЛЬЗУЕМЫЙ, ДОСТИЖИМЫЙ КОД

Ниже приведен пример того, как сделать код удобочитаемым, возможным для повторного использования и достижимым.

Фильмы, созданные с помощью файлов `unimportant1 fla` и `unimportant2 fla`, содержащихся на прилагаемом к этой книге компакт-диске, выглядят совершенно одинаково (рис. 11.3). Однако код файла `unimportant2 fla` разработан так, что является удобочитаемым, достижимым и доступным для неоднократного использования.

Что касается файла `unimportant1 fla`, то его код, разработанный во времена всеобщего хаоса, выглядит совершенно загадочно:

```

OnClipEvent(load) {
    var frame=1;
    var rotate=5;
    var radian
}
onClipEvent(enterFrame) {
    rotate2=Math.round((rotate/10)+(5*rotate*(Math.abs(Math.cos(radian))))) ;
    this._rotation+=rotate2;
    radian=(Math.PI/180)*(this._rotation);
    if (Math.abs(Math.cos(radian))<.01) {
        frame=(frame+5);
        if (frame>20) {frame=1;}
        this.gotoAndPlay(frame);
        rotate=-rotate;
    }
}

```

В файле `unimportant2 fla` сделано следующее: создана функция `pendulumSwing()`, выполняющая все действия, которые должны быть включены в обработчик события вступления кадра; выполнено преобразование в формат нового обработчика события; присвоено имя экземпляру видеоклипа (так, что им можно управлять с помощью `ActionScript`); добавлены комментарии.

```

_root.happyPerson.frame = 1;
    //seed value for rotation
_root.happyPerson.rotate = 5;
    //rotation in radians
_root.happyPerson.radian;
////////////////////////////////////
//pendulumSwing() function
////////////////////////////////////

```

```

//back and forth pendulum_like motion
//starting rotation should be in approx.range -65 to +65
// (otherwise swing is inverted)
_root.happyPerson.pendulumSwing = function () {
    with (_root.happyPerson) {
        // 10 and 5 are "magic numbers", found by experimentation
        rotate2=Math.round((rotate/10)+(5*rotate*
            (Math.abs(Math.cos(radian)))));
        this._rotation += rotate2;
        // convert to radians for use with Math.cos
        radian=(Math.PI/180)*(this._rotation);
        // true at each extreme of the swing
        // wag the tail and reverse direction
        if (Math.abs(Math.cos(radian))<.01 {
            frame +=5;
            if (frame>20) {frame = 1;}
            this.gotoAndPlay(frame);
            rotate=-rotate; //reverse rotation direction
        }
    }
}
_root.onEnterFrame = function () {
    _root.happyPerson.pendulumSwing();
};

```

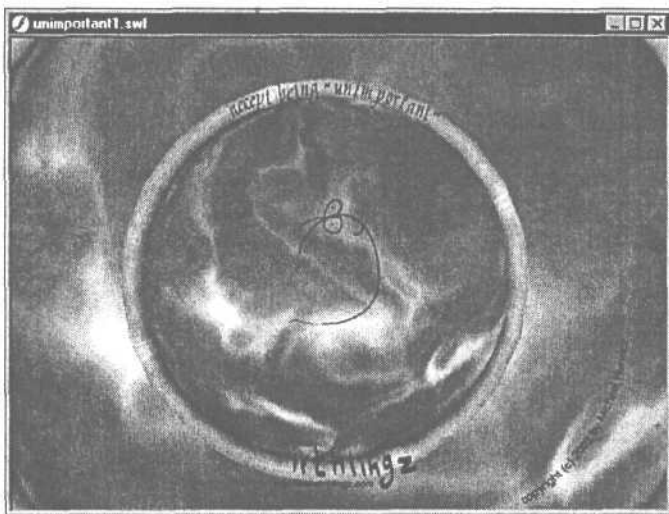


Рис. 11.3. Мультипликация, созданная для узла irthingz.com

Теперь, глядя на код, совершенно понятно, какие операции выполняет программа. Кроме того, у нас есть функция, которую можно повторно использовать в качестве "черного ящика". Для этого не обязательно понимать ее сущность и запоминать, как она получена.

Эта программа очень маленькая, и ее удобочитаемость или возможность повторного использования может не иметь большого значения. А вот в сложных программах, содержащих десятки или сотни страниц кода, удобочитаемость и возможность повторного использования могут иметь очень важное значение.

УПРАВЛЕНИЕ ПЕРЕМЕННЫМИ, ДАННЫМИ И ТИПАМИ ДАННЫХ

В ЭТОЙ ГЛАВЕ...

Работа с переменными	217
Группирование данных по типам	225
Возможные проблемы	241
Flash за работой: детектирование версии	242

РАБОТА С ПЕРЕМЕННЫМИ

Переменная — это имя, которое назначается для элемента данных. *Переменная* называется так, поскольку назначаемые имена могут меняться. Например, в данный момент переменная с именем `myScore` может равняться 10, а через мгновение, когда вы набрали еще одно очко в игре, значение `myScore` уже будет равно 11. Однако этим изменчивость переменных в ActionScript не ограничивается. Одной и той же переменной сначала можно поставить в соответствие число 11, затем — строку "иллюстрации", а после этого — массив строк и чисел.

Объявить переменную можно с помощью ключевого слова `var` следующим образом:

```
var x; // объявляет переменную x без первоначального значения
var y; // объявляет переменную y без первоначального значения
```

В большинстве случаев переменная объявляется только один раз. Затем в программе ее можно использовать много раз по мере необходимости. Например:

```
var x = 10; // объявляет переменную x и задает ее равной 10
trace (x); // отображает "10" в окне вывода данных
```

```
y = x + 10;  
trace (y); // отображает "20" в окне вывода данных
```

Во многих языках программирования переменную следует объявить **прежде**, чем ее можно будет использовать. При работе в **ActionScript** в случае использования переменной, которая не была объявлена, интерпретирующая программа ActionScript создаст переменную **"на лету"** (*неявно*).

Однако признаком хорошего тона считается объявление переменных явным образом и включение в код комментариев о том, как и где используется каждая переменная. Такой прием сделает код более удобочитаемым и, что наиболее важно, позволит однозначно контролировать область действия переменных, т.е. где в программе задается переменная и где к ней можно получить доступ.

Совет

В данной главе затрагиваются некоторые вопросы, связанные с областью действия данных. Более подробно они рассматриваются в главе 15 "Объединение предложений в функции".

Кроме того, явные объявления можно централизовать и, таким образом, их будет легче найти. Объявление переменных неявным образом сродни тому, что при работе в мастерской нужные инструменты будут разбросаны по разным углам. Вы никогда не сможете найти то, что вам нужно в данный момент, но обязательно наткнетесь на эту вещь в тот момент, когда она нужна меньше всего.

Совет

Подробная информация об объявлении переменных приведена ниже, в разделе "Где объявлять переменные".

НАЗНАЧЕНИЕ ЗНАЧЕНИЙ ПЕРЕМЕННЫХ

Назначить значения переменных можно с помощью оператора присваивания (=) или оператора **set**. Например, в обоих приведенных ниже выражениях значение переменной **firstName** устанавливается равной **"John"**:

```
firstName = "John";  
set (firstName , "John");
```

Значение в правой части может быть сложным выражением:

```
X = y + 10;  
set (x, y + 10);
```

Из предыдущих примеров видно, что оператор **set** не дает никакого преимущества, а только усложняет выражение. Однако, если вы хотите использовать для образования имени переменной выражение, оператор **set** позволит сделать это, в отличие от оператора присваивания. Например, в следующем выражении использование оператора **set** будет уместным:

```
1 = 4;  
set("day"+i+"night", "a French movie");  
trace (day4night); // "a French movie"
```

А вот такое выражение приведет к ошибке:

```
"day"+ i+"night" = "a movie"; // ERROR!!!
```

Оператор присваивания можно применять для образования имени переменной с помощью выражения при использовании функции **eval**. Но использование функции **eval** не менее сложно, чем использование оператора **set**:

```
eval("day"+ i+"night") = "a movie";  
trace (day4night); // "a movie"
```

ГДЕ ОБЪЯВЛЯТЬ ПЕРЕМЕННЫЕ

Не существует оптимального для всех программ и программистов набора правил о том, где нужно объявлять переменные. Однако при принятии решения по этому поводу, как правило, преобладающими являются два фактора.

- Возможность найти, понять и отредактировать переменные в процессе работы над программой. Этот вопрос касается разработки программы.
- Что нужно сделать для того, чтобы быть уверенным в том, что интерпретирующая программа **ActionScript** сможет в нужный момент найти переменные? Этот вопрос касается *области действия* переменных.

Первый фактор — возможность найти код — приведет вас к тому, чтобы сосредоточить весь код или большую его часть в одном кадре, например в первом кадре верхнего слоя основной временной шкалы. Именно такой тип **централизации** кода рекомендован компанией Macromedia.

Доступ к переменным основной временной шкалы из подчиненных фрагментов можно получить, используя в качестве пути префикс `_root`. Предположим, например, что в основной временной шкале имеется строка

```
myVar = 20;
```

Доступ к переменной `myVar` из подчиненного фрагмента можно получить следующим образом:

```
trace(_root.myVar); // displays "20"
```

В некоторых ситуациях принцип всеобщей централизации кода не работает. Помимо оп-ределения рамок, в пределах которых этот принцип применим, на повестке дня еще остается вопрос о том, как систематизировать код в пределах одного контейнера так, чтобы макси-мально повысить его удобочитаемость и минимизировать возможность ошибок.

КОГДА НЕ НУЖНО ЦЕНТРАЛИЗОВАТЬ код

Структура программы препятствует централизации кода, по крайней мере, в двух ситуациях.

- **При использовании предварительного загрузчика.** В этом случае идея заключается в том, чтобы как можно быстрее получить предварительный загрузчик, а затем на его фоне выполнить остальные задачи, необходимые для инициализации программы. Предварительный загрузчик обычно занимает один или два кадра в начале фильма. Любые переменные, связанные с предварительным загрузчиком, будут определены в этих начальных кадрах. При запуске основного фильма связанные с ним переменные будут определены в кадре 2 или последующих кадрах.
- **При использовании компонентов.** Компоненты объединяют в себе графику и код. При использовании компонентов невозможно централизовать содержащийся в них код. Кроме того, в основе набора компонентов, вероятнее всего, лежит иерархическая структура классов, согласно которой все компоненты набора принадлежат одному базовому классу, а различные подмножества компонентов принадлежат соответствующим подклассам. В этом случае каждый класс и подкласс имеют свой собственный код, размещенный в определенном месте. Идея заключается в том, что обычному программисту никогда не нужно влезать в этот код.

КАК СИСТЕМАТИЗИРОВАТЬ ПЕРЕМЕННЫЕ В ОСНОВНОЙ ВРЕМЕННОЙ ШКАЛЕ

При централизации переменных на основной временной шкале возникает ряд других вопросов. Размещение огромного числа отдельных переменных во временной шкале сродни хранению в одном ящике кухонной посуды, инструментов для настройки фортепиано и ремонта автомобиля.

Чтобы немного уменьшить путаницу, переменные с помощью комментариев можно разделить на категории. Код каждой категории можно также хранить и редактировать во внешнем файле, а затем загружать его с помощью директивы `tinclude`.

Совет

Подробная информация о хранении кода ActionScript во внешних файлах приведена в главе 24 "Коллективное использование ActionScript".

Однако само по себе использование комментариев и внешних файлов не является решением проблемы. К примеру, каждый видеоклип, имеющий собственную временную шкалу и обработчики событий, определяет одну область действия. То же самое касается и основной временной шкалы.

Временная шкала в чем-то подобна папке с файлами, содержащейся на компьютере, за исключением того, что временная шкала содержит элементы программы, такие как переменные, объекты и функции. Как и в случае с папкой файлов, два элемента программы, содержащиеся в одной временной шкале, не могут иметь одинаковые имена. Если это происходит, то элемент, заданный позже, обычно записывается поверх существовавшего ранее. (В некоторых случаях существующее имя делает новое непригодным к употреблению.)

Если в основной временной шкале просто объявить переменные, без явного назначения области действия каждой из них, то областью действия всех переменных будет основная временная шкала. В этом случае каждая переменная должна иметь уникальное имя. Поэтому, чтобы отличить кухонные принадлежности (`kitchenWax`) от аксессуаров фортепиано (`musicWax`) и инструментов для ремонта автомобиля (`carWax`), имена переменных должны быть достаточно сложными. Кроме того, можно случайно повторно использовать имя временной, переписав предыдущее значение.

Помимо этого, если для загрузки SWF-файла во временную шкалу, содержащую переменные, использовать функцию `loadMovie()`, то переменные будут потеряны, поскольку временная шкала нового видеоклипа заменит существующую. С другой стороны, если для загрузки клипа из библиотеки использовать функцию `attachMovie()`, а имя экземпляра, присвоенное новому клипу, совпадет с именем существующей переменной, объекта, массива или функции той же временной шкалы, то уже существующее имя будет работать, но вы не сможете управлять присоединенным видеоклипом, поскольку имя его экземпляра будет "захвачено" уже существующим.

Код любой временной шкалы очень чувствителен к таким конфликтам пространства имен. Если при поиске элементов фильма вы руководствуетесь принципом вывода всего кода основной временной шкалы, будьте очень внимательны и постарайтесь избежать конфликтов.

ИСПОЛЬЗОВАНИЕ ВИДЕОКЛИПОВ для СИСТЕМАТИЗАЦИИ ПЕРЕМЕННЫХ

Одна из стратегий избежания конфликта пространства имен заключается в использовании временной шкалы видеоклипа и обработчиков событий в качестве контейнеров кода. Например, можно создать клип `kitchen` с таким обработчиком событий:

```
onClipEvent (load) {  
    var wax = true; // у нас есть кухонные принадлежности  
}
```

Областью действия переменных, объявленных в обработчике событий обычного типа, так как вышеприведенный, является видеоклип. К переменным можно обращаться непосредственно из временной шкалы клипа и его обычных обработчиков событий:

```
onClipEvent (enterFrame) {  
    wax = false; // не относится к кухонным принадлежностям  
}
```

Объявление переменных в обычном событии `load` позволяет использовать одно и то же имя переменной (`wax`) в нескольких видеоклипах. Однако, если в другом клипе есть код обращения к переменной клипа `kitchen`, необходимо указывать полный путь к ней:

```
_root.kitchen.wax = false; // обращение к клипу kitchen из другого клипа
```

Неудобство хранения кода в видеоклипах заключается в том, что при наличии большого числа клипов, в которых разбросаны переменные, редактирование кода будет затруднительным.

Одним из возможных решений является использование *клипа-контейнера* (называемого также *клипом кода*) в качестве центрального контейнера кода для нескольких других видеоклипов или даже для всех видеоклипов программы. Размещение всех переменных в одном обработчике события `load` не слишком отличается от их сосредоточения в одном кадре основной временной шкалы. Однако, помня о том, что к клипу кода никогда не нужно присоединять или загружать фильмы, вы устраните одну из основных причин возникновения конфликта пространства имен. Кроме того, эффективным может быть распределение переменных в небольшом количестве клипов-контейнеров.

Еще одним хорошим способом является создание одного или нескольких видеоклипов исключительно для загрузки и присоединения фильмов. Эти клипы являются пустыми, поэтому присоединенный или загруженный фильм не приведет к замещению элементов.

Для облегчения редактирования можно поместить весь код, создающий переменные, в одно место и указать пути к видеоклипам, у которых переменные являются свойствами. С точки зрения интерпретирующей программы `ActionScript` переменные являются частью видеоклипов, для которых они были назначены. Например, предположим, что на основной временной шкале имеются видеоклипы `kitchen`, `shop` и `musicRoom`:

```
_root.kitchen.wax = true;  
_root.shop.wax = false;  
_root.musicRoom.wax = true;
```

Интерпретирующая программа `ActionScript` будет искать указанные переменные `wax` соответственно в клипах `kitchen`, `shop` и `musicRoom`. Такой подход позволяет избежать конфликта пространства имен и при этом централизовать код.

Однако, если загрузить внешний SWF-файл в основную *временную* шкалу, то все содержащиеся в нем видеоклипы и связанные с ними переменные будут уничтожены.

ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ для СИСТЕМАТИЗАЦИИ ПЕРЕМЕННЫХ

Где бы ни был размещен код, переменные можно хранить в объектах в качестве свойств. В этом случае чувствительными к конфликту пространства имен будут только имена объектов верхнего уровня. Например, пусть на основной временной шкале имеются три объекта верхнего уровня: `kitchen`, `shop` и `musicRoom`. В этом случае нужно быть внимательным и не загружать и не присоединять к основной временной шкале видеоклипы, имеющие те же самые имена.

Другие имена свойств, такие как `wax`, "защищены" внутри соответствующих объектов, как если бы они находились внутри видеоклипов. И обращение к ним может выглядеть идентичным образом:

```
_root.kitchen.wax = true;  
_root.shop.wax = false;  
_root.musicRoom.wax = true;
```


Однако пустой видеоклип занимает 800 байт памяти, а новый объект — только 340 байт.

Даже объекты верхнего уровня можно защитить от возникновения конфликтов пространства имен с участием экземпляров видеоклипов. Во Flash 5 программисты иногда прибегали к этому способу, задавая свойства объектов верхнего уровня глобального объекта `Object` следующим образом:

```
Object.kitchen = new Object();
```

Объект `Object.kitchen` защищен от случайного совпадения с именами видеоклипов, поскольку к нему нельзя присоединять или загружать видеоклипы. То же самое касается и любого другого объекта, не являющегося видеоклипом. Единственный недостаток этого подхода заключается в том, что ни к одной из переменных нельзя будет обратиться просто по ее имени. Всегда нужно будет использовать имена типа `Object.kitchen.wax`.



Во Flash MX для достижения той же самой цели есть лучшее средство: идентификатор `_global`, который является обращением к глобальному объекту, содержащему встроенные объекты `ActionScript` и классы, в том числе и сам объект `Object`, а также `Array`, `Math` и `String`.

Переменные можно хранить как глобальные свойства, аналогично приведенному ниже примеру,

```
_global.myGlobalVar = 10;
```

Можно также создать несколько глобальных объектов высокого уровня, таких как `_global.kitchen`, `_global.shop`, `_global.musicRoom`, и поместить в них все остальные переменные, функции, массивы и объекты. Например:

```
_global.kitchen = new Object();  
_global.kitchen.wax = true;
```

Таким образом реализуется безопасная комплексная система хранения всех элементов программы.

Глобальные переменные и объекты доступны из любого места программы. Код, создающий эти элементы, можно сосредоточить на основной временной шкале или в любом другом месте. Глобальные переменные и объекты нельзя повредить при загрузке внешнего SWF-файла, независимо от того, в какую временную шкалу он загружается.

Кроме того, глобальные переменные и объекты можно считывать только по их именам, без необходимости включения идентификатора `_global`. Например:

```
trace(myGlobalVar); // отображается "10"
```

Если вы хотите изменить переменную или свойство, необходимо использовать идентификатор `_global`. Например:

```
_global.myGlobalVar = 11;  
_global.kitchen.wax = true;
```

Использование объектов несколько сказывается на производительности программы. Один из способов, позволяющих обойти эту проблему, заключается в том, что в случае неоднократного обращения к определенному свойству в определенной точке программы необходимо создать временную переменную, указывающую на данное свойство. Например:

```
kWax = Object.kitchen.wax;
```

Использование переменной `kWax`, вместо того, чтобы заставлять интерпретирующую программу снова и снова искать свойство объекта, может существенно повысить производительность. Кроме того, короткие имена переменных быстрее вводить с клавиатуры.

Видеоклипы, объекты и объект `_global` одновременно решают вопросы разработки и области действия, что не под силу только комментариям и внешним файлам. Например, при распределении переменных с комментариями по категориям интерпретирующая программа не понимает этих категорий. Аналогичным образом, при загрузке в одну временную шкалу нескольких файлов с помощью директивы `tinclude` интерпретирующая программа видит переменные в виде одного большого списка. Видеоклипы, объекты и объект `_global` обеспечивают структуру, которую понимает и программист, и интерпретирующая программа.

Многие программисты для систематизации переменных в программах с большим количеством кода стремятся использовать видеоклипы либо объекты (или и те и другие). Однако для обеспечения централизации кода компания Macromedia рекомендует минимизировать количество кода, присоединяемое к видеоклипам и кнопкам. Использование пользовательских объектов открывает совершенно новые перспективы возможностей, со своими проблемами и сложностями.

Совет

Подробная информация о пользовательских объектах представлена в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

Еще одним типом объектов, которые можно использовать для хранения переменных, являются массивы. Хранение переменных в массивах в виде элементов также открывает новые возможности манипулирования переменными.

Совет

Об объекте `Object` и о массивах рассказывается в главе 19 "Использование встроенных базовых объектов".

ПРАВИЛА ПРИСВОЕНИЯ ИМЕН ПЕРЕМЕННЫМ (и ДРУГИМ ЭЛЕМЕНТАМ)

Имя переменной является ее *идентификатором* и представляет собой ряд буквенно-цифровых символов, использующихся для представления элемента данных или функции.

Все имена переменных являются идентификаторами. Однако не все идентификаторы являются именами переменных. Например, имена функций являются идентификаторами, но они не являются переменными, поскольку содержат определенный, неизменный элемент данных (функцию).

При образовании идентификаторов необходимо соблюдать три правила. Пренебрежение ими может привести к возникновению ошибок или вызвать серьезные проблемы.

- Используйте только буквы, цифры, символы подчеркивания и знак доллара. Не используйте пробелы, знаки препинания (точки, запятые, дефисы, круглые и квадратные скобки, восклицательные и вопросительные знаки), а также следующие символы:

~ @ # ^ & * = + / \ < >

- В качестве начального символа используйте букву, символ подчеркивания или знак доллара. Не используйте цифры.
- Не используйте зарезервированные слова.

В **ActionScript** зарезервированными являются следующие слова:

<code>add(*)</code>	<code>else</code>	<code>instanc eof</code>	<code>super</code>
<code>and(*)</code>	<code>eq(*)</code>	<code>le(*)</code>	<code>switch</code> <code>with</code>
<code>break</code>	<code>for</code>	<code>lt(*)</code>	<code>tellTarget(*)</code>
<code>case</code>	<code>function</code>	<code>ne(*)</code>	<code>this</code>
<code>continue</code>	<code>ge(*)</code>	<code>new</code>	<code>typeof</code>
<code>default</code>	<code>gt(*)</code>	<code>not(*)</code>	<code>var</code>
<code>delete</code>	<code>if</code>	<code>or(*)</code>	<code>void</code>
<code>do</code>	<code>in</code>	<code>return</code>	<code>while</code>

(*) Зарезервированные слова *Flash 4*, которые начиная с версии *Flash 5* используются мало.

В приведенном ниже списке перечислены слова, которые в будущем могут стать зарезервированными. Сейчас их использовать можно, но в дальнейшем программа может и не работать, если в следующей версии данное слово станет "собственностью" **ActionScript**.

abstract	enum	int	static
boolean	export	interface	synchronized
byte	extends	long	throw
catch	final	native	throws
char	finally	package	transient
class	float	private	try
const	goto	protected	volatile
debugger	implements	public	
double	import	short	

Тем же самым правилам необходимо следовать при присвоении имен другим элементам, в том числе и при именовании экземпляров видеоклипов. Несмотря на то что официально имена экземпляров видеоклипов *не* являются идентификаторами, они часто выступают в этой роли при использовании в **ActionScript**, и, таким образом, несоблюдение правил также может привести к возникновению ошибок.

Совет

Возникли проблемы с кодом, который содержит имя видеоклипа? Обратитесь к разделу "Возможные проблемы" в конце этой главы.

Многие программисты следуют указанным выше правилам и при именовании других элементов, таких как надписи кадров и имена слоев.

В общем идентификаторы **ActionScript** не чувствительны к регистру символов. Это означает, что приведенные ниже выражения *не* определяют разные переменные.

```
myVar = 6 ;  
myvar = 10 ;  
Myvar = "Hello" ;
```

Недостаточная чувствительность к регистру символов, требующаяся для совместимости с более ранними версиями **Flash** и позаимствованная из стандарта **ECMA-262**, является камнем преткновения для программистов **JavaScript**, привыкших к абсолютной чувствительности к регистру. При работе с **JavaScript** приведенные выше выражения объявляют три различные переменные.

В **ActionScript** чувствительными к регистру символов являются только перечисленные выше зарезервированные слова.

Говоря о чувствительности к регистру, упомянем об одном правиле, связанном с этим вопросом, которому следуют большинство программистов: обычно идентификаторы начинаются со строчного символа. Единственным исключением являются имена функций конструктора.

Совет

Подробная информация о функциях конструктора приведена в главе 15 "Объединение предложений в функции", в главе 19 "Использование встроенных базовых объектов" и в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

Чтобы программы были более удобочитаемыми, желательно следовать одним и тем же правилам применения прописных символов.

ГРУППИРОВАНИЕ ДАННЫХ по ТИПАМ

Главное, что нужно знать о переменной, — к какому типу данных она относится. Например, многие проблемы возникают из-за использования текстовых строк, там где **ActionScript** требует наличия цифр, и наоборот.

Совет

Зачастую из-за использования неправильного типа данных возникают совершенно загадочные ошибки. Обратитесь к разделу "Возможные проблемы" в конце этой главы.

Отследить типы данных может быть достаточно сложно, потому что иногда **ActionScript**, чтобы программа работала надлежащим образом, автоматически преобразовывает их. Чтобы указать, какие типы данных следует применять в том или ином случае, необходимо понимать логику выполняемых преобразований.

ДЕВЯТЬ ТИПОВ ДАННЫХ ACTIONSCRIPT

В **ActionScript** имеется девять основных типов данных. Пять *элементарных* типов данных, наиболее распространенными среди которых являются **number** (число) и **string** (строка), содержат только один примитивный элемент данных. Например, свойство видеоклипа `_currentFrame` содержит только одно число, которое указывает текущее положение воспроизводящей головки на временной шкале. Аналогичным образом, свойство видеоклипа `_name` содержит одну текстовую строку.

Четыре *составных* типа данных (**object**, **array**, **movieclip** и **function**) содержат несколько элементов данных. Например, видеоклип содержит свойства `_currentFrame`, `_name`, `_property` и др.

Определить тип данных любой переменной можно с помощью оператора `typeof` следующим образом:

`typeof выражение`

Рассмотрим следующий пример:

```
var x = 2;  
trace (typeof x); // отображает в окне вывода данных "number"
```

Манипулирование данными с помощью операторов

Наиболее распространенный и основной способ манипулирования данными заключается в использовании *операторов*. Операторы — это символы и ключевые слова, которые принимают одно или несколько существующих значений в качестве *операндов* и возвращают новое значение. Многие операторы, такие как "плюс" (+), "минус" (-) или "разделить" (/), знакомы еще из школьного курса математики. Подробно операторы рассматриваются в главе 13 "Использование операторов".

В табл. 12.1 приведены типы данных **ActionScript** с соответствующими возвращаемыми значениями `typeof`, примеры назначения переменных и разрешенные значения типов данных.

В примерах назначения переменных, приведенных в табл. 12.1, подразумевается, что объект, массив, видеоклип и функция уже существуют.

Совет

В главе 15 "Объединение предложений в функции" рассматривается процедура создания объектов (включая массивы, видеоклипы и функции).

Видеоклипы создают путем перетаскивания символов с панели **Library** (Библиотека) в рабочее поле либо с помощью функции `attachMovie()`, `duplicateMovieClip()` или `loadMovie()`.

ТАБЛИЦА 121. Типы ДАННЫХ ACTIONSCRIPT

ТИП ДАННЫХ	ЗНАЧЕНИЕ TYPEOF	ПРИМЕР НАЗНАЧЕНИЯ	РАЗРЕШЕННЫЕ ЗНАЧЕНИЯ
number (число)	"number"	var x = 6;	Любое число
string (строка)	"string"	var x = "foo"	Любая строка символов
Boolean (булевы)	"boolean"	var x = true	true, false
object (объект)	"object"	var x = myObj	Любой объект
array (массив)	"object"	var x = myArray	Любой массив
null (пустой)	"null"	var x = null	null
movieClip (видеоклип)	"movieClip"	var x = myClip	Любой экземпляр видеоклипа
function (функция)	"function"	var x « myPunc	Любая функция
undefined (неопределенный)	"undefined"	var x;	undefined

Мощь ЧИСЕЛ

Большинство фрагментов фильма, в котором применяются **сценарии**, управляются посредством чисел. Анимация с использованием **ActionScript** в первую очередь зависит от эффективности манипулирования числами. Например, из **19 свойств** объекта MovieClip 13 являются числовыми. Посредством чисел регулируется прозрачность, местоположение, вращение, масштабирование, размеры и положение видеоклипа во временной шкале.

Первым шагом на пути к эффективному использованию чисел является понимание того, какие типы значений представляет тип данных number. Двумя наиболее важными типами чисел являются *целые числа*, не имеющие дробной составляющей (например, -10, 0, 2, **856**), и *числа с плавающей точкой*, имеющие дробную составляющую (например, -10.2, .01, 635.8916).

На заметку

Во Flash используется представление дробных чисел с двойной точностью. Это означает, что числа могут иметь до **15 значащих цифр**. Таким образом, при экспоненциальном представлении чисел во Flash в их основании никогда не будет больше 15 цифр.

К типу данных number также относятся приведенные ниже специальные значения.

- NaN ("Не число") обозначает нечисловые данные в элементе типа number,
- Number.MAX_VALUE — наибольшее число, которое можно представить в ActionScript (1.79769313486231e+308).
- Number.MIN_VALUE — наименьшее положительное число, которое можно представить в ActionScript (5e-324).
- Infinity (Положительная бесконечность) — неопределенное **число**, большее, чем Number.MAX_VALUE.
- -Infinity (Отрицательная бесконечность) — неопределенное число, меньшее, чем Number.MIN_VALUE.
- Константы, такие как pi и квадратный корень из 2; могут храниться как свойства объекта Math.

Подробная информация об объекте Math и об управлении числами с помощью встроенных функций приведена в главе 19 "Использование встроенных базовых объектов".

Экспоненциальное представление чисел

Для представления очень больших или очень малых дробных чисел (как положительных, так и отрицательных) в ActionScript используется экспоненциальное представление. Число в нем представлено в виде основания (мантииссы) и порядка, разделяемых буквой *e* (экспонентой). Чтобы получить основание, поставьте десятичную точку после первого разряда числа и отбросьте все старшие и младшие нулевые разряды. Например, числа 123 000 000 000 000 и .000000000000000123 имеют одинаковое основание: 1.23. Порядок числа указывает, на сколько позиций была смещена десятичная точка. Таким образом, указанные выше числа можно представить в следующем виде: $1.23e+15$ и $1.23e-15$. Экспоненциальным представлением чисел можно воспользоваться и в программах. Например, вместо числа 126 000 можно записать $126e+3$.

Числами можно управлять с помощью операторов или встроенных функций.

Подробная информация об управлении числами с помощью операторов представлена в главе 13 "Использование операторов".

СТРОКИ

Если пользователь вводит данные в программе, вы получаете строковую переменную. Если вы извлекаете имя видеоклипа, то тоже имеете дело со строкой. Фактически из шести нечисловых свойств видеоклипа четыре являются строками. Ключом к эффективному программированию на языке ActionScript является умение манипулировать строками.

Определить тип данных **string** (строка) очень легко: *строкой* является любая последовательность буквенно-цифровых символов, заключенных в кавычки.

Термин "буквенно-цифровой" означает, что тот или иной элемент состоит из буквенных и цифровых символов, а также знаков препинания, математических и других условных символов, используемых в компьютерных системах.



Для представления текста или вводимых пользователем строк Flash MX использует двухбайтовый набор символов DBCS или набор символов Unicode. Последний необходим для отображения множества языков, в том числе китайского и хинди.

В качестве подмножества в Unicode входят наборы символов, которые можно кодировать одним байтом, например **Latin-1** (относящийся к стандарту ISO-8859). Набор **Latin-1** удовлетворит потребности пользователей, работающих на обычных компьютерах и на одном из распространенных западноевропейских языков (английском, французском, немецком, итальянском, португальском, испанском и др.). Flash MX (как и Flash 5) поддерживает кодировку **Shift-JIS**, используемую для отображения японских символов. Данная кодировка является эквивалентом двухбайтового (16-битового) набора символов, предназначенного для японского языка.

С подробной информацией о наборах символов Unicode, Latin-1 и Shift_JIS можно ознакомиться на Web-узле <http://www.unicode.org>.

И еще несколько замечаний, касающихся символов Unicode.

- Чтобы отобразить шрифт, например набор символов для китайского, японского, корейского языков, хинди или иврита, необходимо, чтобы этот шрифт был установлен на компьютере или был внедрен во Flash-фильм.
- Чтобы загрузить символы Unicode из файла, в том числе из XML-файла, он должен быть сохранен в формате Unicode. В наиболее полнофункциональных текстовых редакторах есть опция сохранения в виде текста Unicode. К примеру, если вы хотите загрузить символы Unicode из файла `test.xml`, обязательно сохраните его как текст Unicode:

```
var myXML = new XML();
myXML.load("test.xml");
```

- Чтобы получить возможность отображения символов Unicode, не загруженных из файла, можно воспользоваться недокументированным свойством `System.useCodepage`. В приведенном ниже примере показано отображение строки двухбайтовых символов.

```
System.useCodepage = true;
aeioun="%E1+%E9+%ED+%F3+%FA+%F1";
trace(unescape(aeioun)) ;
```

Для создания Flash-фильмов с динамической загрузкой любых из миллиона возможных символов Unicode можно воспользоваться Flash-генератором многосимвольного набора, swfx.org. При этом подразумевается, что у вас есть шрифт с нужными символами, который можно внедрить в SWF-файл.

Чтобы в ActionScript создать код символов Unicode, используются длинные или короткие управляющие последовательности Unicode. В длинной форме используются символ обратной косой черты (\) и символы, за которыми следуют четыре шестнадцатеричных цифры. В короткой форме используется символ обратной косой черты (\) и символ х, за которыми следуют две шестнадцатеричных цифры. Приведем несколько примеров:

```
trace ('\u00A3'); // знак английского фунта стерлингов: £
trace ('\xA3'); // знак английского фунта стерлингов: £
trace ('\u00A9'); // символ авторского права: ©
trace ('\xA9'); // символ авторского права: ©
```

Строки можно объединять, или *связывать*, с помощью оператора "плюс" (+), как показано ниже.

```
trace ("This "+"works!"); // отображается "This works!"
```

В строках также применяются оператор присваивания (=), оператор равенства (==), оператор неравенства (!=), оператор строгого равенства (===) и оператор строгого неравенства (!==). Например:

```
x = "foo";
y = "foo";
trace (x ==y); // "true" -
trace (x ===y); // "true"
trace (x !=y); // "false"
trace (x !==y); // "false"
```

Строками манипулируют с помощью операторов и встроенных функций.

Совет

Подробная информация о манипулировании строками с помощью операторов представлена в главе 13 "Использование операторов".

УПРАВЛЕНИЕ И ПРИНЯТИЕ РЕШЕНИЙ с помощью БУЛЕВЫХ ДАННЫХ

Тип данных Boolean (булевы) исключительно прост, поскольку использует только два значения: `true` (истина) и `false` (ложь). Этот тип данных широко используется при принятии решений, т.е. при проверке того, является ли определенное условие истинным или ложным, и решения о том, что следует делать дальше. Например, приведенное ниже выражение проверяет, является ли видеоклип `myClip` видимым.

```
if (myClip._visible == true)
```

Если клип является видимым, интерпретирующая программа `ActionScript` назначает для выражения `(myClip._visible == true)` булево значение `true`. В противном случае для выражения назначается значение `false`.

Аналогичным образом, следующее выражение устанавливается в значение `true`, если объект `myClip` является видимым:

```
if (myClip._visible == false)
```

Обратите внимание на то, что булевы данные в двух приведенных выражениях используются по-разному: в качестве значения свойства `_visible` (видимый) видеоклипа и в качестве ответа интерпретирующей программы на ваши вопросы.

В приведенном ниже выражении булевы данные используются в качестве значения свойства, а не в качестве ответа на вопрос.

```
myClip._visible == true; // делает клип видимым
```

В приведенной ниже двухстрочной программе булевы данные нет, но интерпретирующая программа `ActionScript` откликнется на вторую строку, отобразив булево значение `true`.

```
x = "Barbara";
trace ( x == "Barbara" ); // "true"
```

Как правило, булевыми данными управляют с помощью **операторов**, хотя к ним можно применить и встроенные функции.

МОДЕЛИРОВАНИЕ ОБЛАСТИ ДЕЙСТВИЯ ЗАДАЧИ с помощью ОБЪЕКТОВ

Тип данных `object` в `ActionScript` является наиболее гибким и основополагающим. Простота и гибкость этого типа данных делают его пригодным для моделирования практически всего, что можно включить в программу, какой бы ни была область действия задачи.

Структура типа данных `object` очень проста — это неупорядоченный набор свойств, каждое из которых представляет собой пару "имя—значение". Свойство объекта может содержать любой тип данных, поддерживаемый `ActionScript`.

Многие операторы, широко использующиеся с элементарными типами данных, неприемлемы к объектам и другим составным типам данных. Например, объекты нельзя складывать, вычитать или умножать.

Однако есть один оператор, который применяется только к объектам, в том числе и к видеоклипам. Это оператор "точка" (`.`), официально именуемый "оператором доступа к свойству объекта". Для доступа к свойствам объекта можно также использовать квадратные скобки, которые официально называются оператором "массив-элемент/объект-свойство". В целом формат выражения выглядит следующим образом:

`object [строка имени свойства]`

Обратите внимание на то, что имя свойства обязательно должно быть строкой. Это означает, что оно должно быть заключено в квадратные скобки. Таким образом, `root.myObj["name"]` — то же самое, что и `_root.myObj.name`. Помните, что объекты являются свойствами временной шкалы, в которой они находятся, поэтому существует и третий способ написания данного выражения: `_root["myObj"]["name"]`.

Элемент данных типа `object` создается с помощью оператора `new`. Другими операторами, которые можно использовать применительно к объектам, являются `delete`, `typeof`, равенство (`==`), строгое равенство (`===`), неравенство (`!=`), строгое неравенство (`!==`) и присваивание (`=`).

Совет

Подробная информация об операторах, которые можно использовать с объектами, приведена в главе 13 "Использование операторов".

Сравнение объектов

Два объекта не будут считаться равными только из-за того, что они имеют одинаковые свойства с одинаковыми значениями. Два объекта с разными именами будут равными только в том случае, если будут ссылаться на один и тот же объект.

Совет

Подробная информация о сравнении объектов приведена в главе 13 "Использование операторов".

Совет

Некоторые преимущества использования объектов в программах рассмотрены в главе 11 "Знакомство с ActionScript".

Совет

В главе 19 "Использование встроенных базовых объектов" подробно описан встроенный объект `Object`, являющийся основой всех других объектов ActionScript.

Совет

Чтобы узнать, как создаются пользовательские объекты, обратитесь к главе 15 "Объединение предложений в функции".

МАССИВ

Массив — это упорядоченный список. Например, массив можно использовать для хранения названий дней недели или месяцев. Доступ к *элементам* списка осуществляется посредством цифровых индексов. Начальным индексом всегда является 0. В приведенном ниже примере первому элементу массива `myArray` присваивается значение "Monday" (понедельник), а затем он отображается в окне вывода данных.

```
myArray[0] = "Monday";  
trace (myArray[0] ); // "Monday"
```

Фактически массив является особым случаем типа данных `object`. Как и объекты, массивы создаются с помощью оператора `new`. Применительно к массивам используются те же операторы, что и для объектов, за исключением оператора "точка" (`.`), который зарезервирован для объектов (в том числе и для видеоклипов).

Как и в случае с типом данных `object`, для интерпретирующей программы `ActionScript` одинаковое содержимое не делает два массива равными. Если имена `arr1` и `arr2` относятся к массивам, то приведенное ниже выражение будет справедливо только в том случае, если `arr1` и `arr2` являются двумя именами *одного и того же массива*.

```
(arr1 == arr2)
```

Совет

Подробная информация о сравнении массивов приведена в главе 13 "Использование операторов".

Как и объекты, массивы могут содержать любые данные `ActionScript`.

Совет

Цифровые индексы, используемые для доступа к массивам, открывают некоторые возможности, недоступные для объектов. Эти возможности рассматриваются в главе 19 "Использование встроенных базовых объектов".

ТИП ДАННЫХ `NULL`

Тип данных `null` разрешает только одно значение: элементарное значение `null`. Это значение ставится в соответствие контейнеру данных (например, переменной, свойству объекта или элементу массива) для указания того, что контейнер является пустым. Значение `null` можно также присвоить идентификатору, используемому в качестве имени функции, что приведет к отключению данной функции.

Тип данных `null` тесно связан с типом данных `undefined`. Это подтверждается тем фактом, что интерпретирующая программа `ActionScript` считает их равными:

```
trace (null == undefined); // "true"
```

Не существует ничего больше (кроме самого `null`), что интерпретирующая программа рассматривала бы равным `null`. Например:

```
trace (null == ""); // "false" - пустая строка не считается равной null
```

Совет

Отличия между типами данных `null` и `undefined` будут рассмотрены ниже, в разделе "Использование типов данных `null` и `undefined`".

ТИП ДАННЫХ `MOVIECLIP`

Тип данных `movieClip` знаком каждому программисту, работающему с `Flash`. Его отличительной чертой является графическое содержимое, которым можно управлять посредством свойств видеоклипа. Таким образом, `movieClip` является единственным встроенным объектом `Flash`, имеющим атрибуты цвета, местоположения, вращения, масштабирования, размеров и прозрачности. Добавление этого типа данных было одним из способов адаптивирования стандарта `ECMA-262` к `Flash`. В стандарте `ECMA-262` и в языке `JavaScript` тип данных `movieClip` отсутствует.

Применительно к видеоклипам можно использовать шесть операторов: "точка" (`.`) и "квадратные скобки" — для доступа к свойствам, а также операторы равенства (`==`), неравенства (`!=`), строгого равенства (`===`) и строгого неравенства (`!==`) — для сравнения. Как и в

случае с объектами и массивами, если (`myClip1 == myClip2`), то `myClip1` и `myClip2` являются двумя именами одного и того же видеоклипа.

Совет

Подробно о сравнении видеоклипов рассказывается в главе 13 "Использование операторов".

Видеоклипы не создаются с помощью оператора `new` и не удаляются с помощью оператора `delete`.

Совет

Видеоклипы создаются с помощью средств графического пользовательского интерфейса Flash с помощью методик, описанных в главе 6 "Символы, экземпляры и элементы библиотек", или с помощью методик, описанных в главе 17 "Демонстрация мощи видеоклипов".

Совет

С помощью методик, описанных в главе 17 "Демонстрация мощи видеоклипов", можно создавать новые экземпляры существующих видеоклипов и удалять их, а также загружать и выгружать внешние SWF-файлы.

ФУНКЦИЯ

Функция — это именованный блок кода **ActionScript**, выполняющий определенную задачу. Ряд *глобальных* функций, таких как `trace()`, не являются свойствами любого из объектов. Функции, являющиеся свойствами объектов, называются *методами*. Например, функция `gotoAndStop()` является методом объекта `movieClip`, вызывающим переход видеоклипа к определенному кадру во временной шкале и его последующую остановку.

Часто функция в качестве входных значений принимает один или несколько *аргументов* и возвращает элемент данных. Например, `isNaN()` является глобальной функцией, которая в качестве аргумента принимает любое выражение и возвращает либо булево значение `true` (если выражение, трактуемое как число, принимает специальное значение NaN (не число)), либо `false` (в противном случае).

Хотя это не так очевидно, как в случае с массивами, тип данных **function** фактически также является особым случаем типа данных **object**. Несмотря на то что программисты практически не используют эту возможность, функции могут иметь свойства, доступ к которым реализуется посредством тех же операторов "точка" и "квадратные скобки", которые используются для типа данных `object`.

Тот факт, что функция является специальным типом объекта, выступает на первый план в объектно-ориентированном программировании. Является ли проблемой, что функции являются объектами и одновременно могут быть свойствами объектов? Совсем нет. Объекты в качестве свойств могут содержать другие объекты, в том числе и функции.

Несмотря на сходство с объектами, функции нельзя создавать или удалять программным образом: операторы `new` и `delete` с функциями не работают. Однако отключить функцию можно, присвоив ее имени значение `null`:

```
function foo() { // сейчас вы ее видите
}
foo = null; // сейчас нет
```

Совет

Подробная информация о функциях, в том числе о функциях конструктора, использующихся для создания объектов, содержится в главе 15 "Объединение предложений в функции".

ТИП ДАННЫХ UNDEFINED

Как и тип данных `null`, тип данных `undefined` указывает на отсутствие данных. Однако, если значение `null` вы задаете самостоятельно, то значение `undefined` по умолчанию присваивается интерпретирующей программой **ActionScript** в случае, когда переменной или другому контейнеру данных не присвоено никакого значения. Например:

```
var middleName;  
trace (typeof middleName); // "undefined"
```

Интерпретирующая программа **ActionScript** возвратит значение `undefined` и в случае обращения к несуществующей переменной или другому контейнеру данных. (JavaScript в таких условиях генерирует ошибку.)

ActionScript берет свое начало в стандарте **ECMA-22** и отличается от **JavaScript** своей трактовкой значения `undefined` в контексте строки. Когда интерпретирующая программа **ActionScript** сталкивается со значением `undefined` там, где она ожидает обнаружить строку, обычно выполняется преобразование `undefined` в пустую строку (`" "`). Элемент данных, равный пустой строке, отображаться не будет.



Во **Flash MX** есть одно исключение из этого правила — при наличии одиночного элемента `undefined`. В приведенном ниже примере, который является продолжением предыдущего, **Flash MX** отобразит значение `"undefined"`, тогда как **Flash 5** не покажет ничего (отобразит пустую строку).

```
trace (middleName); // Flash MX отображает "undefined"
```

Интерпретирующая программа **JavaScript** в этом случае, в соответствии со стандартом **ECMA-262**, также отобразит значение `"undefined"`.

Однако в следующем примере и **Flash MX**, и **Flash 5** для переменной `middleName` будут отображать пустую строку, а не `"undefined"`:

```
trace ("my "+middleName); // Flash отображает только "my "
```

В этом случае **ActionScript** для реализации обратной совместимости с кодом программы **Flash 4** отходит от стандарта. **Flash 4** не поддерживает тип данных `undefined`, поэтому для выполнения проверки на отсутствие данных программисты часто используют пустую строку:

```
if (middleName eq "")
```

Условие не выполняется, если `undefined` будет равно чему-либо, кроме пустой строки.

Из-за этой особенности для определения принадлежности данных типу `undefined` предпочтительнее использовать оператор `typeof`.

```
if (typeof middleName == "undefined"); // true
```

Совет

Отличия между типами данных `null` и `undefined` будут рассмотрены ниже, в разделе "Использование типов данных `null` и `undefined`".

НЕЯВНОЕ И ЯВНОЕ ПРЕОБРАЗОВАНИЕ ДАННЫХ

В языках со строгими типами данных, таких как **Java**, при создании или объявлении переменной необходимо объявить и ее тип данных, например `integer` (целое число) или `string` (строка). Если явным образом не изменить тип данных (*приведение типа*), **Java** использует исходное объявление, генерируя ошибку, например, при попытке назначения строкового значения для целочисленной переменной.

В противоположность этому, **ActionScript** (как и JavaScript) является языком с *нестрогими типами данных*. В любом месте программы интерпретатор ActionScript по своему усмотрению может назначить для величины любой тип данных, не требуя от программиста выполнения явного приведения типа. Это называется *неявным* преобразованием типа данных.

Не все преобразования типов данных в ActionScript являются неявными. Аналогом приведения типа является назначение типа данных для величины явным образом. Например, можно преобразовать число в строку с помощью функции **String()** или **ToString()**.

Совет

Подробная информация о преобразовании чисел в строку приведена ниже, в разделе "Преобразование в строку".

Явное преобразование типа данных не будет вызывать слишком больших проблем, поскольку программисты осознают, что они делают. С другой стороны, программист может даже не знать о том, что было выполнено неявное преобразование типа, и это может привести к неожиданным результатам.

Однако, зная правила неявного преобразования типа данных, можно избежать неприятных сюрпризов и извлекать из такого преобразования максимум пользы. Мы уже сталкивались с одним из этих правил, а именно с преобразованием типа **undefined** в пустую строку, когда этот тип данных используется в качестве строки.

В целом, если интерпретирующая программа ActionScript ожидает один тип данных, а вы используете другой, то она создаст новую величину ожидаемого типа. Предположим, что в программе содержатся следующие выражения:

```
x = 63 - "my dentist";  
trace(x); // ??
```

Что отобразит интерпретирующая программа? Чем для нее является переменная *x*?

Если попросить интерпретирующую программу ActionScript вычесть строку из числа, она попытается выполнить эту задачу путем преобразования строки в число и последующего вычитания. В данном случае строку **"my dentist"** успешно преобразовать в число нельзя. В таких случаях создается значение **NaN**. Интерпретирующая программа ActionScript преобразует строку **"my dentist"** в значение **NaN**, официальным типом данных которого является **number** (число). Результатом любых математических операций с участием значения **NaN** также является значение **NaN**. Следовательно, переменная *x* станет равной **NaN**.

В данном случае неявное преобразование типа данных работает не слишком хорошо, однако оно позволяет программе продолжать работать (в отличие от Java, где альтернативой неявному преобразованию является генерация ошибки). Опять же, интерпретирующая программа ActionScript вынуждена выполнять невозможную задачу. С другой стороны, если имеется такая строка, как **"3"**, которую *можно* преобразовать в число, то интерпретирующая программа ActionScript выполнит вполне полезную операцию:

```
x = 63 - "3";  
trace(x); // 60
```

Результатом неявного преобразования всегда является элементарный тип данных: **string** (строка), **number** (число) или **Boolean** (булево значение). В табл. 12.2 представлены правила преобразования в число.

ТАБЛИЦА 12.2. НЕЯВНОЕ ПРЕОБРАЗОВАНИЕ в число

ВХОДНОЕ ЗНАЧЕНИЕ	ЗНАЧЕНИЕ ПОСЛЕ ПРЕОБРАЗОВАНИЯ
undefined	0
null	0

ВХОДНОЕ ЗНАЧЕНИЕ	ЗНАЧЕНИЕ ПОСЛЕ ПРЕОБРАЗОВАНИЯ
Булево значение <code>false</code>	0
Булево значение <code>true</code>	1
Строка чисел, которую можно преобразовать в число. (Может содержать цифры, десятичную точку, знаки "плюс" и "минус" и пробел.)	Эквивалентное числовое значение. (Исключение: оператор "плюс" связывает все строки, в том числе и числовые. См. главу 13 "Использование операторов"
Не числовая строка (это может быть пустая строка или любая строка, содержащая буквенные символы, в том числе начинающаяся с символов <code>x</code> , <code>0x</code> и <code>ff</code>)	NaN
"Infinity"	"Infinity"
"-Infinity"	"-Infinity"
"NaN"	"NaN"
Любой массив	"NaN"
Любой объект	"NaN"
Любой видеоклип	"NaN"

В табл. 12.3 представлены правила преобразования в строку.

ТАБЛИЦА 12.3. НЕЯВНОЕ ПРЕОБРАЗОВАНИЕ в СТРОКУ

ВХОДНОЕ ЗНАЧЕНИЕ	ЗНАЧЕНИЕ ПОСЛЕ ПРЕОБРАЗОВАНИЯ
<code>undefined</code>	" " (пустая строка)
<code>null</code>	"null"
Булево значение <code>false</code>	"false"
Булево значение <code>true</code>	"true"
NaN	"NaN"
0	"0"
<code>Infinity</code>	"Infinity"
<code>-Infinity</code>	"-Infinity"
Любое другое числовое значение	Строка, эквивалентная числовому значению
Любой массив	Список значений элементов, разделяемых запятыми
Любой объект	Значение, возвращаемое при использовании для данного объекта метода <code>toString()</code> . По умолчанию это значение "[object Object]". Встроенный объект <code>Date</code> возвращает дату
Любой видеоклип	Абсолютный путь к экземпляру видеоклипа. Например, <code>"_level10.myClip"</code>

В табл. 12.4 представлены правила преобразования в булевы значения.

ТАБЛИЦА 124. НЕЯВНОЕ ПРЕОБРАЗОВАНИЕ в БУЛЕВЫ ЗНАЧЕНИЯ

ВХОДНОЕЗНАЧЕНИЕ	ЗНАЧЕНИЕ ПОСЛЕ ПРЕОБРАЗОВАНИЯ
undefined	false
null	false
NaN	false
0	false
Infinity	true
- Infinity	true
Любое другое числовое значение	true
Пустая строка	false
Любая строка, которую можно преобразовать в число, не равное нулю	true
Любая строка, которую нельзя преобразовать в число, не равное нулю. (Отход от стандарта ECMA-262 для поддержания совместимости с Flash 4.)	false
Любой массив	true
Любой объект	true
Любой видеоклип	true

ЯВНОЕ ПРЕОБРАЗОВАНИЕ ТИПОВ ДАННЫХ

В большинстве случаев неявных преобразований типов данных вполне достаточно. Flash автоматически преобразует данные в тип, подходящий для той или иной ситуации. Однако в некоторых случаях легче всего получить нужный тип путем явного указания Flash, с каким типом данных нужно иметь дело.

Например, любая информация, вводимая пользователем в предназначенное для этого поле, первоначально является строкой. Если вы хотите сложить числа, представленные в двух таких строках, то сначала их нужно преобразовать. В противном случае при использовании оператора “плюс” Flash будет предполагать, что требуется объединить две строки.

Любой элемент данных можно явным образом преобразовать в число, строку или булево значение. Преобразование будет проводиться согласно правилам, приведенным в табл. 12.2–12.4.

ПРЕОБРАЗОВАНИЕ в число

Преобразование в число осуществляется с помощью глобальной функции `Number()`. Например, если в текстовое поле пользователь вводит свой возраст, то первоначально эти данные являются строкой. Преобразование в число выполняется следующим образом:

```
user1Age = Number(user1Age);
```

Если пользователь вводит "23", то интерпретирующая программа трактует предыдущую строку следующим образом:

```
user1Age = Number("23");
```

Интерпретирующая программа устанавливает значение переменной `user1Age`, равным 23.

Функция `Number()` может воспринимать десятичные числа с показателем степени:

```
Number("2.637e-4"); // 0.0002637
```

Если, как в следующем примере, строка начинается с символов Ох, числа трактуются как шестнадцатеричные:

```
greenColor = "0x00FF00"; // шестнадцатеричное представление зеленого
цвета, строка
hexGreen = Number(greenColor); // 65280, десятичная форма 0x00FF00
```

Функции `parseInt()` и `parseFloat()` также преобразуют строки в числа, но при этом строки должны иметь следующий формат: первый символ, не являющийся пробелом, должен быть первым символом искомого числа, а первый символ после искомого числа должен быть нечисловым. В приведенных ниже примерах используется функция `parseInt()`.

```
1: parseInt("12years") ; // извлекает 12
2: parseInt(" 12years") ; // извлекает 12
3: parseInt("12 years") ; // извлекает 12
4: parseInt("12 1 year olds") ; // извлекает 12
5: parseInt("twelve 1 year olds") ; // NaN
```

Посмотрим на строку 1. Первым после числа стоит символ у, который является нечисловым. В строке 2 показано, что первый пробел не имеет значения. В строках 3 и 4 продемонстрировано, что символ пробела, следующий за числом, трактуется как нечисловой. В строке 5 первый символ, не являющийся пробелом, не является первым символом числа, и поэтому из данной строки извлекается значение NaN.

Функция `parseInt()` может также принимать второй аргумент, указывающий *основание* (порядок) результата. Например:

```
parseInt("10",16); // извлекает 16 - шестнадцатеричное 10 (основание 16)
parseInt("10",8); // извлекает 8 - восьмеричное 10 (основание 8)
parseInt("10",10); // извлекает 10 - десятичное 10 (основание 10)
```

Функция `parseInt()` и шестнадцатеричные числа

Функция `parseInt()` всегда подразумевает, что числа, начинающиеся с Ох, являются шестнадцатеричными:

```
parseInt("0x10"); // 16
```

Попытка уйти от неявного шестнадцатеричного представления не всегда приводит к полезным результатам:

```
parseInt("0x10", 10); // 0
```

```
parseInt("0x10", 8); // 0.
```

Числа, начинающиеся с о (но не с Ох), по умолчанию являются восьмеричными:

```
parseInt("010"); //8
```

В отличие от шестнадцатеричного представления, в данном случае можно заменить неявное восьмеричное основание и получить полезные результаты:

```
parseInt("010", 10); // извлекается десятичное число, равное 10
```

Не забывайте заключать строку в кавычки. Без них можно обойтись только иногда (но далеко не всегда):

```
trace(parseInt(010)); // 0 - это не то, что нужно !!!
```

В приведенных ниже примерах показано, как используется функция `parseFloat()`.

```
trace(parseFloat("12.5years")) ; // извлекает 12,5
```



```

trace(parseFloat(" 12.5years")) ; // извлекает 12,5
trace(parseFloat("12.5 years")) ; // извлекает 12,5
trace(parseFloat("12.5 1 year olds")) ; // извлекает 12,5
trace(parseInt("twelve and a half 1 year olds")) ; // NaN

```

В последнем примере извлекается значение NaN, так как первый символ, не являющийся пробелом, не является числовым.

ПРЕОБРАЗОВАНИЕ В СТРОКУ

Преобразование в строку можно выполнять либо с помощью метода `toString()`, либо с помощью глобальной функции `String()`. По умолчанию выполняется одна и та же операция, хотя синтаксис используется разный:

```

var x = 6;
x.toString(); // "6"
String(x); // "6"

```

Несмотря на то что методы принадлежат только объектам, неявное преобразование типов данных позволяет использовать метод `toString()` применительно к любым типам данных в соответствии с правилами, приведенными в табл. 12.3.

```

var x = 6;
trace(x.toString()); // "6"
y = true;
trace(y.toString()); // "true"
z = null;
trace(z.toString()); // "undefined"
trace(Math.toString()); // "[object Object]" - Math является встроен-
ным объектом

```

Метод `toString()` имеет дополнительный аргумент, устанавливающий основание результата в случае, если элемент данных является числом. Например, если число и основание одинаковы (например, число 6 по основанию 6, число 7 по основанию 7), то результатом будет "10".

```

trace(x.toString(x)); // "10" - истина, если x является любым числом

```

Ниже приведен другой пример. Обратите внимание на круглые скобки, в которые заключено число 65280.

```

(65280).toString(16); // "ff00"

```

НЕЯВНОЕ ПРЕОБРАЗОВАНИЕ ТИПОВ ДАННЫХ

Чтобы извлечь максимум пользы от неявного преобразования типов данных, можно пуститься на несколько уловок. Преимущество неявного преобразования заключается в очень кратком синтаксисе. С другой стороны, при чтении кода он не всегда будет интуитивно понятным.

Например, оператор "добавить и переназначить" (`+=`) может выполнять ту же самую задачу, что и метод `toString()` или глобальная функция `String()`. При использовании последних более очевидно, что производится преобразование в строку, но оператор "добавить и переназначить" является более кратким.

Любое элемент данных можно преобразовать в строку в соответствии с правилами, изложенными в табл. 12.3, если к нему добавить пустую строку. Так, в приведенном ниже примере вторая строка выполняет ту же самую операцию, что и функция `String()` или метод `toString()`.

```

x = 77; // x является числом
x += ""; // сейчас x является строкой: "77"

```

Операторы "добавить и переназначить" и "вычитать и переназначить"

Оператор "добавить и переназначить" (`+=`) добавляет второй операнд к первому и ставит полученный результат в соответствие первому операнду. Например, `x += ""` эквивалентно `x = x + ""`.

Оператор "вычитать и переназначить" (`-=`) вычитает второй операнд из первого и ставит полученный результат в соответствие первому операнду. Например, `x -= 0` эквивалентно `x = x - 0`.

Таким образом, в соответствии с правилами, приведенными в табл. 12.2, вычитание 0 из любого элемента данных преобразует этот элемент в число. Например:

```
x = new Object(); // x является объектом
x -= 0; // x является числом: NaN
```

ЧИСЛА И СТРОКИ ПРОТИВ числовых и СТРОКОВЫХ ЛИТЕРАЛОВ

Переменная — это абстрактное представление элемента данных. Она никоим образом не похожа на сам элемент данных. Это означает, что переменная `x` может представлять любой элемент данных, который только можно вообразить. В противоположность переменной, *литерал* представляет элемент данных буквально, а не абстрактно. Литерал описывает элемент данных совершенно точно, а интерпретирующая программа **ActionScript** фактически создает элемент данных, используя литерал в качестве проекта.

В приведенном ниже примере переменная `x` сначала задает числовой, а затем — строковый литерал. Переменная `x` — одна и та же, но содержащееся в ней значение изменяется и, следовательно, **так же** изменяется тип данных.

```
var x;
trace (typeof x); // "undefined"
// 6 является числовым литералом
x = 6; // x теперь относится к числовому литералу
trace (typeof x); // "number"
// "Howdy Pete!" является строковым литералом
x = "Howdy Pete!" // x теперь относится к строковому литералу
trace (typeof x); // "string"
```

Переменная является числом, если назначенное ей значение является числовым литералом. Она также является числом, если назначенное ей значение является сложным выражением, которое можно упростить до числового литерала. Например:

```
x = 3 + 3; // может быть упрощено до числового литерала: 6
```

Как вы видите, строки также можно образовывать с помощью сложных выражений:

```
x = "Howdy" + "Pete!"; // превращается в строковый литерал: "Howdy Pete!"
```

Литералы не могут занимать несколько строк кода. Например, приведенное ниже выражение не будет корректным.

```
test = "Howdy
Pete!";
```

Чтобы обойти это ограничение, можно воспользоваться оператором (`+`):

```
test = "Howdy" +
"Pete!";
```

Строковый или числовой литерал можно использовать временно, без сохранения. Если вы хотите повторно использовать элемент данных, представленный строковым или числовым литералом, его нужно сохранить в виде переменной, элемента массива или свойства объекта. Если не сохранить литерал, он будет непригоден для дальнейшего использования.

ЛИТЕРАЛЫ ОБЪЕКТОВ и МАССИВОВ

Массив — это именованный упорядоченный список, доступ к элементам которого осуществляется посредством числовых индексов. *Литерал массива* — это разделенный запятыми список элементов, заключенных в квадратные скобки, без имени массива или индексов. Например, ниже приведены первые три значения массива строк, представляющих собой календарные месяцы.

```
months[0] = "January";  
months[1] = "February";  
months[2] = "March";
```

Соответствующий литерал массива выглядит следующим образом:

```
["January" , "February" , "March"]
```

Создать массив можно путем назначения для переменной анонимного, безымянного литерала массива. В результате переменная становится именем массива:

```
months = ["January" , "February" , "March"];
```

По аналогии, *объектом* является именованный неупорядоченный список, доступ к свойствам которого осуществляется по имени. Например, ниже представлены свойства объекта `computer`.

```
computer.monitor = "SVGA";  
computer.processor = "Pentium 4";  
computer.price = 1700;
```

На заметку

Формат литерала объекта дает хорошее визуальное представление объекта, а синтаксис *свойство: значение* представляет собой краткий способ обращения к объектам и свойствам.

Соответствующий литерал объекта представляет собой разделенный запятыми список свойств, заключенный в фигурные скобки, без имени объекта. Каждое свойство состоит из имени и значения, разделенных символом двоеточия. Например:

```
{ monitor : "SVGA" , processor : "Pentium 4" , price : 1700 }
```

Для переменной можно назначить литерал объекта, в результате чего она становится именем объекта:

```
computer = { monitor : "SVGA" , processor : "Pentium 4" , price : 1700 };
```

Как и в случае со строковыми и числовыми литералами, литералы массива и объекта будут утеряны, если они не поставлены в соответствие переменной или не сохранены в виде элемента массива или свойства объекта.

ИСПОЛЬЗОВАНИЕ ТИПОВ ДАННЫХ NULL и UNDEFINED

Типы данных `null` и `undefined` исключительно полезны для программиста, но используются они по-разному, несмотря на то, что оба обозначают отсутствие данных. Значение

`null` используется в сообщениях программиста интерпретирующей программе, а значение `undefined` зарезервировано для сообщений интерпретирующей программы программисту.

Интерпретирующая программа `ActionScript` по умолчанию назначает тип данных `undefined`, когда объявленной переменной не поставлено в соответствие никакого содержимого. Такое объявление представлено в строке 1 приведенного ниже примера. Интерпретирующая программа также возвращает значение `undefined` при запросе типа несуществующей переменной. Это продемонстрировано в строке 7.

Присваивать переменной значение `undefined` разрешено, но делать это не рекомендуется. (Такая нежелательная практика продемонстрирована в строке 5.) Если не назначать тип данных `undefined` самостоятельно, то когда вы увидите это значение, всегда будете знать, что интерпретирующая программа присвоила его автоматически. Если вы хотите присвоить переменной значение, указывающее на отсутствие данных, воспользуйтесь значением `null`. Тогда `undefined` всегда будет означать нечто, что не было определено.

```
1: var X; // правильно
2: trace (typeof x); // "undefined"
3: x = null; // правильно
4: trace (typeof x); // "null"
5: x = undefined; // разрешено, но нежелательно !!!
6: trace (typeof x); // "undefined" : значение не присвоено
7: trace (typeof y); // "undefined" : не существует
```

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему я не могу из субклипа обратиться к переменной на основной временной шкале?

Если переменные хранятся в основной временной шкале, а доступ к ним производится из субклипа, то при обращении к переменным не забудьте о префиксе `_root`. Это наиболее распространенная ошибка. Глядя на переменную, объявленную в основной временной шкале, вы не видите префикса `_root` и поэтому забываете указать его при обращении к переменной.

Вам надоело каждый раз добавлять `_root` ко всем именам переменных? Задайте глобальные переменные с помощью идентификатора `_global`. Доступ к глобальной переменной для операций считывания можно получить с помощью только имени переменной.

Может ли использование определенной переменной или имени экземпляра видеоклипа привести к возникновению проблем ?

Да! Чаше всего причиной ошибки является использование зарезервированного слова в качестве имени переменной, имени функции или имени объекта. Несмотря на то что вам, скорее всего, никогда не понадобятся все ключевые слова `ActionScript`, лучше все же знать их и избегать их использования.

Использование зарезервированного слова в качестве имени экземпляра видеоклипа может вызвать ошибки, отследить которые будет очень тяжело. Например, `le` является зарезервированным словом. И хотя во Flash разрешается использовать его в качестве имени экземпляра видеоклипа, при попытке использовать это имя в `ActionScript` возникают проблемы. В приведенном ниже примере видеоклип имеет имя `le`, являющееся зарезервированным словом. Данный код обычно меняет цвет контура видеоклипа на зеленый. Однако в связи с использованием зарезервированного имени он работать не будет.

```
var green = 65280 // числовой код зеленого цвета
myColor = new Color("le"); // "le" является именем экземпляра видеоклипа
myColor.setRGB(green); // ЭТОТ КОД НЕ РАБОТАЕТ!
```

Код, приведенный в предыдущем примере, просто не будет работать. А вот следующий код тоже не будет работать, но при этом сгенерирует сообщение об ошибке (Expected a field name after '.'):

```
_root._le.gotoAndStop(1); // ERROR!!!
```

Всегда ли выводится сообщение об ошибке, когда что-то сделано неправильно?

К сожалению, нет. Например, при использовании несоответствующих типов данных могут генерироваться так называемые "тихие" ошибки, при которых **ActionScript** не выводит никакого сообщения об ошибке. Код просто не работает надлежащим образом. Например, приведенный ниже код дублирует видеоклип **myClip**, создавая его копию под именем **myClip1**.

```
duplicateMovieClip ("myClip", "myClip1", 1);
```

А вот следующий код работать не будет, причем без сообщения об ошибке:

```
duplicateMovieClip ("myClip", myClip1, 1);
```

В чем причина ошибки? Без заключения в кавычки **myClip1** не является ни текстовой строкой, ни переменной, относящейся к строке. А для корректной работы кода требуется выполнение либо одного, либо другого условия.

FLASH ЗА РАБОТОЙ: ДЕТЕКТИРОВАНИЕ ВЕРСИИ

Доступ к переменной **\$version**, введенной в версии *Flash 4.0r11*, можно получить программным образом, помимо использования комбинаций клавиш **<Cmd+Option+V>** (Macintosh) или **<Ctrl+Alt+V>** (Windows) (в среде проектирования при прогоне программы). Переменную **\$version** нельзя использовать для того, чтобы выяснить, установлен ли в системе Flash-плеер, поскольку она используется только внутри уже выполняющейся Flash-программы. Стандартным способом проверки наличия в системе Flash-плеера является использование сценария JavaScript на HTML-странице. Однако переменная **\$version** имеет свои преимущества. К примеру, она работает в браузерах, не поддерживающих сценариев, или в браузерах с отключенной поддержкой сценариев.

Во Flash 5 была включена новая функция **getVersion()**, которая проверяла переменную **\$version** и возвращала результат. Но если вы хотите иметь возможность выявить наличие плеера Flash 4, необходима функция, работающая во Flash 4.

Приведенный ниже сценарий используется для получения версии Flash-плеера с помощью переменной **\$version** и помещает три ее основных компонента (платформу, основную версию и дополнительную версию) в **другие** переменные.

Этот сценарий можно скопировать из фильма **detect fla** и вставить в кадр нового фильма. Чтобы сценарий был максимально эффективным, можно сделать "ответвление" к альтернативному содержанию или, например, обновить страницу на основе искомой версии.

```
// Примеры строк $version:
// "WIN 6,0,21,0"
// "MAC 4,0,11,0"
// "UNIX 5,0,30,0"
```

```
// делит $version на 2 части, в месте " " (символ пробела)
// и помещает эти части в массив с именем platformVersionArray
platformVersionArray = $version.split(" "),•
```

```

// Вы получаете:
// platformVersionArray[0] == т.е., WIN or MAC
// platformVersionArray[1]) == т.е., 6,0,10,0
// назначает первый элемент переменной "platform"
platform = platformVersionArray[0]; // WIN или MAC

// разделяет второй элемент на 4 части, в месте "," (символ запятой)
// и помещает эти части в массив с именем versionArray
versionArray = platformVersionArray[1].split(",");

// присваивает значения второго и третьего элементов соответствующим
переменным
majorVersion = versionArray[0]; // 6
minorVersion = versionArray[2]; //10

// Это место ответвления
// на основе номера основной/дополнительной версии:
if (majorVersion >= 4) {
    if (majorVersion >= 6) {
        // Последняя и лучшая версия!
    }
    if ((majorVersion >= 5) && (minorVersion >= 41)) {
        // Версии плеера 5.0r41 и 42 обрабатывают файлы XML лучше.
    }
    // Версия плеера 4.0r11 или последующая.
    if ((majorVersion >= 4) && (minorVersion >= 20)) {
        // Версия плеера 4.0r20 поддерживает печать.
    }
} else {
    // Версия до 4.0r11
}

```


ИСПОЛЬЗОВАНИЕ ОПЕРАТОРОВ

В ЭТОЙ ГЛАВЕ...

Операнды, выражения и предложения	245
Арифметические операторы	250
Побитовые операторы	254
Присваивание и сложное присваивание	260
Операторы сравнения	262
Логические (булевы) операторы	266
Условный оператор	268
Оператор "запятая"	270
Именованные операторы	271
Другие операторы	273
Возможные проблемы	274
Flash за работой: применение побитовых операторов к игре в крестики-нолики	275

ОПЕРАНДЫ, ВЫРАЖЕНИЯ и ПРЕДЛОЖЕНИЯ

Операторы — это символы и ключевые слова, использующиеся для изменения, доступа, создания, удаления, анализа и систематизации данных. Большинство операторов берет свое начало в областях математики и логики. Ряд из них знакомы всем еще из школьного курса математики. Например, оператор умножения (*) выполняет умножение двух значений, оператор суммирования (+) складывает два значения, оператор деления (/) делит одно значение на другое.

В некоторых случаях форма оператора **ActionScript** отличается от той, что рассматривалась в школьном курсе математики. Например, в **ActionScript** оператором умножения является звездочка (*), а не символ \times . Оператором деления является косая черта (/), а не символ \div .

Несмотря на кажущуюся простоту, операторы являются чрезвычайно мощными инструментами, и их использование может потребовать определенных умений. Всего существует 52 оператора, 51 из которых указан в табл. 13.1.



Оператор `super`, новый для Flash MX, рассматривается в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

Сорок операторов разделяются на пять основных категорий.

- *Арифметические* операторы работают с числовыми операндами с целью получения числового результата.
- Операторы *присваивания* назначают результат для переменной, свойства объекта или элемента массива. Операторы *переназначения* наряду с присваиванием выполняют арифметические или побитовые операции.
- *Побитовые* операторы работают с отдельными битами в пределах байта.
- Операторы *сравнения* сравнивают два значения.
- *Логические* (булевы) операторы сокращают выражения до булевых значений (`true` — истина или `false` — ложь) и исходя из этих значений возвращают результаты.

Есть еще 12 "единичных" операторов, не относящихся ни к одной из категорий. Это операторы массив—элемент/объект—свойство, запятая, условный, `delete`, точка, обращение к функции, группировка, `new`, `typeof`, `instanceof`, `void` и `super`.

ТАБЛИЦА 13.1. ОПЕРАТОРЫ ACTIONSCRIPT

ОПЕРАТОР	КАТЕГОРИЯ	ИСПОЛЬЗОВАНИЕ	СТАРШИНСТВО	АССОЦИАТИВНОСТЬ
<code>x++</code>	Арифметический	Постфиксное приращение	16	L-R
<code>x--</code>	Арифметический	Постфиксное отрицательное приращение	16	L-R
<code>.</code>	Нет	Доступ к свойству объекта	15	L-R
<code>[]</code>	Нет	Массив—элемент/объект-свойство	15	L-R
<code>()</code>	Нет	Круглые скобки, группирование	15	L-R
<code>function()</code>	Нет	Круглые скобки, вызов функции	15	L-R
<code>++x</code>	Арифметический	Префиксное приращение	14	R-L
<code>--x</code>	Арифметический	Префиксное отрицательное приращение	14	R-L
<code>-</code>	Арифметический	Унарное отрицание	14	R-L
<code>~</code>	Побитовый	Побитовое НЕ	14	R-L
<code>!</code>	Логический	Логическое НЕ	14	R-L
<code>new</code>	Нет	Создание объекта/массива	14	R-L

ОПЕРАТОР	КАТЕГОРИЯ	ИСПОЛЬЗОВАНИЕ	СТАРШИН- СТВО	АССОЦИАТИВ- НОСТЬ
delete	Нет	Удаление объекта/свойства/элемента массива	14	R-L
typeof	Нет	Определение типа данных	14	R-L
void	нет	Возвращение неопределенного значения	14	R-L
*	Арифметический	Умножение	13	L-R
/	Арифметический	Деление	13	L-R
%	Арифметический	Деление по модулю	13	L-R
+	Арифметический	Сложение (чисел)	12	L-R
	Строковый	Объединение (строк)		
-	Арифметический	Вычитание	12	L-R
<<	Побитовый	Побитовый сдвиг влево	11	L-R
>>	Побитовый	Побитовый сдвиг вправо со знаком	11	L-R
>>>	Побитовый	Побитовый сдвиг вправо без знака	11	L-R
<	Сравнение	Меньше	10	L-R
<=	Сравнение	Меньше или равно	10	L-R
>	Сравнение	Больше	10	L-R
>=	Сравнение	Больше или равно	10	L-R
instance of	Нет	Определение класса	10	L-R
==	Сравнение	Равенство	9	L-R
!=	Сравнение	Неравенство	9	L-R
===	Сравнение	Строгое равенство	9	L-R
!==	Сравнение	Строгое неравенство	9	L-R
&	Побитовый	Побитовое И	8	L-R
^	Побитовый	Побитовое и сключающее ИЛИ	7	L-R
	Побитовый	Побитовое ИЛИ	6	L-R
&&	Логический	Логическое И	5	L-R
	Логический	Логическое ИЛИ	4	L-R
?:	Нет	Условный	3	R-L
=	Присваивание	Назначение	2	R-L
+=	Присваивание	Сложение и переназначение	2	R-L
-=	Присваивание	Вычитание и переназначение	2	R-L

ОПЕРАТОР	КАТЕГОРИЯ	ИСПОЛЬЗОВАНИЕ	СТАРШИН-СТВО	АССОЦИАТИВНОСТЬ
*=	Присваивание	Умножение и переназначение	2	R-L
/=	Присваивание	Деление и переназначение	2	R-L
%=	Присваивание	Деление по модулю и переназначение	2	R-L
<<=	Побитовый	Сдвиг разряда влево и переназначение	2	R-L
>>=	Побитовый	Сдвиг разряда вправо и переназначение	2	R-L
>>>=	Побитовый	Сдвиг разряда вправо (без знака) и переназначение	2	R-L
&=	Побитовый	Побитовое И и переназначение	2	R-L
^=	Побитовый	Побитовое исключающее ИЛИ и переназначение	2	R-L
=	Побитовый	Побитовое ИЛИ и переназначение	2	R-L
,	Нет	Запятая	1	L-R

ОПРЕДЕЛЕНИЕ ВЗАИМОСВЯЗЕЙ ПРОСТЫХ ДАННЫХ с помощью ВЫРАЖЕНИЙ

Операнды — это то, с чем работают операторы. Операторы вместе с операндами образуют *выражения*. Выражение является частью кода, выдающего одно значение. Ниже приведены примеры выражений.

3 + 7 // операндами являются 3 и 7, оператором является +, результатом является 10

"Bed #" + 6 // операндами являются "Bed #" и 6, оператором является +, результатом является "Bed #6"

a-b // операндами являются a и b, оператором является -, результатом является разность a и b

x++ // операндом является x, оператором является ++, результатом является сумма x + 1

Предложение — это наименьшая единица кода **ActionScript**, которая может вызвать определенные действия программы. Часто для превращения выражения в предложение достаточно символа точки с запятой, как в следующих примерах:

x++; // полное предложение, увеличение на 1

x--; // полное предложение, уменьшение на 1

При использовании символа точки с запятой многие выражения становятся правильными предложениями, но не обязательно полезными. Например, следующие предложения дают результат, но этот результат никак не используется:

```
3 + 7;  
a - b;
```

А вот примеры полезных выражений:

```
x = 3 + 7; // результат хранится в переменной x  
trace(a-b); //результат отображается в окне вывода данных
```

Внимание!

Если в предложении пропустить символ точки с запятой, интерпретирующая программа ActionScript постарается установить, где должен стоять этот символ. Однако пропускать данный знак пунктуации не рекомендуется, поскольку это может привести к ошибкам.

СТАРШИНСТВО, АССОЦИАТИВНОСТЬ И ГРУППИРОВАНИЕ ОПЕРАТОРОВ

Когда одно утверждение включает в себя несколько операторов, то интерпретирующая программа ActionScript определяет, какой оператор следует рассматривать первым, согласно следующим двум правилам.

- Каждый оператор имеет значение старшинства, указанное в табл. 13.1. Интерпретирующая программа рассматривает операторы в порядке старшинства; более старшие операторы рассматриваются первыми.
- Операторы, имеющие одинаковое старшинство, рассматриваются в соответствии с *ассоциативностью*, либо слева направо (L-R), либо справа налево (R-L). В табл. 13.1 указана ассоциативность каждого оператора.

Ниже приведены примеры старшинства:

```
1 + 2 * 10 // 21 — умножение выполняется перед сложением  
--6 * 100 // 500 — отрицательное приращение выполняется перед умножением
```

Круглые скобки можно использовать в качестве *операторов группирования* для изменения заданного по умолчанию порядка выполнения операции. Например:

```
(1+2) * 10 // 30 — сложение выполняется перед умножением  
--(6 * 100) // 599 — умножение выполняется перед отрицательным приращением
```

Круглые скобки можно использовать и для того, чтобы сделать предложение более очевидным, даже если порядок выполнения операций при этом не меняется. Например:

```
1 + (2 * 10) // 21 — ничего не меняется
```

Ниже приведен пример ассоциативности. Операторы "сложение и переназначение" ($+=$) и "вычитание и переназначение" ($-=$) имеют одинаковое старшинство и ассоциативность "справа налево". Следовательно, в выражении $a += b -= c$ интерпретирующая программа начнет с рассмотрения оператора, стоящего справа, т.е. $a += b -= c$ — то же самое, что и $a += (b -= c)$. В приведенном ниже примере интерпретирующая программа сначала рассматривает выражение $b -= c$, делая его равным -1. Затем она рассматривает выражение $a += b$, делая его равным 0.

```
a = 1;  
b = 2;  
c = 3;  
a += b -= c; // результат: a == 0, b == -1, c == 3
```

АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

Арифметические операторы выполняют математические операции с числовыми операндами с целью получения числовых результатов. Интерпретирующая программа **ActionScript** автоматически преобразовывает нечисловые операнды в числа для использования с арифметическими операторами.

СПЕЦИАЛЬНОЕ ЗНАЧЕНИЕ NaN ("НЕ число")

Если операнд, использующийся с арифметическим оператором, нельзя преобразовать в число, то интерпретирующая программа **ActionScript** изменит его на специальное значение NaN ("Не число").

Возможно, хотя это и бывает редко, получить значение NaN в результате манипулирования числами. Например, при делении нуля на ноль, получим значение NaN:

```
trace(0/0); // NaN
```

Чаще всего значение NaN получается при неудачной попытке преобразования строки в число.

При любой математической операции с участием операнда, значение которого равно NaN, будет получен результат NaN.

ПОЛОЖИТЕЛЬНОЕ И ОТРИЦАТЕЛЬНОЕ ПРИРАЩЕНИЕ

Оператор приращения, представляющий собой два знака "плюс" (**++**), добавляет к операнду 1. Оператор отрицательного приращения, представляющий собой два знака "минус" (**--**), вычитает из операнда 1. Например:

```
x = 1;
x++; // теперь x равен 2

y = 2;
y--; // теперь y равен 1
```

Оба оператора приращения можно использовать в качестве *префикса* или *постфикса*, т.е. перед операндом или после него. Если выполняется только положительное или отрицательное приращение выражения, то использование префиксной или постфиксной формы не имеет значения, хотя по традиции чаще используют постфиксную форму.

Однако, если в этом же выражении с полученным результатом производятся какие-либо действия (например, сохранение его в переменной или отображение в окне вывода), то пре-

фиксная и постфиксная формы функционируют по-разному. Префиксная форма сначала изменяет операнд, а затем выполняет с ним какое-то действие. Постфиксная форма сначала манипулирует операндом, а затем изменяет его.

В приведенных ниже двух выражениях операнд *x* всегда изменяется одним и тем же образом: начиная со значения 2 и заканчивая значением 3. При использовании постфиксного приращения переменная *y* принимает первоначальное значение *x*, поскольку назначение выполняется до изменения переменной *x*. При использовании префиксного приращения переменная *y* принимает конечное значение *x*, поскольку назначение выполняется после изменения переменной *x*. (Это можно осмыслить следующим образом: когда оператор находится около переменной *y*, он влияет на нее. Когда оператор не находится около переменной *y*, он не влияет на нее.)

```
X = 2;
y = x++; // постфикс, результатом является x: 3, y: 2
y = ++x; // префикс, результатом является x: 3, y: 3
```

То же правило применимо и к оператору отрицательного приращения. В приведенных ниже выражениях конечное и первоначальное значения *x* равны соответственно 2 и 1. При использовании постфиксного отрицательного приращения конечным значением *y* является первоначальное значение *x*, а при использовании префиксного отрицательного — конечное значение *x*.

```
x = 2;
y m x--; // X: 1, y: 2
y = --x; // X: 1, y: 1
```

И снова, оператор отрицательного приращения влияет на переменную *y*, когда стоит рядом с ней, а когда рядом с *y* стоит *x*, получаем просто значение *x*.

Аналогичным образом, в следующем примере показано, что оператор приращения влияет на отображение переменной, когда он стоит рядом с ключевым словом `trace`:

```
X = 2;
trace(x++); // отображается 2 , x изменяется, но не влияет на trace

X = 2;
trace(++x); // отображается 3 , x изменяется и влияет на trace
```

Операторы положительного и отрицательного приращения широко используются в циклических последовательностях.

Совет

Подробная информация о циклах представлена в главе 14 "Работа с данными: использование предложений".

СЛОЖЕНИЕ И ВЫЧИТАНИЕ

Сложение и вычитание отличаются от положительного и отрицательного приращения следующим.

- Требуют наличия двух операндов.
- Не изменяют операнды, а просто выдают результат.

Рассмотрим следующий пример:

```
a = 6;
b = 2;
c = 1
d = a - b + c; // Теперь d равно 5. Никаких изменений a, b или c, они также
               // равны соответственно 6, 2 и 1.
```

Совет

Предложения, содержащие несколько операторов, описывались выше, в разделе "Старшинство, ассоциативность и группирование операторов".

Операндами могут быть любые выражения, преобразующиеся в вещественные числа. Операнды, не удовлетворяющие этому условию, преобразуются в значение NaN.

Если необходимо получить абсолютную (положительную) разность двух чисел, выполните вычитание и примените к результату функцию `Math.abs`:

```
c = Math.abs(a-b); // если b больше a, то c все равно
                  // будет положительным
```

Совет

Подробно об объекте `Math` рассказывается в главе 19 "Использование встроенных базовых объектов".

ПОЛИМОРФНЫЙ ОПЕРАТОР +

С числовыми операндами оператор `+` выполняет простое сложение. Например:

```
2 + 2 // получается 4
```

Однако оператор `+` является *полиморфным*. Это означает, что он выполняет разные операции с различными типами данных или классами. В частности, он связывает строки и складывает числа. Если хотя бы один из операндов является строкой, то результатом операции также будет строка. Например:

```
x = "1" + 6; // "16"
trace(typeof x); // string
```

```
x = 6 + "1"; // "61"
trace(typeof x); // string
```

Такое поведение можно изменить, явным образом преобразовав строки в числа, например:

```
x = 6 + Number("1"); // 7
trace(typeof x); // number
```

Другие арифметические операторы не являются *полиморфными*, и в результате операций с их участием всегда получаются числа, даже если все операнды являются строками, как в приведенных ниже примерах.

```
x = "3" * "7"; // 21, число; умножение не полиморфно
x = "7" - "3"; // 4, число; вычитание не полиморфно
x = "6" / "3"; // 2, число; деление не полиморфно
```

ПЕРЕГРУЖЕННЫЙ ОПЕРАТОР -

Несмотря на то что оператор `-` ("минус" и унарное отрицание) не является полиморфным, т.е. не выполняет разные операции с различными типами данных или классами, он является *перегруженным*. Это означает, что он выполняет разные операции в пределах одного типа данных или класса в зависимости от количества операндов.

При наличии одного операнда он является оператором *унарного отрицания*, меняющим знак числового значения на противоположный. Положительное значение становится отрицательным и наоборот, как показано в следующем примере:

```
x = 8;
trace(-x); // отображается -8 (отрицательное 8)
```

При наличии двух операторов мы имеем дело с хорошо знакомым *вычитанием*, как показано в следующем примере:
2 -2 // получаем 0 (нуль)

УМНОЖЕНИЕ и ДЕЛЕНИЕ

Операции умножения и деления могут вызвать некоторые проблемы. Они выполняются так же, как и в обычной арифметике. Как обычно, при наличии неразрешенных операций интерпретирующая программа **ActionScript** автоматически преобразовывает типы данных (а не выдает ошибки или исключения, как некоторые другие языки программирования, например Java). В обычной арифметике при делении на нуль результат получается неопределенным, а во Flash результатом выполнения этой операции будет значение **Infinity**(бесконечность):

```
x = 0;  
y = 8;  
z = y / x; // z равно бесконечности
```

Результатом деления целых чисел может быть дробное число. Об этом можно было даже не упоминать, если бы не тот факт, что в Macromedia Lingo (язык программирования для приложения Director) при делении целых чисел результатом всегда будет целое число. В приведенном ниже примере показано, что в ActionScript при делении целых чисел можно получить дробь.

```
x = 17;  
y = 8;  
z = x / y; // z равно 2,125
```

ДЕЛЕНИЕ по МОДУЛЮ (%)

При выполнении операции *деления по модулю* мы получаем остаток, или *модуль* от операции деления. Например, результатом операции 11 % 3 будет 2, так как в результате будет получено частное 3 и остаток 2. Ниже приведено еще **несколько** примеров.

```
4 % 3 // 1  
16 % 6 // 4  
200 % 100 // 0  
18 % 10 // 8  
10 % 9.5 // .5
```

Обратите внимание на последний пример, в котором показано, что в отличие от других языков программирования (в частности, C и C++) в ActionScript при делении по модулю в качестве операнда можно использовать числа с плавающей точкой.

Операцию деления по модулю можно также использовать для выполнения задачи через равные промежутки времени. Например:

```
onClipEvent (load) {  
    var interval = 5;  
    var frameCounter = 0;  
}  
onClipEvent (enterFrame){  
    if( ++frameCounter % interval == 0){  
        // Выполните какое-то действие. Оно будет происходить через каждые  
        // пять кадров.  
    }  
}
```


Приведенный ниже код фильма `leglift2 fla` является модифицированной версией кода `leglift fla`, который рассматривался в главе 11 "Знакомство с **ActionScript**". Здесь с помощью операции деления по модулю ногу можно заставить двигаться в два раза медленнее.

```
onClipEvent(load) {
    var slowdown = 2; // добавлено
    var frameCounter = 0; // добавлено
    var degrees = _rotation;
    function raise0 j degrees = 20; }
    function lower() { degrees = 68; }
    function halt() { degrees = _rotation; }
    function lift(degrees) {
        if (_rotation < degrees) rotation++;
        if (_rotation > degrees) rotation--;
    }
}

onClipEvent(enterFrame) {
    if( ++frameCounter % slowdown == 0){ // добавлено
        lift(degrees);
    }
}
```

ПОБИТОВЫЕ ОПЕРАТОРЫ

Побитовые операторы являются эффективной лаконичной альтернативой отслеживания большого количества двоичных переменных — переменных, которые могут принимать только два значения. При использовании побитовых операторов можно получить SWF-файл меньшего размера. Кроме того, если часто требуется одновременно задавать или получать несколько значений, то использование побитовых операторов, скорее всего, существенно повысит производительность программы.

К сожалению, код, в котором задействованы побитовые операторы, часто бывает труден для чтения. Кроме того, для использования побитовых операторов необходимо хорошо знать двоичную (по основанию 2) арифметику. Побитовые операторы никогда не являются абсолютной необходимостью: того же самого результата всегда можно достичь с помощью логических (булевых) операторов. Поэтому многие программисты избегают использования побитовых операторов. Однако для выполнения некоторых задач использование побитовых операторов будет гораздо более эффективным методом. Вы практически наверняка выиграете, если будете иметь их в своем арсенале средств **ActionScript**.

Основой двоичной арифметики является *бит*. Бит представляет собой единицу информации, которая может иметь только два состояния. Эти состояния можно назвать 0 и 1, вкл. и выкл., истина и ложь, установлено и сброшено или любым другим образом, наиболее подходящим для той или иной задачи.

Двоичная арифметика **ActionScript** основана на 32-битовых двоичных числах, представленных на рис. 13.1. Существуют два способа использования этих чисел.

- Их можно получать и задавать как целые числа, изменяя все 32 бита в одной операции.
- Каждое число можно использовать для представления ³¹ двоичной переменной, которые можно получать или задавать в любой нужной комбинации.



Рис. 13.1. 32-битовое число. Данное число равно 1, поскольку "включен" только один младший разряд

Зачем ограничиваться 31 двоичной переменной, если можно работать с 32 битами? Об этом читайте в разделе "Возможные проблемы" в конце данной главы.

Вы можете по собственному усмотрению переходить от одной моды к другой, используя обычные арифметические операторы для операций одного типа и побитовые операторы для операций другого типа. Именно второй способ использования двоичных чисел (при котором **каждый** разряд представляет отдельную переменную) делает их столь мощным инструментом ActionScript.

Очень краткий курс двоичной арифметики

В десятичной системе счисления (по основанию 10) значение разрядов справа налево возрастает в соответствии с показателем степени числа 10: 1, 10, 100, 1000, 10000 и т.д. Каждый разряд в 10 раз больше того, что находится справа. В двоичной системе счисления значение разрядов возрастает в соответствии с показателем степени числа 2: 1, 2, 4, 8, 16, 32, 64 и т.д. Каждый разряд в 2 раза больше того, что стоит справа.

Число 1 в 32-битовом представлении будет выглядеть так:

```
00000000000000000000000000000001
```

Число 2 будет представляться так:

```
00000000000000000000000000000010
```

Число 4 будет выглядеть так:

```
00000000000000000000000000000100
```

В десятичной системе счисления в каждом разряде может стоять число от 0 до 9. В двоичной системе каждый бит может содержать либо 0, либо 1. Каждая позиция является либо включенной, либо **выключенной**. Если она включена, к итоговому числу добавляется вес позиции. Если позиция отключена, не добавляется ничего.

Рассмотрим следующие примеры:

11 в двоичном представлении равно 3: $1 \times 2 + 1 \times 1$;

100 в двоичном представлении равно 4: $1 \times 4 + 0 \times 2 + 0 \times 1$;

1001 в двоичном представлении равно 9: $1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1$.

На рис. 13.2 показан двоичный счет от 0 до 8 с использованием четырех битов. Черный овал обозначает 1, белый овал — 0. В то же время эти биты можно рассматривать как четыре переключателя **вкл./выкл.**

Исходя из этого, перейдем к рассмотрению побитовых операторов. Их можно разделить на две основные категории. Первую мы будем называть "побитовой логической". (Обычно операторы этой категории называют просто "побитовыми", чтобы не путать их с булевыми логическими операторами.) Другой категорией является "битовый сдвиг".

Кроме того, каждый побитовый оператор, как логический, так и со сдвигом, можно комбинировать с операцией присваивания так же, как это делалось с арифметическими операторами.

ПОБИТОВЫЕ ЛОГИЧЕСКИЕ ОПЕРАТОРЫ

Существуют четыре побитовых логических оператора: AND (&), NOT (-), OR (|) и XOR (^) (И, НЕ, ИЛИ, исключающее ИЛИ соответственно). Оператор "побитовое НЕ" (NOT) выполняет обращение каждого разряда 32-битового двоичного числа. Таким образом, если все разряды числа были установлены в 1, то они будут обращены в 0, и наоборот.

Остальные три побитовых логических оператора сравнивают два 32-битовых двоичных числа, и в результате получается третье 32-битовое двоичное число. Выполняется побитовое сравнение двух исходных чисел, а результат каждого побитового сравнения используется для определения значения соответствующего бита в третьем числе. Чтобы наглядно представить такое сравнение, расположите два исходных числа по вертикали, а результат сравнения — внизу.

ОПЕРАТОР "ПОБИТОВОЕ И"

При использовании оператора "побитовое И" результирующий бит будет равным 1 только в том случае, если оба входных бита являются 1. На рис. 13.3 это продемонстрировано на примере числа, состоящего из четырех бит. На рисунке видно, что в обоих входных числах 1 равняется только наименьший значащий бит, бит 0. Следовательно, в результате только наименьший значащий бит будет установлен в значение 1.

Оператор "побитовое И" можно использовать для того, чтобы проверить, является ли включенным определенный бит или группа битов. Например, в приведенном ниже коде отображено сравнение, показанное на рис. 13.3.

```
x = 13;
y = 1;
result = x & y; // результатом является 1
```

Полученный результат позволяет ответить на вопрос о том, является ли в переменной *x* включенным наименьший значащий бит (бит 0). Если результатом является 1, то наименьший значащий бит включен. Если результатом является 0, то наименьший значащий бит отключен. В данном случае он включен.

Полученный результат можно рассматривать и под другим углом: в качестве ответа на вопрос о том, являются ли включенными биты 0, 2 или 3 в переменной *y*. Те биты, которые являются включенными в полученном результате, будут включены и в переменной *y*. В данном случае это только бит 0.

Если установить переменную *y* = 3, то результат *result* = *x* & *y* отвечает на вопрос о том, какой бит является включенным в переменной *x*: бит 0 или бит 1? В табл. 13.2 представлены возможные результаты и их значения

На заметку

В качестве оператора "побитовое И" используется один символ (&), тогда как для представления оператора "логическое И" используются два символа (&&).

Например, в базе данных домов, выставленных на продажу, биты 0 и 1 могут представлять соответственно гараж и навес для автомобиля. С помощью всего одного запроса вы сможете выяснить, имеет ли искомый дом либо то, либо другое (и если да, то что именно), или же он имеет и то и другое, либо вообще ничего.

С помощью двух битов можно проверить четыре возможные комбинации. С помощью трех битов можно проверить восемь комбинаций. Каждый дополнительный бит удваивает число комбинаций, возможных для проверки. Сколько комбинаций можно проверить с помощью 31 бита? Ответом будет: 2 в степени 31, т.е. 2 147 483 648! (Проверьте этот результат во Flash с помощью операции `trace(Math.pow(2, 31))`.) Совсем неплохо для четырех байт информации.



Рис. 13.2. Двоичный счет от 0 до 8. Четыре разряда можно рассматривать как четыре переключателя

ТАБЛИЦА 13.2. ИСПОЛЬЗОВАНИЕ ОПЕРАТОРА "ПОБИТОВОЕ И" для ОПРЕДЕЛЕНИЯ состояний "Вкл./Выкл."

РЕЗУЛЬТАТ	Бит 1 в x	Бит 0 в x
3	Вкл.	Вкл.
2	Вкл.	Выкл.
1	Выкл.	Вкл.
0	Выкл.	Выкл.

ОПЕРАТОР "ПОБИТОВОЕ ИЛИ"

При использовании оператора "побитовое ИЛИ" результирующий бит будет равен 1, если хотя бы один из входных битов установлен в значение 1, как показано на рис. 13.4.

Оператор "побитовое ИЛИ" используется для включения определенного бита или группы битов. Например, рассмотрим код, описывающий рис. 13.4 :

```
x = 13;  
y = 1;  
result = x | y; // результатом будет 13
```

Нулевой бит в итоговом значении будет включен, независимо от того, какое значение имеет переменная x. В приведенном примере данный бит в переменной x уже включен, поэтому результат будет таким же, как и исходное значение x.

Если биты в операндах x и y представляют собой газетные статьи в базах данных, то предыдущий пример будет давать ответ на вопрос о том, содержится ли хотя бы в одной из баз данных статья, представленная битом 0. Если нулевой бит в результате является включенным, то одна из баз данных содержит необходимую статью. Если нулевой бит отключен, то статьи нет ни в одной из баз данных.

На заметку

В качестве оператора "побитовое И" используется один символ (&), тогда как для представления оператора "логическое И" используются два символа (|&|).

Оператор "побитовое ИЛИ" можно рассматривать с точки зрения того, что он объединяет два набора данных. Например, в примере с двумя базами данных в результате будут скомбинированы x и y.

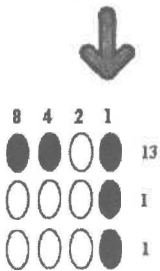


Рис. 13.3. Результирующий бит при использовании оператора "побитовое И" будет равняться 1 только в том случае, если оба исходных бита установлены в значение 1

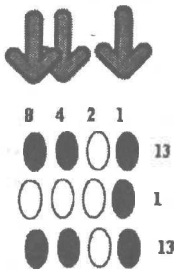


Рис. 13.4. При использовании оператора "побитовое ИЛИ" результирующий бит будет равен 1, если хотя бы один из входных битов установлен в значение 1

ОПЕРАТОР "ПОБИТОВОЕ ИСКЛЮЧАЮЩЕЕ ИЛИ"

Результатом оператора "побитовое исключающее ИЛИ" будет 1 только в том случае, если входные биты различны (рис. 13.5). Для обозначения данного оператора используется символ (^), вставляемый на большинстве клавиатур с помощью комбинации клавиш <Shift+6>.

Оператор "побитовое исключающее ИЛИ" используется для обращения определенного бита или группы битов. Например, рассмотрим код, описывающий рис. 13.5:

```
x = 13;  
y = 1;  
result = x ^ y; // result = 12, поскольку  
                //наименьший значащий бит  
                // переключается из 1 в 0
```

В результате бит 0 будет обращен, независимо от того, какой была переменная x. В данном случае бит 0 в x был включенным, а в результате является выключенным.

Если в программе имеются кнопки включения/выключения, то посредством одного 32-битового двоичного числа можно отслеживать состояние до 31 кнопки и с помощью оператора "побитовое исключающее ИЛИ" одновременно задавать их состояния.

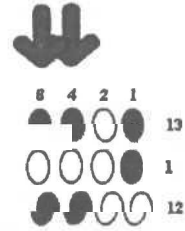


Рис. 13.5. Результатом оператора "побитовое исключающее ИЛИ" будет 1 только в том случае, если входные биты различны

ОПЕРАТОР "ПОБИТОВОЕ НЕ"

Оператор "побитовое НЕ" обращает каждый разряд 32-битового двоичного числа. Если в программе имеются кнопки включения/выключения и их состояния представлены 32-битовым двоичным числом, то с помощью оператора "побитовое исключающее ИЛИ" можно посредством одной операции переключить все кнопки.

ИСПОЛЬЗОВАНИЕ НЕСКОЛЬКИХ ПОБИТОВЫХ ЛОГИЧЕСКИХ ОПЕРАТОРОВ

В одном выражении можно использовать несколько побитовых логических операторов:

```
x = 13;  
y = 1;  
z = 2;  
result = x & (y | z); //result = 1
```

Последняя строка дает ответ на вопрос о том, какой бит в x является включенным: бит 0 или бит 1? Включенным в x будет тот бит, который является включенным в результате.

С теми же значениями x, y и z, что и в предыдущем примере, следующий код дает ответ на вопрос о том, являются ли бит 0 и бит 1 включенными в x.

```
result = (x & (y | z)) == (y | z); //результат ложный
```

В данном случае бит 0 является в x включенным, а бит 1 — нет, поэтому результатом является значение false.

Чтобы понять логику данного выражения, считайте, что $(y | z)$ представляет собой некоторую группу битов; назовем ее d. Тогда $(x \& (y | z))$ является наложением между x и d — биты, которые включены в обоих переменных. Это означает, что в результате будет получен ответ на вопрос, является ли перекрытие между x и d равным d. Если это так, то это аналогично тому, что все биты, включенные в d, являются включенными в x. В рассмотренном случае y установлен в значение 1, т.е. бит 0 в y является включенным. Переменная z равна 2, т.е. бит 1 в z является включенным. Таким образом, в рассматриваемом примере d включает в себя и бит 0, и бит 1. Но x установлен в значение 13, где биты 3, 2 и 0 являются включенными, а бит 1 — нет. Таким образом, перекрывающимся между x и d является только бит 0, который не равен d.

ОПЕРАТОРЫ со сдвигом БИТА

Три оператора со **сдвигом** бита перемещают значение каждого бита на определенное количество позиций вправо или влево. Биты в конце строки отбрасываются и теряются.

Сдвиг бита на одну позицию вправо эквивалентен делению на 2. Сдвиг на один **шаг** влево эквивалентен умножению на 2.

Сдвиг битов аналогичен десятичной системе счисления, в которой сдвиг разряда влево или вправо соответствует делению или умножению на 10. Например, если начать с числа 100 и сдвинуть разряд на одну позицию вправо, получим значение 10. Иными словами, выполняется деление на 10. Последний 0 в числе 100 отбрасывается и теряется.

Аналогичным образом, если начать с двоичного числа 100 (которое равно 4 в десятичной системе счисления) и сдвинуть разряд на одну позицию вправо, то получим двоичное 10 (2 в десятичной системе). В данном случае было выполнено деление на 2. Последний 0 в двоичном числе 100 отбрасывается и теряется.

При использовании операторов со сдвигом бита для деления или умножения целых чисел программа выполняется значительно быстрее, чем с помощью операторов деления (/) или умножения (*). В некоторых случаях разница в скорости составляет 30%. Конечно, операторы со сдвигом бита для умножения или деления можно применять только в том случае, если делитель или множитель являются числом 2 в какой-то степени. Если выполнять сдвиг бита с нечетным операндом, то результат будет округлен в меньшую сторону, поскольку наименьший значащий бит отбрасывается.

СДВИГ ВПРАВО СО ЗНАКОМ

Оператор "сдвиг вправо со знаком" выполняет деление на 2 и отбрасывает остаток. Он сохраняет знак отрицательного числа.

Сохранение знака при сдвиге бита

При задании дополнительного кода отрицательного числа в двоичной системе счисления компьютеры устанавливают старший значащий бит в значение 1. Для положительных чисел старший значащий бит установлен в значение 0.

При сдвиге разряда двоичного числа вправо, биты отбрасываются справа, а слева добавляются новые. Для отрицательных чисел оператор побитового сдвига вправо со знаком устанавливает новые биты в значение 1. Для положительных чисел он устанавливает новые биты в значение 0. Таким образом, знаки чисел сохраняются даже при обращении процесса и выполнении сдвига влево на то же количество разрядов, что и вправо.

Операция по выполнению сдвига вправо со знаком имеет следующий формат:

`result = значение >> количество разрядов`

Рассмотрим приведенные ниже примеры сдвига вправо со знаком.

```
x = 10;
y = 1;
result = x >> y ; // result = 5, т.е., 10 / 2, без остатка
```

```
x = 13;
y = 2;
result = x >> y ; // result = 3, т.е., 13 / 4, округление в
                  // меньшую сторону
```

```
X = -13;
y = 2;
result = x >> y ; // result = -4, т.е., -13 / 4, округление в меньшую
                  //сторону
```

СДВИГ ВПРАВО БЕЗ ЗНАКА

Оператор сдвига вправо без знака смещает разряды вправо. Однако в отличие от оператора сдвига вправо со знаком, он всегда добавляет слева нули. Следовательно, результат всегда будет положительным числом.

Операция по выполнению сдвига вправо без знака имеет следующий формат:

result = значение >>> количество разрядов

Данный оператор удобен при выполнении так называемой "настройки битов". Однако он не имеет очевидных арифметических применений. Для положительных чисел он дает тот же самый результат, что и оператор сдвига вправо со знаком. Для отрицательных чисел значение 1, установленное для старшего бита и сдвинутого вправо, дает результаты, которые не связаны с исходным числом очевидным образом. Например:

```
X = -13;
Y = 1;
result = x >>> y ; // result = 2,147,483,641
```

СДВИГ ВЛЕВО СО ЗНАКОМ

Оператор сдвига влево со знаком выполняет умножение на 2 с сохранением знака отрицательного числа.

Операция по выполнению сдвига влево со знаком имеет следующий формат:

result = значение << количество разрядов

Рассмотрим следующие примеры сдвига влево со знаком:

```
X = 13;
Y = 2;
result = x << y ; // result = 52, т.е. 13 * 4

x = 10;
y = 1;
result = x << y ; // result = 20, т.е. 10 * 2

x = -13;
y = 2;
result = x << y ; // result = -52, т.е. -13 * 4
```

ПРИСВАИВАНИЕ и СЛОЖНОЕ ПРИСВАИВАНИЕ

Оператор присваивания (=) сохраняет результат выражения в переменной, в элементе массива или свойстве объекта. Ниже приведены соответствующие примеры.

```
x = 2; // сохраняет число 2 в переменной x
month[11] = "December"; // 12-м элементом массива месяцев является
```

```
        // "December" (декабрь)
car.color = 0xFF0000; // свойством color (цвет) объекта car
        // (автомобиль) является 0xFF0000 (красный)
```

Десять операторов *сложного присваивания* являются кратким представлением комбинации оператора присваивания с различными арифметическими или побитовыми операторами. Например, оператор "сложения и переназначения" (+=) объединяет присваивание (назначение) и сложение. Каждый оператор сложного присваивания выполняет операцию с двумя операндами и сохраняет результат в левом операнде. Например, выражение `x += 2` означает, что производится сложение 2 их, а результат сохраняется как `x`. Это выражение эквивалентно `x = x + 2`.

В табл. 13.3 перечислены операторы сложного присваивания с примерами и эквивалентными выражениями.

ТАБЛИЦА 13.3. ОПЕРАТОРЫ сложного ПРИСВАИВАНИЯ

Имя	ОПЕРАТОР	ПРИМЕР	ЭКВИВАЛЕНТНОЕ ВЫРАЖЕНИЕ
Сложение и переназначение	<code>+=</code>	<code>i += 2</code>	<code>i = i + 2</code>
Вычитание и переназначение	<code>-=</code>	<code>balance -= debit</code>	<code>balance = balance - debit</code>
Умножение и переназначение	<code>*=</code>	<code>rate *= increase</code>	<code>rate = rate * increase</code>
Деление и переназначение	<code>/=</code>	<code>price /= discount</code>	<code>price = price / discount</code>
Деление по модулю и переназначение	<code>%=</code>	<code>frameCounter % slowdown</code>	<code>frameCounter = frameCounter % slowdown</code>
Сдвиг бита влево и переназначение	<code><<=</code>	<code>answer <<= num</code>	<code>answer = answer << num</code>
Сдвиг бита вправо и переназначение	<code>>>=</code>	<code>result >>= 1</code>	<code>result = result >> 1</code>
Побитовое И и переназначение	<code>&=</code>	<code>test &= 4</code>	<code>test = test & 4</code>
Побитовое исключающее ИЛИ и переназначение	<code>^=</code>	<code>finalMask ^= initialMask</code>	<code>finalMask = finalMask ^ initialMask</code>
Побитовое ИЛИ и переназначение	<code> =</code>	<code>onOff = on</code>	<code>onOff = onOff on</code>

Операторы сложного присваивания не являются более эффективными с точки зрения вычислений по сравнению со своими более длинными эквивалентами. Они также не приводят к уменьшению размеров SWF-файла и используются исключительно для краткости написания.

Отметьте, что выражение

```
x *= y + 2
```

эквивалентно

```
x = x * (y + z),
```


а не выражению

$$x = x * y + z,$$

эквивалентному следующему:

$$x = (x * y) + z.$$

ОПЕРАТОРЫ СРАВНЕНИЯ

Операторы сравнения используются для определения того, являются ли два операнда равными или является ли один операнд больше или меньше другого. Существует восемь операторов сравнения.

- Равенство (==)
- Неравенство (!=)
- Строгое равенство (===)
- Строгое неравенство (!==)
- Меньше (<)
- Меньше или равно (<=)
- Больше (>)
- Больше или равно (>=)

Первые четыре оператора называют операторами *равенства*, а последние четыре — операторами *отношения*. Все восемь операторов сравнивают две строки или числа и возвращают значение `true` (истина) или `false` (ложь), в зависимости от того, является ли отношение, указанное между операндами, точным или нет.

Кроме того, операторы отношения возвращают значение `undefined`, если хотя бы один из операндов имеет значение `NaN` ("Не число"). Операторы равенства и строго равенства в этом случае возвращают значение `false`, а операторы неравенства и строгого неравенства — значение `true`.

ОПЕРАТОРЫ РАВЕНСТВА

Операторы равенства (==) и строгого равенства (===) выполняют проверку на "тождество", но оператор равенства при необходимости перед выполнением сравнения выполняет преобразование типов данных. Для оператора строгого равенства важно, чтобы операнды уже относились к одному типу данных, в противном случае он возвращает значение `false`.

Например, выражение `null == undefined` — истинно, а `null === undefined` — ложно. В связи с тем, что `null` и `undefined` относятся к разным типам данных, они не являются строго равными. Однако оператор равенства преобразует оба их в 0, так что они будут равными.

Операторы неравенства (!=) и строгого неравенства (!==) выполняют операции, противоположные операторам равенства и строгого равенства. Если первые возвращают значение `true`, то последние — `false` (и наоборот).

Оператор равенства, вероятно, является наиболее распространенным оператором и одновременно источником наиболее распространенных ошибок. Дело в том, что очень часто вместо него указывают один знак равенства (оператор присваивания), тогда как нужны два (оператор равенства).

Совет

Подробно об ошибке, связанной с тем, что пользователи путают оператор равенства с оператором присваивания, рассказывается в разделе "Возможные проблемы" в конце этой главы.

Числа сравниваются математически. Например, результатом приведенных ниже выражений будет значение `true`.

```
1 + 2 == 3
-20 < -1
80.5 != 80
```

А результатом этих выражений будет значение `false`:

```
6 <= 5
5 >= 6
-200 > 0
```

Сравнение строк производится с помощью "кодовых точек" набора символов ISO-8859 (Latin-1) или Unicode. Кодовая точка — это число или "код символа", соответствующие каждому символу. Например, результатом приведенных ниже выражений будет значение `true`.

```
"A" < "a" // прописные буквы стоят перед строчными
"A" > "1" // числа стоят перед прописными буквами
"<" <= ">" // знак "меньше" стоит перед
           // знаком "больше"
```

Если вы хотите использовать код символа буквы и сравнить ее с числом, воспользуйтесь функцией `charCodeAt()`. Например, кодом символа пробела является 32. Следовательно, приведенное ниже выражение истинно.

```
" ".charCodeAt(0) == 32 // true, первый (0)
                        // символ является пустым
```

Приведенное ниже выражение является ложным.

```
" " == 32
```

АВТОМАТИЧЕСКОЕ ПРЕОБРАЗОВАНИЕ типов ДАННЫХ для СРАВНЕНИЯ

Выражение `" " == 32` является ложным из-за способа, которым интерпретирующая программа `ActionScript` выполняет автоматическое преобразование типов данных. При работе с элементарными типами данных интерпретирующая программа преобразовывает операнды так, чтобы в результате выполнялось сравнение либо двух строк, либо двух чисел. Помните, что интерпретирующая программа создает временные копии операндов и преобразовывает именно копии. Она не изменяет исходные данные, подлежащие сравнению.

Строка символов преобразовывается в число только в том случае, если буквально расшифровывается как число. Например, приведенное ниже выражение истинно.

```
"32" == 32
```

Совет

Правила преобразования других типов данных в числа представлены в главе 12 "Управление переменными, данными и типами данных".

Во всех остальных случаях строки преобразовываются в значение `NaN`. Если один или оба операнда преобразовываются в значение `NaN`, то результатом сравнения будет `false` для всех операторов сравнения, кроме `!=` и `!==`. Трудное для понимания выражение `NaN == NaN` является ложным, по крайней мере, что касается `ActionScript`. Кроме того, выражение `NaN != NaN` является истинным. Все это делает операторы сравнения интуитивно непонятными в части, касающейся определения того, является ли элемент данных не числом (`NaN`). Не помогает и оператор `typeof` — он просто выдает значение `number`. Чтобы проверить, относится ли значение к типу `NaN`, воспользуйтесь функцией `isNaN()`:

```
X = NaN;
trace(x); // NaN(
trace(x == NaN); // false
trace(typeof x); // number
trace(isNaN(x)); // true
```

Все вышесказанное справедливо для двух элементарных типов данных. Если при операции сравнения хотя бы один из операндов относится к сложному типу данных (объект, массив или функция), то интерпретирующая программа **ActionScript** обратится к методу `valueOf()` операнда. Если метод `valueOf()` вернет элементарное значение, то оно и будет использоваться в операции сравнения. В противном случае результатом сравнения будет значение **false**.

Во Flash 5 метод `valueOf()` использовался при сравнении двух сложных типов данных. Во Flash MX для операторов равенства и неравенства он больше не используется.



Во Flash MX, если *оба* операнда относятся к сложному типу данных, то операции равенства и неравенства выполняются *по ссылке*. Это означает, что операнды будут равными, если они относятся к *одному объекту*.

Например, если `f1` и `f2` являются литералами функции, то выражение `(f1 == f2)` будет истинным только в том случае, если оба литерала относятся к одной и той же функции, как в приведенном ниже примере.

```
myfunc = function (){};
f1 = myFunc;
f2 = myFunc;
trace(func1==func2); // true
```

В приведенном ниже примере `func1()` и `func2()` идентичны по форме, но не являются литералами одной и той же функции.

```
function func1(){}
function func2(){}
trace(func1 == func2); // false
```

Видеоклипы всегда сравнивают по именам их экземпляров, в том числе и во Flash MX. Однако результаты, имеющие какое-то значение, дают только операторы равенства и неравенства. Операторы `>=`, `<=`, `>` и `<` дают значение **undefined**.

При выполнении автоматического преобразования типов данных предпочтение отдается числам. Если один из операндов является числом, а другой — строкой, булевым значением, **null** или **undefined**, то нечисловой операнд преобразовывается в число. Это относится и к нечисловым операндам, возвращаемым функциями `valueOf()`.

БОЛЕЕ НЕ ИСПОЛЗУЮЩИЕСЯ ОПЕРАТОРЫ СРАВНЕНИЯ ВЕРСИИ FLASH 4

Операторы сравнения Flash 4 `eq`, `ne`, `gt`, `le` и `ge` являются эквивалентами операторов `==`, `!=`, `<`, `>`, `<=` и `>=`, за исключением того, что операторы `lt`, `gt`, `le` и `ge` используются только со строками. Начиная с версии Flash 5 операторы Flash 4 исключены, т.е. они поддерживаются, но не рекомендуются к использованию, за исключением случаев, когда файл экспортируется в формат Flash 4.

Бывают ситуации, когда исключенные объекты работают лучше новых. Вместо того чтобы пользоваться устаревшими объектами, можно поступить мудрее и создать собственные новые функции на основе новых операторов, но с учетом компенсации элементов, создававших проблему.

Macromedia перестает поддерживать исключенные операторы и функции, как только это становится возможным. Программы, в которых используется устаревший синтаксис, будут иметь недолгую жизнь.

Использование операторов сравнения в объектах и массивах

По умолчанию операторы `>=`, `<=`, `<` и `>` не дают значащих результатов, если в качестве операндов используются объекты и массивы. Вновь созданные объекты можно протестировать следующим образом:

```
trace(o == o2); // false
trace(o != o2); // true
```

Однако вместе взятые результаты нижеприведенных выражений приводят к заключению о том, что объекты являются равными.

```
trace(o); // [object Object]
trace(o2); // [object Object]
trace(o <= o2); // true
trace(o >= o2); // true
trace(o > o2); // false
trace(o < o2); // false
```

Например, пусть выражение `o >= o2` является истинным (`o` больше или равно `o2`), а `o > o2` — ложным (`o` не просто больше `o2`). Тогда `o` должно быть равным `o2`, не так ли? (Ничего подобного. Через мгновение вы в этом убедитесь.)

Результаты для двух вновь созданных массивов являются похожими, за исключением того, что там, где для объектов отображается `[object Object]`, для массивов не отображается ничего.

Однако эти результаты являются всего лишь свидетельством того, каким образом Flash проверяет выражения, в которых используются операторы `>=` и `<=`. В обоих случаях "за кулисами" Flash для окончательного сравнения фактически используется оператор `<`, а затем полученный ответ обращается. Например, при проверке выражения `o <= o2` Flash спрашивает: "Является ли `o2 < o`?" Если нет, то программа приходит к выводу о том, что выражение `o <= o2` должно быть истинным! При проверке выражения `o >= o2` Flash спрашивает "Является ли `o < o2`?" Если нет, то программа приходит к выводу о том, что выражение `o >= o2` должно быть истинным!

Эти заключения являются действительными в отношении числовых операндов или строк, которые можно упростить до числовых символов кода. Однако при использовании объектов интерпретирующая программа преобразует оба объекта в `NaN`. Результат выражения `(NaN < NaN)` является неопределенным:

```
trace(typeof (NaN < NaN)); // undefined
```

В результатах сравнения интерпретирующая программа `ActionScript` преобразует значение `undefined` в `false`. Таким образом, в обоих случаях сравнения интерпретирующая программа обращает результат и выводит значение `true`!

Ясно, что по умолчанию операторы `>=`, `<=`, `>` и `<` не дают полезных результатов, если операндами являются объекты и массивы. Но для объектов или массивов можно создать методы `valueOf()`, которые возвращают числовые или строковые элементарные значения, которые уже можно использовать с операторами сравнения `>=`, `<=`, `>` и `<` и получать полезные результаты.

Обратите внимание, что в этом смысле объекты и массивы отличаются от функций. Для сравнения двух функций метод `valueOf()` не действует. Он имеет смысл только при сравнении функций с элементарными типами данных.

ЛОГИЧЕСКИЕ (БУЛЕВЫ) ОПЕРАТОРЫ

Логические операторы позволяют принять решение на основе оценки двух или более выражений. Как правило, используются выражения, которые естественно или интуитивно дают булевы значения.

Тремя логическими операторами являются AND (&&), OR (||) и NOT (!).

Логический оператор NOT (НЕ) возвращает булево значение, противоположное одиночному операнду, которому он предшествует. Пусть, например, в рабочем поле имеется видеоклип `myClip`, который является видимым:

```
trace(myClip._visible); // true
trace(!myClip._visible); // false
```

Аналогично, если перед выражением, возвращающим значение `false`, поставить логический оператор NOT, получим значение `true`. Пусть, например, есть функция `checkKillList()`, которая возвращает значение `false`:

```
trace(checkKillList()); // false
trace(!checkKillList()); // true
```

Оператор AND (И) отвечает на вопрос о том, являются ли оба выражения истинными.

Оператор OR (ИЛИ) отвечает на вопрос о том, является ли одно из двух выражений истинным.

Ниже приведены примеры псевдокода, состоящие из двух выражений, которые либо истинны, либо ложны.

Пример 1

```
ЕСЛИ
начальным пунктом является San Francisco
И
местом назначения является Berkeley
ТО
сядьте на поезд Bay Area Rapid Transit
```

Пример 2

```
ЕСЛИ
пользователь щелкает на Cancel
ИЛИ
возникает ошибка
ТО
отменить операцию
```

Пример 3

```
ЕСЛИ
пользователь ВВОДИТ правильный пароль
И
пользователь НЕ в списке "kill"
ТО
разрешить пользователю доступ
```

Пример 4

```
ЕСЛИ
видеокалип НЕ существует
ИЛИ
видеокалип НЕ видим
ТО
сказать пользователю "Извините, этот видеокалип недоступен!"
```

На языке ActionScript приведенные примеры могут выглядеть следующим образом.

Пример 1

```
if ((start == "San Francisco") && (end == "Berkeley") )
{
    wayToGo = "BART";
}
```

Пример 2

```
if ( (input == cancel) || (input == error) ) {
    cancelOperation();
}
```

Пример 3

```
if ( (checkPassword() ) && (!checkKillList() ) ) {
    grantAccess();
}
```

Пример 4

```
if ( (!myClip) || (!myClip._visible) ) {
    returnError(unavailable)
}
```

На заметку

В качестве оператора OR (ИЛИ) используются два символа вертикальной черты. На большинстве клавиатур для ввода такого символа нужно нажать клавишу <Shift> и клавишу с символом обратной косой черты (\).

Логика логических операторов всегда можно продублировать путем использования нескольких операторов if (если). Например, оператор AND (И) эквивалентен двум вложенным выражениям if. Будучи преобразованными в выражения if, примеры 1 и 3 будут выглядеть следующим образом:

```
if (start == "San Francisco") {
    if (end == "Berkeley") {
        wayToGo = "BART";
    }
}

if (checkPassword()) {
    if (!checkKillList()) {
        grantAccess();
    }
}
```

В большинстве случаев использование оператора AND или вложенных выражений if является исключительно делом вкуса программиста. Некоторые программисты предпочитают использовать вложенные выражения if, поскольку в этом случае код является более удобочитаемым. При использовании оператора AND код является более кратким.

Оператор OR можно заменить выражением типа if-else-if. Будучи преобразованным в такое выражение, пример 2 будет выглядеть следующим образом:

```
if (input == cancel) {
    cancelOperation();
}
else if (input == error) {
    cancelOperation();
}
```

В данном случае использование выражения типа **if-else-if** не дает никаких преимуществ. Оператор **OR** является и более лаконичным, и более удобочитаемым.

Совет

Подробная информация о выражениях **if-else-if** приведена в главе 14 "Работа с данными: использование предложений".

Фактически выражения **if** не совсем точно воспроизводят функциональность операторов **AND** и **OR**. Выражения **if** возвращают булевы значения, **true** и **false**, тогда как операторы **AND** и **OR** фактически возвращают один из операндов. Оператор **AND** возвращает второй операнд в случае, если оба операнда истинны, а если один из операндов ложен, то оператор **AND** возвращает именно его. Оператор **OR** возвращает первый операнд, если он является истинным; в противном случае возвращается второй операнд. Операторы **AND** и **OR** обычно используются в выражениях **if**, и выражение **if** преобразовывает возвращенный операнд в левозначение.

Совет

В табл. 124 приведены правила, которым следует интерпретирующая программа **ActionScript** при извлечении булевых значений из операндов в выражениях **AND** и **OR**.

В табл. 13.4 приведены примеры выражений **OR**, их возвращаемые значения, отображаемые с помощью функции **trace()**, и булевы значения, в которые преобразовываются возвращаемые значения.

Обратите внимание на то, что величина, фактически возвращаемая выражением **OR** в каждом примере, идентична тому, что указано с одной из сторон от оператора **OR**. Например, в выражении **["one", "two"] || "hi"** массив **["one", "two"]**, будучи преобразованным в булево значение, принимает значение **true**, т.е. выражение **OR** возвратит массив, который функция **trace()** отобразит как **one, two**.

На практике значения, возвращаемые выражениями **AND** и **OR**, редко используются сами по себе. Обычно они преобразовываются в булевы значения посредством оператора **if**.

ТАБЛИЦА 13.4. ВЫРАЖЕНИЯ **OR**, ВОЗВРАЩАЕМЫЕ ими ЗНАЧЕНИЯ и БУЛЕВЫ ЭКВИВАЛЕНТЫ

ВЫРАЖЕНИЕ OR	TRACE(ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ)	БУЛЕВ ЭКВИВАЛЕНТ ВОЗВРАЩАЕМОГО ЗНАЧЕНИЯ
"ho" "hi"	hi	false
"32" "hi"	32	true
32 "hi"	32	true
Infinity "hi"	Infinity	true
["one", "two"] "hi"	one, two	true
myClip "hi"	_level10.myClip	true
{eyes : "green", age : 32} "hi"	[object Object]	true

УСЛОВНЫЙ ОПЕРАТОР

Условный оператор является слегка оптимизированным способом внедрения логики условного перехода **if-else**. Он отличается тем, что выражение **if** *требует* наличия заявления **else**, в противоположность обычному оператору **if**, для которого заявление **else** обязательным не является.

Условный оператор всегда ссылается и на условие `true`, и на `false`, что делает его *трех-местным* оператором, т.е. оператором, имеющим три аргумента. Поскольку это единственный оператор, всегда имеющий три аргумента, иногда его называют *троичным* оператором. Он имеет следующий формат:

```
условие ? результат_if_true : результат_if_false;
```

Первая часть условного выражения (условие) является проверкой, которая возвращает значение `true` или `false`. Следующие две части являются возможными возвращаемыми значениями. Если условие истинно, то оператор возвращает вторую часть. Если условие ложно, оператор возвращает третью часть. Рассмотрим псевдокод со следующим условием: "Если расстояние больше 1000 миль, то следует лететь; если расстояние не больше 1000 миль, то следует ехать".

Расстояние больше 1000 миль ? лететь : ехать

На языке **ActionScript** этот пример может выглядеть так:

```
distance = 2000;
howtoGo = distance > 1000 ? "fly" : "drive";
trace(howtoGo); // fly
```

Вторая строка примера эквивалентна следующему коду:

```
if (distance > 1000) {
    howtoGo = "fly"
} else {
    howtoGo = "drive"
}
```

В данном примере можно также использовать функции. Таким образом, условный оператор будет использоваться для управления ходом выполнения программы. Например:

```
function fly() {
    trace("fly");
}
function drive() {
    trace("drive");
}
miles = 2000;
miles > 1000 ? fly() : drive(); // отображается "fly"
```

Проверить несколько условий можно с помощью вложенных условных операторов. Например, предположим, вы хотите лететь, если расстояние больше 1000 миль; идти пешком, если расстояние составляет 3 мили или меньше; или ехать на машине во всех остальных случаях. В этом случае в предыдущую программу можно добавить функцию `walk()` (идти пешком):

```
function walk() {
    trace("walk");
}
```

Все это можно реализовать следующими строками кода:

```
Спросить: "расстояние больше 1000 миль?"
Если да, то лететь.
Если нет, то спросить: "расстояние больше 3 миль?"
Если да, то ехать.
Если нет, то идти пешком.
```


В приведенном ниже примере расстояние не превышает 3 мили, поэтому выполняется функция `walk()`.

```
miles = 3;  
miles > 1000 ? fly() : miles > 3 ? drive() : walk(); // отображается "walk"
```

В данном случае условный оператор выполняется быстрее и результирующий размер SWF-файла будет меньшим (хотя и ненамного), по сравнению со структурой `if/else`.

В приведенном ниже примере оператор `if-else` был бы немного проще и быстрее, но условный оператор является более удобочитаемым.

```
(day == "Monday") ? trace("is fair of face") :  
(day == "Tuesday") ? trace("is full of grace") :  
(day == "Wednesday") ? trace("is full of woe") :  
(day == "Thursday") ? trace("has far to go") :  
(day == "Friday") ? trace("is loving and giving") :  
(day == "Saturday") ? trace("works hard for a living") :  
(day == "Sunday") ? trace("is bonny and blithe and good and gay.") : null;
```

Обратите внимание на то, что в конце концов используется выражение `null`. Условный оператор всегда должен возвращать какую-то величину. Поэтому можно либо использовать что-то, подобное `null`, либо в качестве действия по умолчанию выбрать одно из действий `trace`. Например, чтобы сделать последнюю функцию `trace` используемой по умолчанию, изменим последнюю строку следующим образом:

```
trace("is bonny and blithe and good and gay.");
```

Использование последней функции `trace` по умолчанию приведет в тому, что переменная `day` в качестве значения всегда будет иметь действительный день недели, т.е. если днем не является никакое из значений, начиная с "Monday" (понедельник) и заканчивая "Saturday" (суббота), то им должно быть "Sunday" (воскресенье).

Совет

Сравнение функций, выполняемых условным оператором, с идентичными функциями, реализуемыми с помощью оператора `if-else`, приведено в главе 14 "Работа с данными: использование предложений".

ОПЕРАТОР "ЗАПЯТАЯ"

Оператор "запятая" используется, в первую очередь, для объявления двух и более переменных в одном выражении, как в приведенном ниже примере.

```
var x=1, y=2, z=3;
```

Это эквивалентно следующему:

```
var x=1;  
var y=2;  
var z=3;
```

Данный оператор пригодится в случае необходимости инициализации двух и более индексных переменных в цикле `for`, как в следующем примере:

```
for (i = 0, j = 0, k = 0; i < 50; i++, j -= i, k--) {  
    trace(i+" "+j+" "+k);  
}
```

Результат будет следующим:

```
0 0 -10
1 -1 -11
2 -3 -12
3 -6 -13
* * *
```

Совет

Более подробная информация о циклах приведена в главе 14 "Работа с данными: использование предложений".

Значение, возвращаемое оператором "запятая" (значение последнего операнда), в целом полезным не является. Например, при задании цикла `for` в средней (проверяемой) части выражения присутствие нескольких разделенных запятыми элементов будет совершенно бесполезным (хотя и разрешенным), поскольку оцениваться будет только последняя величина. Так, в приведенном ниже примере в цикле `for` выполняется та же функция, что и в предыдущем примере, так как часть `j >= k, k < i` игнорируется, а в каждом цикле проверяется только часть `i < 50`.

```
for (i = 0, j = 0, k = -10; j >= k, k < i, i < 50; i++, j -= i, k--) {
    trace(i+" "+j+" "+k);
}
```

ИМЕНОВАННЫЕ ОПЕРАТОРЫ

Обращение к некоторым операторам осуществляется по их именам, а не по символам. Это операторы `new`, `typeof`, `instanceof`, `delete` и `void`.

ОПЕРАТОР NEW

Оператор `new` используется с функцией конструктора для создания элемента данных типа `object` (объект) или `array` (массив). Так, в приведенных ниже примерах функциями конструктора являются `Object()`, `Array()` и `Date()`.

```
myObj = new Object();
myArray = new Array();
myDate = new Date();
```

Совет

Знакомство с оператором `new` состоялось в главе 12 "Управление переменными, данными и типами данных". Более подробно он описывается в главе 15 "Объединение предложений в функции".

ОПЕРАТОР TYPEOF

Оператор `typeof` возвращает строку, указывающую тип данных операнда. Его формат следующий:

`typeof выражение`,

где выражение — любое разрешенное выражение.

Совет

Оператор `typeof` представлен и подробно описан в главе 12 "Управление переменными, данными и типами данных".

ОПЕРАТОР INSTANCEOF



Оператор `instanceof` определяет, принадлежит ли объект классу. Его формат следующий:

объект instanceof класс

В **ActionScript** класс реализуется в функции конструктора. Следовательно, операндом *класс* в правой части должно быть имя функции конструктора. Оператор `instanceof` следует цепочке наследования, чтобы определить, наследует ли объект свойства того или иного класса. Так, в приведенном ниже примере `a` является экземпляром `Object`, поскольку `a` является массивом, т.е. объектом, а все объекты являются наследуемыми из `Object`. С другой стороны, `a` не является экземпляром `_global`, так как `_global` не является функцией конструктора.

```
a = new Array();  
trace(a instanceof Array); // true  
trace(a instanceof Object); // true — это справедливо для всех объек-  
тов  
trace(a instanceof _global); // false — _global не является функцией  
конструктора
```

В следующем примере `6` не является экземпляром `Number`, поскольку `6` — это примитивный элемент данных, а не объект. С другой стороны, `n` является объектом, значение которого равно `6`, и `n` является экземпляром `Number`.

```
trace(6 instanceof Number); // false - 6 is not an object  
n = new Number(6);  
trace(n instanceof Number); // true
```

ОПЕРАТОР DELETE

Оператор `delete` пытается удалить переменную, объект (в том числе и функцию), свойство объекта, массив или элемент массива. Формат его простой:

delete идентификатор

Приведем пример использования оператора:

```
myVar = "three"; // создать переменную  
delete myVar; // удалить ее
```

Оператор `delete` возвращает значение `true` или `false`, в зависимости от того, было ли удаление успешным. В приведенном ниже примере оператор возвращает значение `false` при попытке удалить свойство второй раз.

```
myObj = {a: "one", b: "two"} // создать объект  
trace(delete myObj.a); // true  
trace(delete myObj.a); // false - уже удалено
```

ОПЕРАТОР VOID

Я не смог найти ни одного примера из **ActionScript**, в котором фактически бы использовался оператор `void`. Вероятно, его наличие объясняется исключительно вопросами совместимости со стандартом **ECMA**. Оператор имеет следующий формат:

void (выражение)

В результате его применения интерпретирующая программа отбрасывает результаты выражения и возвращает значение `undefined`.

В словаре **ActionScript** (**ActionScript Dictionary**) говорится о том, что оператор `void` часто используется в сравнениях, реализуемых посредством оператора `==`, для тестирования неопределенных значений. Я не могу привести никаких примеров из реальной жизни, однако приведенный ниже код работает.

```
a = undefined;
b = 1;
trace(void(b) == a); // true
```

Тем не менее проще воспользоваться следующей схемой:

```
a = undefined;
trace(undefined == a); // true
```

ДРУГИЕ ОПЕРАТОРЫ

Применение остальных операторов, представленных в главе 11 "Знакомство с ActionScript" и в главе 12 "Управление переменными, данными и типами данных", будет широко продемонстрировано в следующих главах. Здесь они упоминаются только для полноты картины и быстрого ознакомления.

ДОСТУП К СВОЙСТВАМ ОБЪЕКТА: ОПЕРАТОР "ТОЧКА"

Оператор "точка" (`.`) используется для обозначения свойства объекта или вложенного видеоклипа. Предположим, свойство объекта устанавливается в следующее значение:

```
myObject.myProperty = "myValue"; // задание свойства значения строкового типа
```

Ниже переменная `clipVar` устанавливается равной имени экземпляра видеоклипа.

```
clipVar = myClipParent.myClipChild; // устанавливает переменную равной
//имени видеоклипа
```

Совет

Оператор "точка" был представлен в главе 12 "Управление переменными, данными и типами данных". Подробно он описывается в главе 19 "Использование встроенных базовых объектов" и в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

ОПЕРАТОР "МАССИВ-ЭЛЕМЕНТ/ОБЪЕКТ-СВОЙСТВО"

Квадратные скобки могут обозначать элемент массива либо свойство объекта. Ниже приведен пример элемента массива.

```
months[0] = "January";
```

Совет

Подробно о массивах рассказывается в главе 19 "Использование встроенных базовых объектов".

Ниже приведен пример свойства объекта.

```
computer["display"] = "SVG";
```

Совет

Подробная информация об объектах приведена в главе 19 "Использование встроенных базовых объектов" и в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

ОПЕРАТОР "КРУГЛЫЕ СКОБКИ/ОБРАЩЕНИЕ к ФУНКЦИИ"

Помимо использования круглых скобок для группирования других операторов, их можно применить как оператор обращения к функции для ее активизации.

Совет

Информация об использовании круглых скобок для группирования операторов приведена выше, в разделе "Старшинство, ассоциативность и группирование операторов".

Формат обращения, или активизации, функции следующий:

идентификатор (*список*)

где *идентификатор* — имя функции, а *список* — необязательный список разделенных запятыми *аргументов*, или параметров, передаваемых функции. Оператором являются круглые скобки, в которые заключены аргументы.

Совет

Правила образования идентификаторов описываются в главе 12 "Управление переменными, данными и типами данных".

Совет

Процедура объявления или создания функций и получения их значений с помощью оператора возврата описывается в главе 15 "Объединение предложений в функции".

Совет

Подробная информация об операторе возврата приведена в главе 14 "Работа с данными: использование предложений".

Совет

Подробная информация о функциях приведена в главе 15 "Объединение предложений в функции".

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Если операнды не являются равными, то почему они тестируются как одинаковые?

Одной из наиболее распространенных ошибок в **ActionScript** является использование оператора присваивания (=) в случаях, когда необходимо использовать оператор равенства (==). Оператор присваивания задает переменную, элемент массива или свойство **объекта** равным какому-то значению. Оператор равенства проверяет, являются ли два выражения тождественными. В данном случае знакомство с арифметической системой обозначений работает против вас. Очень часто вы пишете `if (x != 10)` — выражение, которое интерпретирующая программа **ActionScript** всегда трактует как истинное, тогда как на самом деле имеется в виду `if (x == 10)` — выражение, которое является истинным только тогда, когда переменная `x` равна 10.

Может ли использование 32 битов в побитовых операциях вызвать проблемы ?

Да! Самый старший бит, находящийся слева в 32-битовых двоичных числах, в побитовых операциях лучше не использовать. Компьютеры используют этот разряд для представления отрицательных чисел (как дополнительный код числа), и использование данного бита может привести к неправильным результатам, особенно на платформах **Macintosh**.

FLASH ЗА РАБОТОЙ: ПРИМЕНЕНИЕ ПОБИТОВЫХ ОПЕРАТОРОВ К ИГРЕ В КРЕСТИКИ-НОЛИКИ

Побитовые операторы могут изменить скорость игры в крестики-нолики, поскольку их можно использовать **для** повторяющихся операций.

Каждый раз, когда игрок делает ход, необходимо хотя бы раз проверить каждую выигрышную комбинацию, чтобы определить, создал ли только что игрок какую-то из них. Если вы хотите, чтобы компьютер играл умно, необходимо также определить, сможет ли игрок создать одну из выигрышных комбинаций при следующем ходе. Побитовые операторы позволяют выполнить эти операции очень быстро.

С помощью девяти бит, представляющих девять клеток игрового поля, можно в виде всего одного целого числа записать, **где** размещены крестики. Аналогичным образом, в виде одного целого числа записывается положение всех ноликов. Когда необходимо сравнить эти положения с выигрышными комбинациями (чтобы проверить, выиграл ли один из игроков), каждую выигрышную комбинацию также можно представить в виде одного целого числа.

Несмотря на то что в данном случае используются обычные 32-битовые двоичные целые числа ActionScript, задействованными будут только девять первых битов, поскольку остальные просто не нужны.

В приведенном ниже примере выигрышные комбинации игры в крестики-нолики помечены символом X.

```
XXX 000 000 X00 0X0 00X X00 00X
000 XXX 000 X00 0X0 00X 0X0 0X0
000 000 XXX X00 0X0 00X 00X X00
```

Каждая позиция трактуется как бит, начиная с 0 в левом верхнем углу. Таким образом, биты с 0 по 8 соотносятся с полем игры в крестики-нолики следующим образом:

```
012
345
678
```

В табл. 13.5 показаны выигрышные позиции в виде десятичных (основание 10), двоичных (основание 2) и восьмеричных (основание 8) чисел. Обратите внимание на то, как каждый восьмеричный разряд представлен тремя двоичными разрядами. Каждая группа из трех двоичных разрядов, в свою очередь, подобна строке поля для игры в крестики-нолики.

ТАБЛИЦА 13.5. ВЫИГРЫШНЫЕ позиции в ИГРЕ в КРЕСТИКИ-НОЛИКИ

ДЕСЯТИЧНЫЕ	ДВОИЧНЫЕ	ВОСЬМЕРИЧНЫЕ
7	000 000 111	7
56	000 111 000	70
448	111 000 000	700
73	001 001 001	111
146	010 010 010	222
292	100 100 100	444
84	001 010 100	124
273	100 010 001	421

В листинге 13.1 приведен пример фильма `tictactoe fla` (упрощенный вариант файла `охо fla` Ральфа Бокельберга (**Ralf Bokelberg**), расположенного на компакт-диске). Здесь показано, как можно использовать массивы чисел, представляющих восемь выигрышных комбинаций, с побитовыми операторами. В файле `tictactoe fla` компьютер играет за обоих игроков, выбирая свободные клетки случайным образом. В файле `охо fla` компьютер играет за одного игрока, а за другого играет пользователь. Компьютер думает наперед и блокирует игрока, если следующим ходом он может составить выигрышную комбинацию.

Листинг 13.1. ПРИМЕР ФИЛЬМА TICTACTOE.FLA

// abridged and adapted from oxo by Ralf Bokelberg, www.QLOD.com (c) 2002

```
onClipEvent (load) {
    squares = new Array ( _root.s0, _root.s1, _root.s2,
        _root.s3, _root.s4, _root.s5,
        _root.s6, _root.s7, _root.s8 );
    function getFreeSquare () {
        do {
            i=Math.random();
        } while (i == 1);
        i = Math.floor(i * 9);
        if (squares[i] != null) return i;
        else {
            for(var j = 0 ; j < 9; j++){
                i++;
                if (i==9) i = 0;
                if (squares[i] != null) return i;
            }
        }
    }
    function omove(num) {
        squares[num].attachMovie("o", "o", (num*2)+20);
        o |= Math.pow ( 2, num );
        squares[num] = null;
    }
    function xmove(num) {
        squares[num].attachMovie("x", "x", num+1);
        x |= Math.pow ( 2, num );
        squares[num] = null;
    }
    var whoseMove = true;
    function move (num) {
        if (whoseMove) xmove(num);
        else omove (num);
        whoseMove = !whoseMove;
    }
    wins = new Array ( parseInt ("0111"),
        parseInt ("0222"),
        parseInt ("0444"),
        parseInt ("07"),
        parseInt ("070"),
        parseInt ("0700"),
        parseInt ("0124"),
        parseInt ("0421"));
    var o = 0;
    var x = 0;
```

```

function checkResult(){
    for(var i = 0; i < 8; i++){
        if((o & wins[i]) == wins[i]){
            trace("o wins :"+o);
            j = 8; // stop the game
            return;
        }
        if((x & wins[i]) == wins[i]){
            trace("x wins :"+x);
            j = 8;
            return;
        }
    }
}

var j = -1;

}
onClipEvent (enterFrame) {
    j++;
    if (j < 9) {
        num=getFreeSquare();
        move(num);
        checkResult();
    }
}

```


РАБОТА с данными: ИСПОЛЬЗОВАНИЕ ПРЕДЛОЖЕНИЙ

В этой главе...

Образование автономных предложений и управляющих блоков	279
Управление ходом выполнения программы	283
Принятие решений с помощью условных предложений и ключевого слова <code>switch</code>	284
Повторение действий с помощью циклов	287
Возможные проблемы	290
Flash за работой: использование цикла <code>for</code> для создания списка	290

ОБРАЗОВАНИЕ АВТОНОМНЫХ ПРЕДЛОЖЕНИЙ И УПРАВЛЯЮЩИХ БЛОКОВ

Предложение является "фразой" программирования, т.е. наименьшим блоком кода, который сообщает команду выполнения интерпретирующей программе **ActionScript** (которая считывает и исполняет код **ActionScript**). Следовательно, для того чтобы выполнить ту или иную задачу, необходимо использовать предложения.

С точки зрения синтаксиса существуют два основных типа предложений: автономные предложения и управляющие блоки. Управляющий блок — это предложение, содержащее другие предложения.

Существуют два типа управляющих блоков: циклический (или *итеративный*) и нециклический. Циклические управляющие блоки (`while`, `do-while`, `for` и `for-in`) повторяют содержащиеся в них предложения до тех пор, пока не будет соблюдено заданное условие. Нециклические управляющие блоки (`if-else`, `ifFrameLoaded` и `with`) исполняют содержащиеся в них предложения только один раз.

ОБРАЗОВАНИЕ АВТОНОМНЫХ ПРЕДЛОЖЕНИЙ

Автономное предложение состоит из

- вспомогательного ключевого слова, одного из тех, которые зарезервированы ActionScript для собственного использования;
- нулевого или большего количества выражений и операторов;
- точки с запятой.

Самым простым автономным предложением является *пустое* предложение, состоящее только из точки с запятой:

```
;
```

Само по себе пустое предложение ничего не значит. Однако действие *оценки*, доступное при работе на панели Actions (Действие) в режиме Normal (Обычный), создает пустое предложение в качестве заполнителя, в который можно вводить необходимое выражение, подлежащее оценке, в том числе и пользовательскую функцию, к которой следует обратиться.

Теоретически автономное предложение может состоять только из выражения и точки с запятой, хотя такое предложение не будет выполнять никакой задачи.

```
"hi"; // ничего не выполняет
```

Чтобы быть полезным, предложение должно что-то изменять (или вызывать функцию, которая что-то изменяет). Зачастую изменения вызывает оператор, как, например, при задании переменной с помощью оператора присваивания:

```
heSays = "hi";
```

Часто выполнение действия определяется и ключевым словом. Автономное предложение, работающее на основе ключевого слова, начинается с одного ключевого слова и заканчивается точкой с запятой. В двух случаях, **break** и **continue**, в состав предложения больше ничего не входит:

```
break;  
continue;
```

Предложение **break** разрывает (отменяет) текущий циклический управляющий блок, тогда как предложение **continue** повторно запускает его.

При использовании предложения **call**, которое исполняет сценарий в кадре, за ключевым словом следует заключенное в круглые скобки выражение, которое преобразуется в номер кадра. Так, в приведенном ниже примере выражение представляет номер буквально.

```
call(4);
```

Предложение **return**, которое указывает на выход из функции, может опционально сопровождаться выражением. Интерпретирующая программа преобразует выражение в значение, возвращаемое функцией. Таким образом, первое из приведенных ниже предложений просто является выходом из функции, а второе является выходом из функции и при этом возвращает значение `myValue`:

```
return;  
return myValue;
```

Предложение **set**, которое присваивает значение переменной, сопровождается заключенной в круглые скобки и разделенной запятой парой параметров: имени переменной и выражения. Интерпретирующая программа преобразует выражение в значение и присваивает это значение переменной. Например, следующее предложение назначает значение 6 переменной `myNumber`:

```
set(myNumber, 6);
```

В предложении `var`, которое объявляет и может инициализировать переменную, ключевое слово *сопровождается* именем переменной и опционально оператором присваивания и исходным значением переменной. Так, в приведенном ниже примере первое предложение объявляет переменную без ее инициализации, а второе — объявляет переменную и инициализирует ее:

```
var myVar;
var MeVar = myValue;
```

ОБЪЕДИНЕНИЕ ПРЕДЛОЖЕНИЙ в УПРАВЛЯЮЩИЕ БЛОКИ

Обычный управляющий блок состоит из

- ключевого слова;
- заключенной в круглые скобки управляющей структуры;
- тела, состоящего из нулевого или большего количества предложений.

Ниже приведен пример типичного управляющего блока.

```
if (allDone) quit;
```

В предыдущем предложении `if` является ключевым словом, `(allDone)` — управляющей структурой, а `quit` — телом. *Ключевое слово* определяет тип предложения, *управляющая структура* указывает, при каких условиях исполняется тело, а *тело* назначает, какие действия выполняет управляющий блок.

Если в теле содержатся два или более предложений, они должны быть заключены в фигурные скобки. Ниже приведены четыре основных формата управляющего блока.

Формат 1

```
ключевое_слово ( выражение ) предложение;
```

Формат 2

```
ключевое_слово ( выражение ) {
    предложение;
}
```

Формат 3

```
ключевое_слово ( выражение ) { предложение1; предложение2; }
```

Формат 4

```
ключевое_слово ( выражение ) {
    предложение1;
    предложение2;
}
```

Предложение1 и *предложение2* представляют собой список любой длины.

Третий и четвертый форматы выполняют одни и те же задачи, но четвертый является более удобочитаемым и представляет собой стандартный формат с использованием нескольких предложений. Выбор первого или второго формата целиком зависит от предпочтений про-

граммиста. При определении форматов управляющих блоков в данной книге используется четвертый формат.

Предложения `do-while`, `for-in` и `if-else` содержат два ключевых слова. В предложениях `do-while` и `if-else` второе ключевое слово стоит за телом. В предложении `for-in` второе ключевое слово содержится в управляющей структуре.

В табл. 14.1, содержащей список ключевых слов, используемых в предложениях, приведен каждый из применяемых форматов.

Вместе ключевое слово и управляющая структура образуют *заголовок* блока. В приведенном ниже примере показаны числа от 0 до 49.

```
for (i = 0; i < 50; i++) {
    trace(i);
}
```

Здесь `for` является ключевым словом, `(i = 0; i < 50; i++)` — управляющей структурой, а `trace(i);` — одним предложением в теле. Следовательно, `for (i = 0; i < 50; i++)` является заголовком.

ТАБЛИЦА 14.1. КЛЮЧЕВЫЕ СЛОВА ПРЕДЛОЖЕНИЙ ACTIONSCRIPT

КЛЮЧЕВОЕ СЛОВО (СЛОВА)	ФОРМАТ	ОПИСАНИЕ	УПРАВЛЕНИЕ ХОДОМ	ГЛАВА
<code>break</code>	<code>break;</code>	Отменяет текущий цикл	Цикл	14
<code>call</code>	<code>call(кадр);</code>	Выполняет сценарий в другом кадре	Обращение	14
<code>continue</code>	<code>continue;</code>	Перезапускает текущий цикл	Цикл	14
<code>do-while</code>	<code>do</code> <code>{ предложения }</code> <code>while</code> <code>(выражение)</code>	Повторяет предложения, пока <i>выражение</i> истинно	Цикл	14
	<code>i</code>	Резервирует место для предложения в режиме Normal	По умолчанию	14
<code>for</code>	<code>for (init;</code> <code>test; next)</code> <code>{ предложения }</code>	Повторяет предложения, пока <code>test</code> является ложным	Цикл	14
<code>for-in</code>	<code>for (свойство в</code> <code>объекте)</code> <code>{ предложения }</code>	Перечисляет свойства <i>объекта</i>	Цикл	19-21
<code>function</code>	<code>function имя</code> <code>(параметры)</code> <code>{ предложения }</code>	Объявляет функцию	Обращение	15
<code>if/else -</code> <code>if/else</code>	<code>if (условие1)</code> <code>{ предложения }</code> <code>} else if</code> <code>(условие2) {</code> <code>предложения }</code> <code>else { предло-</code> <code>жения }</code>	Выполняет предложения на основе одного или нескольких условий	Условный	14
<code>ifFrameLoad</code> <code>ed</code>	<code>ifFrameLoaded</code> <code>(кадр) { пред-</code> <code>ложения }</code>	Выполняет предложения, если кадр был загружен	Условный	14

КЛЮЧЕВОЕ СЛОВО (СЛОВА)	ФОРМАТ	ОПИСАНИЕ	УПРАВЛЕНИЕ ходом	ГЛАВА
return	return; return <i>выражение;</i>	Выходит из функции и возвращает значение	Обращение	15
set	set <i>(переменная, значение)</i>	Присваивает значение переменной с динамическим именем	По умолчанию	12, 14
switch/case/default	switch(выражение) { <i>case значение : блок</i> default : <i>блок</i> }	Выполняет предложение(я), начиная с блока, в котором значение совпадает с <i>выражением</i> (если совпадений нет, выполняется <i>default</i>)	Условный	14
var	var <i>имя_переменной</i> ; var <i>имя_переменной</i> - выражение;	Объявляет переменную с возможным присвоением значения	По умолчанию	15
while	while <i>(выражение) {</i> <i>предложения</i> }	Повторяет предложения, пока <i>выражение</i> истинно	Цикл	14
with	with <i>(имя_объекта)</i> { <i>предложения</i> }	Выполняет предложения в контексте заданного объекта	Цикл	19-21

УПРАВЛЕНИЕ ходом ВЫПОЛНЕНИЯ ПРОГРАММЫ

По умолчанию интерпретирующая программа *ActionScript* выполняет предложения последовательно, сверху вниз. Однако большинство предложений может изменить ход выполнения программы, заданный по умолчанию. Исключениями являются пустое предложение, а также *set* и *var*, которые не влияют на ход выполнения программы.

В *ActionScript* имеются три типа управления ходом выполнения программы: *условный*, *циклический* и *вызов*. Далее в этой главе основное внимание уделено рассмотрению условного и циклического управления.

Принцип управления ходом выполнения программы *вызов* представлен предложениями *call* и *function*. При этой схеме предложение инициирует вызов, который прерывает выполнение интерпретирующей программой последовательной обработки. Интерпретирующая программа исполняет блок кода в каком-то другом месте программы, а затем возвращается и продолжает обработку с того места, в котором она была прервана.

Совет

Предложение *call* описано выше, в разделе "Образование автономных предложений".

Совет

Знакомство с функциями состоялось в главе 11 "Знакомство с *ActionScript*". Подробнее они рассматриваются в главе 15 "Объединение предложений в функции".

ПРИНЯТИЕ РЕШЕНИЙ С ПОМОЩЬЮ УСЛОВНЫХ ПРЕДЛОЖЕНИЙ И КЛЮЧЕВОГО СЛОВА SWITCH

Когда интерпретирующая программа **ActionScript** сталкивается с условным предложением, она оценивает условие в управляющей структуре и, если оно истинно, исполняет предложения в теле.

Существуют два типа условных предложений: **if-else** и **ifFrameLoaded**. Предложение **if-else** можно разбить на простые предложения **if** и **else**.

ПРЕДЛОЖЕНИЕ IF

Самым простым условным предложением является **if**. Его синтаксис очень простой:

```
if (выражение) {  
    предложение1;  
    предложение2;  
}
```

Интерпретирующая программа преобразует *выражение* в булево значение. Если *выражение* преобразуется в значение **true**, то предложения в теле исполняются. В противном случае они не исполняются.

Совет

Правила преобразования предложений в булевы значения приведены в табл. 12.4.

Обратите внимание на то, что любой массив, объект или видеоклип преобразуются в значение **true**. Следовательно, оператор **if** можно использовать для проверки существования (или несуществования) массива, объекта или видеоклипа. Например:

```
// если видеоклип не существует, обрабатывать условие ошибки  
if (!myClip) {  
    // здесь указывается код для обработки условия ошибки  
}  
// если массив существует, задать переменную len равной длине массива  
if (myArray) {  
    len = myArray.length;  
}
```

ПРЕДЛОЖЕНИЕ IF-ELSE

Предложение **if-else** фактически представляет собой два отдельных, но связанных предложения. Предложение **if** сообщает интерпретирующей программе, что нужно делать, если выражение истинно. Предложение **else** сообщает интерпретирующей программе, что нужно делать, если выражение ложно. Так, в приведенном ниже примере, если массив **myArray** уже существует, программа устанавливает переменную **len** в значение длины массива. Если массив не существует, программа сначала создает его, а затем задает переменную.

```
if (myArray) {  
    len = myArray.length;  
} else {  
    myArray = new Array();  
    len = myArray.length;  
}
```

Для знакомства с предложениями `if-else` обратитесь к главе 13 "Использование операторов". Кроме того, важная информация о комментариях и предложениях `if-else` содержится в главе 11 "Знакомство с ActionScript".

ВЛОЖЕНИЕ УСЛОВНЫХ ПРЕДЛОЖЕНИЙ

Условные предложения можно вкладывать друг в друга на любую нужную глубину, что позволяет последовательно проверить несколько различных условий. Например:

```
if (day == "Monday") {
    trace("is fair of face");
} else if (day == "Tuesday") {
    trace("is full of grace");
} else if (day == "Wednesday") {
    trace("is full of woe");
}
```

Чтобы сделать последнее действие `trace` заданным по умолчанию, замените последние три строки следующей:

```
} else trace("is full of woe");
```

Такое же вложение можно выполнить с помощью условного оператора. Обратитесь за подробными сведениями к главе 13 "Использование операторов".

ПРЕДЛОЖЕНИЕ SWITCH/CASE



В предыдущем примере в каждом условном предложении выполняется проверка на равенство переменной `day`. Такой же тип вложенной условной логики можно представить с помощью предложения `switch`:

```
switch (day) {
    case "Monday":
        trace("is fair of face");
        break
    case "Tuesday":
        trace("is full of grace");
        break
    case "Wednesday":
        trace("is full of woe");
}
```

Интерпретирующая программа тестирует то же самое выражение (в данном случае переменную `day`) на соответствие каждому набору выражений (в нашем примере это строковые литералы "Monday", "Tuesday" и "Wednesday"). При нахождении совпадения она начинает выполнять предложения, начиная с того, которое следует за совпавшим значением.

Если интерпретирующая программа не находит совпадений, она ищет предложение `default`. Например, чтобы сделать последнее действие `trace` заданным по умолчанию, заменим последние три строки на следующее выражение:

```
    default:
        trace("is full of woe");
}
```

Предложение `default` не обязательно должно быть последним в теле предложения `switch`, хотя оно является таковым практически всегда.

В отличие от вложенных условных предложений, рассмотренных выше, предложение `switch` для сравнения значений использует оператор строгого равенства (`===`).

Предложение `switch` используется для удобства обозначений и *не* выполняется быстрее вложенных условных предложений. В байтовом коде SWF-файла они выглядят совершенно одинаково. Это позволяет достичь обратной совместимости фильмов с версией Macromedia Flash 4. Однако в SWF-файлах версии Flash 4 используются операторы равенства (`==`), тогда как в версиях Flash 5 и Flash MX — операторы строгого равенства (`===`). Но это является неизбежным, поскольку Flash 4 не может определять типы данных. Проверка типов данных является главной функцией определения строгого равенства.

Совет

Подробная информация о строгом равенстве приведена в главе 13 "Использование операторов".

Значениями предложений `test (day)` и `case ("Monday", "Tuesday" и "Wednesday")` могут быть любые действительные выражения. Это означает, что ими могут быть массивы, объекты, функции или сложные математические формулы, но при условии их преобразования в одно значение. Однако, как правило, это строковые или числовые литералы.

Обратите внимание на оператор `break` в каждом предложении `case`. Каждое такое предложение указывает начальную точку **исполнения** кода. Оно *не* указывает конечную точку. Операторы `break` сообщают интерпретирующей программе о необходимости выхода из предложения `switch` после исполнения предложений, связанных с отдельным случаем. Если несколько случаев должны выполняться один за другим, операторы `break` могут быть опущены.

Совет

Если оператор `case` надлежащим образом не выполняется, обратитесь к разделу "Возможные проблемы" в конце этой главы.

ПРЕДЛОЖЕНИЕ `IFFRAMELOADED`

Предложение `ifFrameLoaded` является исключенным (хотя пока и поддерживается), но до сих пор широко применяется. Предложение `ifFrameLoaded` обеспечивает совместимость с версией Flash 3. Рекомендованная замена, `_frameLoaded`, требует наличия версии не ниже Flash 4.

Предложение `ifFrameLoaded` является специальным случаем предложения `if`, которое проверяет, загрузился ли уже определенный кадр (в определенной сцене). Оно используется при предварительной загрузке видеоклипов.

Например, если приведенный ниже код наложить на первые два кадра видеоклипа, то они будут циклически выполняться до тех пор, пока не загрузится кадр 60. Затем видеоклип начнет воспроизводиться с кадра 5.

```
//кадр1:
    ifFrameLoaded (60) {
        gotoAndPlay (5);
    }
//кадр2:
gotoAndPlay (1);
```

Рекомендуемая замена для `if FrameLoaded (60)` проста:

```
if (_framesLoaded == 60)
```

С помощью предложения `ifFrameLoaded` в код можно включить имя сцены:

```
ifFrameLoaded ("intro", 60) {
```

Единственным ограничением является то, что предложение `ifFrameLoaded` не поддерживает оператора `else` (тогда как `_ifFramesLoaded` такую поддержку обеспечивает). Внедрить оператор `else` в двухкадровую структуру можно с помощью двухкадрового цикла.

ПОВТОРЕНИЕ ДЕЙСТВИЙ с помощью циклов

Циклические операторы `while`, `do-while`, `for` и `for-in` используются для выполнения повторяющихся действий. Каждый из них неоднократно оценивает условие и использует результат оценки для того, чтобы решить, следует ли продолжать цикл в теле предложения. Например, цикл можно использовать для повторяющейся проверки истечения определенного промежутка времени либо для неоднократного дублирования видеоклипа до тех пор, пока не будет создано определенное количество клипов.

Четыре циклических оператора тесно связаны друг с другом. Операторы `while` и `do` фактически представляют два способа выполнения одной и той же задачи. Использование того или иного из них зависит исключительно от предпочтений программиста. Компилирующая программа Flash транслирует их оба в циклы `while` SWF-файла. Оператор `do-while` является незначительной вариацией `while`.

ПРЕДЛОЖЕНИЕ WHILE

Предложение `while` является основным для выполнения повторяющихся действий. Его формат следующий:

```
while ( выражение ) {  
    предложение1;  
    предложение2;  
    предложение3;  
}
```

Интерпретирующая программа циклически выполняет предложения до тех пор, пока выражение остается истинным, или до тех пор, пока она не столкнется с оператором `break`, — в этом случае происходит выход из цикла. Например, приведенный ниже цикл `while` включает в программу полусекундную (500 миллисекунд) задержку. Этот цикл составлен на основе того факта, что истекшее время является разностью текущего времени и исходного времени.

```
initialTime = getTimerO;  
while (elapsedTime < 500) {  
    (elapsedTime = getTimerO - initialTime);  
}
```

Приведенный ниже код является наиболее распространенным применением оператора `while`. Этот код создает несколько копий видеоклипа, каждой из которых присваивается имя и которая помещается в свой уровень. В данном примере создается девять копий видеоклипа `myClip`. Копия `myClip1` помещается на уровень 1, `myClip2` — на уровень 2 и т.д., заканчивая копией `myClip9`:

```
var i = 1;  
while (i < 10) {  
    duplicateMovieClip("myClip", "myClip"+i, i);  
    i++;  
}
```

Этот пример также иллюстрирует наиболее распространенную структуру циклов `while`. Счетчик (`i`) инициализируется перед циклом `while`, а в пределах цикла происходит его приращение. Условие становится ложным, когда счетчик достигает определенного числа.

Циклы `while` часто используются для доступа к массивам. Приведенная ниже программа считывает элементы из массива `menu` и помещает их в массив `myOrder` до тех пор, пока она не доходит до элемента `"pie"`. В этой точке программа останавливается.

```

menu = ["potatoes", "steak", "peas", "salad", "pie", "ice cream"];
myOrder = []; // пустой массив, с которого начинается выполнение
i = 0; // инициализация счетчика
while (menu[i] != "pie") {
    myOrder[i] = menu[i];
    i++; // приращение счетчика
}
trace(menu); // potatoes, steak, peas, salad, pie, ice cream
trace(myOrder); // potatoes, steak, peas, salad

```

ПРЕДЛОЖЕНИЕ DO-WHILE

Тело цикла `while` не будет исполнено даже один раз, если с самого начала условие будет ложным. Чтобы цикл `while` был выполнен хотя бы один раз, используется предложение `do-while`.

Так, приведенный ниже код выполняет ту же самую задачу, что и цикл `while` в предыдущем примере, за исключением того, что программа будет выполнять цикл даже в случае, если элемент `"pie"` является первым в массиве `menu`, т.е. элементом `menu[0]`.

```

// ДЕФЕКТ !!! опасность неопределенного цикла
do {
    myOrder[i] = menu[i];
    i++;
} while (menu[i] != "pie");

```

Однако этот код таит в себе опасность! Если `"pie"` является первым элементом массива `menu`, т.е. `menu[0]`, то цикл `do-while` поместит `"pie"` в элемент массива `myOrder[0]`. Затем будет выполнено приращение `i` до 1 и проверено, является ли элемент `menu[1]` равным `"pie"`. Если это не так, то условие `(menu[i] != "pie")` будет истинным. При выполнении приращения `i` это условие останется истинным всегда, и интерпретирующая программа *заикнется*.

Совет

Подробная информация о неопределенных циклах приведена в главе 15 "Объединение предложений в функции".

Совет

Вы считаете, что попали в неопределенный цикл в обработчике клипа `enterFrame?` Обратитесь к разделу "Возможные проблемы" в конце этой главы.

ПРЕДЛОЖЕНИЕ FOR

В большинстве циклов `while` имеется счетчик, который инициализируется и обновляется. Предложение `for` делает инициализацию и обновление частью управляющей структуры, заключенной в круглые скобки. Предложение `for` имеет следующий формат:

```

for (инициализация; условие; обновление) {
    предложение1;
    предложение2;
    предложение3;
}

```

Ниже приведен пример использовавшегося ранее для дублирования видеоклипов цикла `while`, представленного в форме цикла `for`:

```

for (var i = 1; i < 10; i++) {
    duplicateMovieClip("myClip", "myClip"+i, i);
}

```

Два цикла одинаковы не только функционально: результирующий код SWF-файла для каждого из них полностью идентичен. Следовательно, не имеет значения, какой цикл вы используете, `while` или `for`, это исключительно дело вкуса. Цикл `for` более лаконичен. Его структура также напоминает о необходимости включения приращения. Приращение, оставленное вне цикла `while`, является наиболее распространенной ошибкой, ведущей к появлению неопределенного цикла. С другой стороны, цикл `while` позволяет отдельно комментировать выражения инициализации, условия и приращения.

Ниже приведен ранее рассмотренный код доступа к массиву, преобразованный в цикл `for`.

```
for (i = 0; menu[i] != "pie"; i++) {  
    myOrder[i] = menu[i];  
}
```

Совет

В предложении `for` может содержаться несколько счетчиков. За подробной информацией обратитесь к главе 13 "Использование операторов".

ПРЕДЛОЖЕНИЕ FOR-IN

Предложение `for-in` нумерует или перечисляет свойства объекта. Оно является исключительно полезным для исследования существующих объектов, которые вы не задавали самостоятельно, таких как встроенные объекты или объекты, связанные с компонентами. Формат предложения следующий:

```
for (переменная in имя_объекта) {  
    предложение1;  
    предложение2;  
}
```

Ниже приведен пример простого цикла `for-in`.

```
myObj = {prop1 : "a", prop2 : "b", prop3 : "c"}  
for (prop in myObj) {  
    trace(prop + " : " + myObj[prop]);  
}  
  
/*  
output:  
prop1 : a  
prop2 : b  
prop3 : c  
*/
```

Предложение `for-in` не выражает условия явным образом. Однако подразумевается следующее условие: "Есть ли еще какое-то свойство в заданном объекте?" Если оно есть, интерпретирующая программа исполняет тело предложения `for-in` в контексте этого свойства.

Обратите внимание на то, что переменная в управляющей структуре содержит *имя* (например, `prop1`), а не значение свойства (например, `"a"`). При наличии имени доступ к значению получить очень легко. Например:

```
trace(myObj.prop1); // a
```

Еще одним способом представления свойства объекта `myObj` `prop1` является выражение `myObj["prop1"]`. В данном примере `myObj[prop1]` сначала является эквивалентом `myObj["prop1"]`, затем — `myObj["prop2"]` и наконец — `myObj["prop3"]`.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему предложение switch не выполняет предложение case?

Помните, что в сравнениях **switch** используется операция строгого равенства. Уверены ли вы в том, что тестируемое значение и значение предложения **case** относятся к одному типу данных? Если в качестве тестируемого значения используется переменная, может возникнуть ошибка, вызванная тем, что тестируемым значением будет **undefined**. В качестве меры по устранению ошибки воспользуйтесь оператором **typeof** для подтверждения типа данных тестируемого значения непосредственно перед предложением **switch**.

Помогите! Я попал в неопределенный цикл в обработчике события enterFrame!

Заикливание в обработчике события вступительного кадра видеоклипа является очень распространенной ошибкой. В каждом кадре имеется по одному обработчику события вступительного кадра. Поэтому, если поместить в него счетчик, он будет каждый кадр автоматически обновляться. Например:

```
onClipEvent (enterFrame) {
    frameCounter++;
    // код выполняется
}
```

Пусть вы решили использовать этот счетчик в цикле **while** следующим образом:

```
onClipEvent (enterFrame) {
    frameCounter++;
    while (frameCounter < 10) {
        // НЕОПРЕДЕЛЕННЫЙ ЦИКЛ !!!
    }
}
```

Результатом является неопределенный цикл. Интерпретирующая программа погружается в цикл **while** и никогда не выходит из него, потому что счетчик никогда не обновляется внутри цикла. Вывод: цикл **while** должен иметь свой собственный счетчик, **обновляемый внутри** цикла.

FLASH ЗА РАБОТОЙ: ИСПОЛЬЗОВАНИЕ ЦИКЛА FOR для СОЗДАНИЯ СПИСКА

В демонстрационном фильме **dropdownmenu fla**, содержащемся на компакт-диске, создается раскрывающееся меню, элементы которого берутся из базы данных SQL. В данном случае элементами меню являются названия 50 штатов США и округа Колумбия.

Раскрывающееся меню используется при регистрации на узле <http://www.assexpressedbyyou.com>.

На заметку

С описанием этого узла можно познакомиться в статье Майкла Гурвица (Michael Hurwicz), опубликованной в журнале New Architect в марте 2002 года. В Internet эта статья размещена по адресу <http://www.newarchitectmag.com/documents/s=2286/na0202cs1/index.html>.

Раскрывающееся меню является всего лишь одной небольшой частью узла. Здесь мы рассмотрим только небольшую часть этого меню — функцию **generateList()**, — основой ко-

торой является цикл `for`. Файл `dropdownmenu fla` является полностью функционирующим, однако он создает меню, совместимые только с Flash 5. На компакт-диске имеется также файл `sqllookup.php`, который необходим для доступа к базе данных SQL. Он написан на серверном языке сценариев PHP 4.0.6. (Подробная информация о PHP приведена по адресу <http://www.php.com>.)

В раскрывающемся меню указываются названия штатов, по 10 за один раз. Функция `generateList()` последовательно просматривает элементы массива (массив `items`), по 10 за один раз.

Каждый элемент содержит ссылку на объект. Каждый объект имеет два свойства: `id` — целое число, уникальным образом идентифицирующее объект, и `name` — имя штата, представленное в виде строки.

Каждый выбранный элемент меню представляет собой видеоклип. Имеется 10 таких клипов, с именами `entity0`, `entity1`...`entity9`. Клипы вручную выложены на временной шкале компонента меню, начиная с кадра 12. Чтобы увидеть это, дважды щелкните на компоненте меню панели **Movie Explorer** (Проводник фильма), в результате чего откроется временная шкала компонента меню.

При каждом прохождении цикла `for` функция `generateList()` берет один объект из массива `items` и переносит значения свойств этого объекта в свойства `text` и `id` одного из десяти видеоклипов `entity`. Это происходит в строках 13 и 14 приведенного ниже листинга кода. Кроме того, в строке 12 функция делает выбранный элемент меню видимым.

Вторая половина программы, начиная со строки 18, делает соответственно видимыми и невидимыми стрелки перемещения по списку вверх и вниз, в зависимости от того, есть ли еще подлежащие отображению имена до или после выбранного в текущий момент пакета из 10 имен.

```
// used by permission of Mass Transmit (http://www.masstransmit.com)
1: var maxdisplay = 10;
2: function generateList(pagenum) {
3:   var startnum = maxdisplay * pagenum;
4:   for(i = startnum; i < startnum + maxdisplay; i++) {
5:     var entity = eval("entity" + (i % maxdisplay));
6:     var entityObj = items[i];
7:
8:     if (i >= items.length) {
9:       entity._visible = false;
10:    }
11:   } else {
12:     entity._visible = true;
13:     entity.text = entityObj.name;
14:     entity.id = entityObj.id;
15:   }
16: } // end for loop
17:
18: if (pagenum == 0) { // set up arrow
19:   uparrow._visible = false;
20: }
21: else {
22:   uparrow._visible = true;
23: }
24:
25: if ((startnum + maxdisplay) >= items.length) { // set down arrow
26:   darrow._visible = false;
27: }
28: else {
29:   darrow._visible = true;
30: }
31:
32: }
```

Кто захочет лучше разобраться с этим кодом, должен выполнить несколько действий.

PHP-сценарий, размещенный на узле <http://masstransit.com/demo/sqllookup>, является жестко закодированным на возвращение списка штатов. Он не является тем же самым, что и PHP-файл на компакт-диске. Последний можно использовать для любого SQL-поиска, инициированного из Flash-файла.

В своем приложении вы можете изменить предложение SQL `select` в параметрах клипа для компонента `menu`. При изменении предложения SQL `select` необходимо запрограммировать Flash-приложение так, чтобы оно соответствовало возвращаемым данным.

В том виде, в котором он есть, фильм `dropdownmenu fla` можно использовать с любой SQL-таблицей, в которой есть поле `id`, являющееся целым числом, и поле `name`, являющееся строкой. Например, в таблице `state` базы данных `mySQL`, доступ к которой осуществляется посредством узла <http://www.masstransit.com/demo>, имеются следующие поля:

```
state_id: integer
name: varchar(64)
```

Если в таблице SQL полям не присвоены имена `id` и `name`, то для соответствующего переименования полей можно воспользоваться элементом `as` в операторе `select`. В приведенном ниже примере поле `indexnumber` переименовано в `id`, а поле `stringvalue` — в `name`.

```
select indexnumber as id, stringvalue as name from mytable;
```

Файл `dropdownmenu fla` содержит следующее SQL-предложение:

```
select state_id as id, name from state where state_id > 0;
```

Чтобы просмотреть SQL-предложение, в основной временной шкале щелкните на компоненте и откройте панель **Properties** (Свойства).

Файл базы данных не обязательно должен быть размещен на самом Web-сервере. При выполнении поиска базы данных PHP соединяется с демоном (сетевой программой) `MySQL`, выполняющимся на этом же сервере. Демон подключается к базе данных `mySQL`, которая может находиться как на этом же сервере, так и на каком-то другом. В ответ будет получена таблица базы данных, выложенная в виде ряда строк с номерами полей в каждой из них. Сценарий PHP принимает эту таблицу, форматирует ее подходящим для Flash образом и отправляет отформатированные данные в приложение, вызывающее Flash.

При попытке просмотреть источник PHP-файла (или любой другой сценарий PHP или CGI) в диалоговом режиме вы не увидите фактического кода. Вместо него вы увидите только текст, сгенерированный путем исполнения сценария PHP или CGI. В этом случае текст будет начинаться так:

```
result=1&numitems51&id0=1&name0=Alabama...
```

Фактически можно взять этот текст, сохранить его на Web-сервере в виде текстового файла (например, <http://your.domain.com/states.txt>), а затем в фильм `dropdownmenu fla` можно будет загрузить текстовый файл. Вы получите тот же список штатов без поиска базы данных.

"Единственным, что заставляет меня лезть на стену при работе с Flash, является то, что Flash-фильмы, доступ к которым осуществляется посредством браузера, могут контактировать только с приложением CGI или PHP, находящимися в том же субдомене, что и Flash-фильм", — отмечает Марк Льюис, ведущий специалист по разработке приложений узла `Mass Transit`. Ограничение, накладываемое одним субдоменом, не имеет значения при запуске Flash-фильма с жесткого диска своего компьютера без использования браузера. Однако при использовании браузера Flash-фильм и сценарий PHP или CGI должны быть на одном и том

же сервере. Это ограничение, введенное компанией Macromedia в целях безопасности, упомянуто в справочнике по **ActionScript**, в статье **LoadVariables**.

"Хотелось бы, чтобы Macromedia предоставила вопросы безопасности на рассмотрение серверных программистов, вместо того, чтобы внедрять их в принудительном порядке", — говорит Льюис. "Фактически это мешает нам разрабатывать некоторые Flash-приложения".

Чтобы просмотреть файл **dropdownmenu fla** на своем узле так же, как другие пользователи смогут сделать это посредством своих браузеров, необходимо установить базу данных **mysql** и отредактировать PHP-сценарий, вставив информацию, аналогичную приведенной ниже, для разрешения доступа к базе данных.

```
$dbname = "genericdb"; // имя базы данных, подлежащей использованию
$dbuser = "username"; // имя пользователя базы данных
$dbpass = "passwd"; // пароль для имени пользователя
```


ОБЪЕДИНЕНИЕ ПРЕДЛОЖЕНИЙ В ФУНКЦИИ

В ЭТОЙ ГЛАВЕ...

Улучшение кода с помощью функций	295
Создание функций	296
Вызов функций	298
Использование ключевого слова <code>var</code> для создания локальных переменных внутри функций	298
Передача данных в функции с помощью аргументов	300
Извлечение результатов с помощью возвращаемых значений	305
Функции как классы: функции-конструкторы	306
Методы <code>Function.apply()</code> и <code>Function.call()</code>	316
Явный обзор данных	319
Автоматический обзор данных	322
Возможные проблемы	325
Flash за работой: расчет факториалов — пример рекурсии	326

УЛУЧШЕНИЕ КОДА с помощью ФУНКЦИЙ

Функция — это целостный блок кода, к которому можно обратиться (или активизировать) из любого места программы. При обращении к функции ее код начинает выполняться. Вызвать функцию можно по ее имени, если она его имеет. Кроме того, к функциям можно об-

ращаться посредством переменных, элементов массивов или свойств объектов. Все эти способы обращения к функции гораздо более лаконичны по сравнению с самой функцией. Таким образом, функции представляют собой эффективный и удобный "рычаг управления" потенциально сложными поведением.

Функции существенно облегчают поддержку программ. Предположим, что для отображения сообщений об ошибке вы создали функцию `displayError()`. Если когда-либо вы захотите изменить ее, то вам нужно будет внести изменения только в одном месте — в объявлении функции `displayError()`. Если вместо функции использовался несколько раз повторяющийся блок кода, то каждый такой блок придется исправлять в индивидуальном порядке.

Функции существенно улучшают удобочитаемость кода и возможность его повторного использования. Они также способствуют уменьшению размера SWF-файла за счет использования краткого обращения к функции вместо длинного повторяющегося блока кода. И напоследок отметим, что функции лежат в основе методики объектно-ориентированного программирования (ООП).

Совет

Знакомство с функциями, удобочитаемостью, возможностью повторного использования кода и объектно-ориентированным программированием состоялось в главе 11 "Знакомство с ActionScript".

Совет

Методика ООП подробно рассматривается в главе 19 "Использование встроенных базовых объектов" и в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

СОЗДАНИЕ ФУНКЦИЙ

Существуют два основных момента работы с функциями: их создание и вызов.

В зависимости от того, хотите ли вы, чтобы функция имела собственное постоянное имя или нет, ее можно создать одним из двух способов.

- Чтобы создать функцию с собственным постоянным именем, ее нужно *объявить*.
- Чтобы создать функцию без собственного постоянного имени, необходимо создать *функциональный литерал*. Чтобы к нему можно было обратиться, функциональный литерал должен храниться в переменной, элементе массива или свойстве объекта.

ОБЪЯВЛЕНИЕ ФУНКЦИЙ

При объявлении функции устанавливается постоянная связь имени функции с блоком кода. Базовый формат объявления функции имеет следующий вид:

```
function идентификатор(список) {  
    предложение1;  
    предложение2;  
}
```

где **function** — это ключевое слово; **идентификатор** — имя функции; **список** — необязательный список разделенных запятыми *аргументов*, передаваемых функции, а *предложение1* и *предложение2* представляют собой набор предложений ActionScript. В наборе может содержаться любое количество предложений.

Совет

Имя функции должно быть действительным идентификатором, образованным в соответствии с правилами, описанными в главе 12 "Управление переменными, данными и типами данных".

Например, ниже приведено объявление функции без аргументов.

```
function displayError() {  
    errorText = "Error!";  
}
```

Идея заключается в том, что эта функция обновляет переменную временной шкалы, которая отображается в текстовом поле, уведомляя пользователя об ошибке.

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ БЕЗ ИМЕН: ФУНКЦИОНАЛЬНЫЕ ЛИТЕРАЛЫ

Литерал — это выражение, представляющее элемент данных *буквально* (в противоположность *абстрактному* представлению). Он реализует элемент данных, а не называет его.

Совет

С числовыми, строковыми литералами и литералами массивов вы познакомитесь в главе 12 "Управление переменными, данными и типами данных".

Функциональный литерал — это функция, которая написана без присвоения ей имени. Ниже приведен пример функционального литерала, который выглядит, как объявление функции без имени.

```
function O {  
    errorText = "Error!";  
}
```

Как и остальные литералы, функциональный литерал после создания необходимо где-то сохранить, иначе он будет потерян. Например, функциональные литералы можно хранить в переменных. При этом синтаксически правильно будет после фигурной скобки вставить точку с запятой, чтобы завершить предложение, хотя пропуск точки с запятой на самом деле не приведет к возникновению проблем. Например:

```
displayError = function O {  
    trace("Error!");  
}; // точка с запятой завершает назначение предложения
```

Кроме того, функциональные литералы можно хранить как свойства объекта (это распространенный и удобный синтаксис создания методов). В следующем примере функции назначены не с помощью предложений присваивания, и, следовательно, после фигурных скобок тел двух функций точки с запятой не стоят. Однако, чтобы завершить предложение присваивания, этот символ стоит в самом конце объявления:

```
myObj = {prop1 : function(){trace("prop1");} ,  
        prop2 : function(){trace("prop2");} };
```

Ниже показан еще один способ выполнения той же задачи. Первая строка создает пустой объект, вторая и третья добавляют к нему свойства. Каждое свойство относится к функции. В данном примере функции назначены с помощью предложений присваивания, следовательно, после закрывающих фигурных скобок тел двух функций ставятся точки с запятой:

```
myObj = {}; // создание пустого объекта  
myObj.prop1 = function(){trace("prop1");};  
myObj.prop2 = function(){trace("prop2");};
```

В приведенном ниже примере те же самые функции назначаются как свойства массива.

```
myArray = []; // создание пустого объекта  
myArray[0] = function(){trace("prop1");};  
myArray[1] = function(){trace("prop2");};
```

Вызов ФУНКЦИЙ

При объявлении функции фактически в программе ничего не происходит. Чтобы было выполнено какое-либо действие, функцию нужно вызвать или выполнить.

Обращение к функции осуществляется с помощью оператора вызова функции, представляющего собой пару круглых скобок: `()`.

Совет

Оператор вызова функции был описан в главе 13 "Использование операторов".

Синтаксис обращения к функции очень простой:

идентификатор (*список*) ;

где *идентификатор* — это имя функции, а *список* — опциональный список аргументов, разделенных запятыми.

Многие "действия" **ActionScript** являются встроенными функциями, которые можно вызывать без их объявления. Например, `nextFrame()` является встроенной функцией, которая отправляет воспроизводящую головку к следующему кадру и затем останавливает ее. Если функция не имеет аргументов, круглые скобки остаются пустыми. Добавление точки с запятой завершает предложение, как показано ниже.

```
nextFrame();
```

Обращение к пользовательским функциям (к тем, что созданы самостоятельно) выполняется точно таким же образом:

```
displayError(); // активизирует функцию displayError()
```

Функция `displayError()` является тривиальной, и ей недостает преимуществ сведения сложных выполняемых операций под одним простым описательным именем. В данном случае выражение `errorText = "Error!"` практически является столь же кратким и удобочитаемым, как и `displayError()`. Основным аргументом в пользу функции `displayError()` является то, что ее легче поддерживать. Если в программе вы напрямую использовали выражение `errorText = "Error!"`, а впоследствии решили внедрить более сложный метод отображения ошибок, вам придется изменять этот участок кода везде, где он будет появляться в программе.



Для вызова любой функции можно также воспользоваться методами `call()` и `apply()` объекта `Function`.

Совет

Подробная информация о методах `call()` и `apply()` приведена ниже, в разделе "Методы `Function.apply()` и `Function.call()`".

ИСПОЛЬЗОВАНИЕ КЛЮЧЕВОГО СЛОВА **VAR** ДЛЯ СОЗДАНИЯ ЛОКАЛЬНЫХ ПЕРЕМЕННЫХ ВНУТРИ ФУНКЦИЙ

Большинство функций содержит одну или несколько переменных, которые используются только в пределах этой функции. Предположим, что вы создаете функцию `greet()`, которая в зависимости от времени дня возвращает одно из трех приветствий ("доброе утро", "добрый день" и "добрый вечер"). Для определения времени в **ActionScript** вы создаете и объект `Date` (например, `myDate`), и переменную времени (например, `hours`). Вы хотите использовать

объект `myDate` и переменную `hours` только в пределах функции `greet()` и при этом для обращения к различным объектам и переменным желаете использовать эти *имена* и в других местах программы. Заглядывая в будущее, вы также хотите иметь возможность внедрять эту функцию в другие программы и хотите быть уверенным, что использование уже существующего имени не приведет к конфликту имен и, следовательно, не повлияет на элементы (переменные или объекты) другой программы, в которой теперь непреднамеренно будут использоваться одинаковые имена.

В подобной ситуации для объявления *локальной* переменной внутри функции можно воспользоваться ключевым словом `var`. Локальная переменная функционирует только в пределах функции и не влияет на переменную с тем же именем, находящуюся вне функции. Например:

```
function greet () {
    var hours;
    var myDate;
    // код для получения времени и возврата приветствия
    // все применения переменной hours и объекта myDate осуществляются
    локально внутри функции
}
```

В приведенном ниже примере предложение `trace (myVar)` отображает локальную переменную, а не переменную временной шкалы с тем же именем.

```
myVar = "timeline";
function myFunc () {
    var myVar = "local";
    trace (myVar); // "local"
}
```

Если в предыдущем примере опустить ключевое слово `var`, то предложение `myVar = "local"` будет *ссылаться* на переменную временной шкалы с тем же именем и *изменит* ее! Если вы хотите использовать какую-то переменную только в пределах функции, всегда объявляйте ее с помощью ключевого слова `var`. Это поможет избежать непреднамеренного изменения переменной временной шкалы с тем же именем.

Совет

Аргументы также трактуются как локальные переменные. Аргументы рассматриваются ниже, в разделе "Передача данных в функции с помощью аргументов".

КЛЮЧЕВОЕ СЛОВО VAR НЕ ИСПОЛЬЗУЕТСЯ ВНЕ ФУНКЦИЙ

Ключевое слово `var` не оказывает воздействия вне функций. Это относится и к обработчикам событий! Ключевое слово `var` можно использовать в обработчике событий, но его присутствие ничего не изменит. Например, если переменная создается в обработчике событий `load`, как показано ниже, *она может* быть изменена предложениями, расположенными во временной шкале видеоклипа, в других обработчиках событий или в коде соответствующих кнопок.

```
onClipEvent (load) {
    var c = 10; // ключевое слово "var" ничего не меняет
}
```

ДОСТУП К ПЕРЕМЕННЫМ ВРЕМЕННОЙ ШКАЛЫ ВНУТРИ ФУНКЦИИ

Если во временной шкале имеется функция и внутри нее с помощью ключевого слова `var` создана локальная переменная, то к переменной временной шкалы, имеющей то же самое имя, можно обратиться с помощью ключевого слова `this`. Например:

```
myVar = "timeline";
function myFunc() {
    var myVar = "local";
    trace (myVar); // "local"
    trace (this.myVar); // "timeline"
}
```

ПЕРЕДАЧА ДАННЫХ В ФУНКЦИИ С ПОМОЩЬЮ АРГУМЕНТОВ

Функция `displayError()`, рассмотренная выше, в разделе "Вызов функций", может отображать только одно сообщение: "Error!" (ошибка). Предположим, вы хотите, чтобы пользователь получил более подробную информацию об ошибке. Это можно сделать, не создавая отдельной функции для каждой ошибки, а передавая в функцию `displayError()` *аргумент*. В приведенном ниже примере аргумент имеет имя `arg`. Параметр `arg` существует в памяти только на время действия функции. Когда выполнение функции завершается, аргумент `arg` исчезает. Он выполнил свою задачу по изменению переменной `errorText`.

```
function displayError(arg) {
    errorText = arg;
}
```

Для отображения различных сообщений об ошибке можно обратиться к функции, как показано в следующем примере:

```
displayError("Error: required field, cannot be left blank");
displayError("Error: password must be 8 or more characters");
displayError("Error: invalid zip code");
```

Функция `displayError()` будет отображать переданный ей аргумент. Добавление аргументов подобным образом делает функцию `displayError()` гораздо более гибкой и полезной. Кроме того, функция становится более доступной для повторного использования. К примеру, с ее помощью можно отображать различные сообщения об ошибках в разных программах.

Число аргументов, которые можно передать функции, не ограничивается. Однако в пределах функции по имени можно обратиться только к тем аргументам, которые были объявлены. Доступ к остальным аргументам можно получить посредством объекта `arguments`, который будет описан далее в этой главе.

ПЕРЕДАЧА АРГУМЕНТОВ по ЗНАЧЕНИЮ и по ССЫЛКЕ

Когда в виде аргумента в функцию передается элементарная величина, она передается *по значению*. Это означает, что интерпретирующая программа выделяет соответствующее место в памяти и копирует в него значение этого элемента данных. Функция никогда не видит исходный элемент данных, а имеет дело только с его копией. Изменение копии в пределах функции не оказывает влияния на исходную величину.

В приведенном ниже примере показано, как передать в функцию строковый примитив и модифицировать переданное значение без изменения оригинала. В данном случае к переданному значению функция добавляет два восклицательных знака, а исходная строка остается без изменений:

```
str = "string";
function myFunc (arg) {
    arg += " !! ";
    return (arg) ;
}
```

```
result » myFunc(str);
trace(result); // отображается "string !!"
trace(str); // отображается "string" - оригинал не меняется
```

С другой стороны, сложные типы данных (объекты, массивы, функции) передаются *по ссылке*. Это означает, что интерпретирующая программа создает указатель на исходный элемент и передает его функции. Любые изменения данных внутри функции *вливают* на оригинал. В приведенном ниже примере показано, как изменяется оригинал при передаче объекта в функцию и изменении его свойств. В данном примере функция изменяет свойство строки str, относящееся к объекту obj. Когда функция модифицирует значение свойства на "changed", она изменяет и свойство исходного объекта:

```
function myFunc(arg) {
    arg.str = "changed"; // изменение исходного значения
}
obj = new Object ();
obj.str = "property";
myFunc(obj);
trace(obj.str); // отображается "changed"- объект имеет новое значение
```

ОБЪЕКТ ARGUMENTS

При вызове функции она автоматически получает свойство, именуемое arguments и представляющее собой массив (или подобный массиву объект в Macromedia Flash 5) с элементами, доступ к которым осуществляется посредством числовых индексов (начиная с 0), и параметром length, указывающим число элементов массива.

При каждом вызове функции создается объект arguments, который заполняется всеми переданными ей аргументами. Кроме того, свойство arguments.length задает число аргументов.



Объект arguments поддерживается плеерами Flash 5 и Flash 6. Однако обрабатывается данный объект этими плеерами по-разному. Например, во Flash 5 со свойством arguments не работают такие методы массивов, как push () и pop ().

В плеере Flash 6 arguments — это реальный массив со все ми соответствующими методами. Кроме того, свойство arguments.callee, работающее в цикле for-in во Flash 5, не поддерживается во Flash 6.

На заметку

Во Flash 6, в отличие от Flash 5, поддерживается свойство arguments.caller.

Свойство arguments доступно только внутри функции при ее выполнении. К нему можно обратиться либо просто как arguments, либо указав перед ним имя функции, например myFunc.arguments.

В приведенной ниже функции myFunc 0 предложения trace(arg1) и trace(arguments[0]) выполняют одну и ту же задачу — отображают значение первого (и единственного) аргумента.

```
function myFunc(arg1) {
    trace(arg1); // отображается "this is the argument"
    trace(arguments[0]); // отображается "this is the argument"
}
myFunc("this is the argument");
```

Объект arguments имеет три свойства:

- length— длина массива arguments;
- callee — функция, выполняющаяся в текущий момент;
- caller — функция, вызвавшая выполняющуюся в текущий момент функцию.



Выше были перечислены *свойства* объекта, а не *элементы* массива. Эти свойства будут присутствовать даже в том случае, если массив `arguments` совсем не имеет элементов, т.е. не было передано никаких аргументов.

СВОЙСТВО ARGUMENTS.LENGTH

Одним из применений свойства `arguments.length` является проверка того, что функции было передано правильное число аргументов. В приведенном ниже примере показано, как для функции можно назначить свойство `length`. Свойство содержит ожидаемое число аргументов, в данном случае — один. При вызове функции с помощью свойства проверяется, совпадает ли фактическое число аргументов с ожидаемым. Если это не так, то возвращается сообщение об ошибке. В данном примере функция `myFunc()` отображает сообщение "Wrong number of arguments! Expected 1, got 0" (Неправильное число аргументов! Ожидалось 1, получено 0).

```
function myFunc(arg) {
    if (myFunc.length != arguments.length) {
        err = "Wrong number of arguments! expected ";
        return (err + myFunc.length + ", got " + arguments.length);
    }
}
```

```
myFunc.length = 1; // ожидаемое число аргументов
trace ( myFunc(arg1, arg2) ); // это приведет к сообщению об ошибке
```

Массив `arguments` со свойством `length` также будет полезен, когда функции передается переменное число аргументов. Например, приведенная ниже функция `avg()` возвращает среднее любого количества аргументов. Массив `arguments` и свойство `arguments.length` позволяют функции `avg()` работать с переменным числом аргументов.

```
function avg() {
    var sum = 0;
    // добавление всех аргументов
    for(var i = 0; i < arguments.length; i++) {
        sum = sum + arguments[i];
    }
    // разделить сумму на число аргументов - это среднее значение
    var average = sum/arguments.length;
    return average;
}
trace("Average = " + avg(2,4,6)); // отображается "Average = 4"
trace("Average = " + avg(10,20,30,40,50)); // отображается "Average = 30"
```

СВОЙСТВО CALLEE

Свойство `arguments.callee` позволяет обратиться к функции изнутри самой функции без использования ее имени.

Одним из предназначений свойства `arguments.callee` является создание более обобщенного кода, который можно легко вырезать и вставлять в различные программы. Например, рассмотренный в предыдущем подразделе код проверки аргументов в том виде, в каком он написан, предназначен только для функции `myFunc()`. Однако в этой функции можно заменить `myFunc` на `arguments.callee`. Результирующий код можно будет вырезать и вставить в любую функцию, имеющую свойство `length`, и этот код будет работать.

При работе с обычными именованными функциями использование свойства `arguments.callee` для создания обобщенного кода является исключительно вопросом удобства. Каждая функция имеет постоянное имя, которое всегда можно использовать.

И наоборот, анонимные функции, созданные посредством функциональных литералов, не имеют фиксированного имени. В любой момент функция может быть поставлена в соответствие определенной переменной, элементу массива или свойству объекта, а впоследствии это назначение можно будет изменить. В этом случае использование свойства `arguments.callee` для создания обобщенного кода является мерой безопасности.

Ниже приведен пример проблемы, возникновению которой может воспрепятствовать свойство `arguments.callee`. Пусть создается функциональный литерал и первоначально ему в соответствие ставится переменная `firstVar`. Вы хотите, чтобы функциональный литерал мог обращаться к самому себе, и для этого включаете в него обращение к `firstVar()`:

```
var firstVar = function (i) {
    if (i < 3) {
        trace(i);
        return firstVar(i+1);
    }
};
firstVar(0);
/*Output
0
1
2
*/
```

Затем вы переназначаете функциональный литерал для другой переменной, `secondVar`, и избавляетесь от переменной `firstVar`:

```
secondVar = firstVar;
delete firstVar;
secondVar(0);
/*Output
0
*/
```

Теперь при обращении к `secondVar()` происходит попытка вызова внутри функционального литерала `firstVar()`, и активизация функционального литерала не выполняется.

С другой стороны, если включить обращение к свойству `arguments.callee()`, то функциональный литерал будет активизировать сам себя, независимо от того, какой переменной он поставлен в соответствие. Вот как должен выглядеть функциональный литерал. Назначьте и переназначьте его для любой переменной, и он будет работать:

```
function (i) {
    if (i < 3) {
        trace(i);
        return arguments.callee(i+1);
    }
};
```

Функции, обращающиеся сами к себе, называются *рекурсивными*. Следовательно, свойство `arguments.callee()` позволяет создавать "надежные" анонимные рекурсивные функции.

Совет

Более подробно тема рекурсии освещается ниже, во врезке "Несколько простых примеров рекурсии".

Переназначение другого имени переменной **является** не единственной причиной "сбоя" рекурсивной анонимной функции с внутренним жестко закодированным обращением. Даже при сохранении одного и того же имени переменной в пределах анонимной функции она может остаться вне поля зрения.

Вопрос "видимости" функциональным литералом переменной, служащей в качестве его имени, и, таким образом, исполнения литерала является предметом области действия имен. В двух приведенных ниже ситуациях переменная, присвоенная функциональному **литералу**, находится вне области действия в пределах функционального литерала.

- Функциональный литерал присвоен локальной переменной внутри функции.
- Функциональный литерал используется в функции-конструкторе.

Приведенный ниже пример иллюстрирует первую ситуацию.

В функции `myFunc()` переменная `localFunc` является локальной, для которой назначен функциональный литерал. Сначала активизируется функция `myFunc` (строка 12). Затем функция `myFunc()`, после возвращения (строка 9), обращается к переменной `localFunc()`. В точке, в которой `localFunc()` обращается к самой себе (строка 5), переменная `localFunc` является неопределенной. С другой стороны, свойство `arguments.callee` успешно исполняет `localFunc()`.

Функция `myFunc()` не выполняет никаких полезных **действий**, но она иллюстрирует ситуацию, в которой свойство `arguments.callee` оказывается очень полезным.

```
// спасибо Tatsuo Kato (http://tatsuokato.com) за эту идею!
1: myFunc = function (i) {
2:   var localFunc = function (i) {
3:     if (i < 3) {
4:       trace(i);
5:       return arguments.callee(i+1); // localFunc() здесь не
                                     // определена!
6:     }
7:     else return i;
8:   };
9:   return localFunc(i);
10:};
11:
12: r = myFunc(0);
13: trace ("r= "+r);
/*
Output
0
1
2
r = 3
*/
```

Несмотря на то что сама по себе функция `localFunc()` в строке 5 работать не будет, в этой ситуации будет выполняться функция `this.localFunc()`. И все-таки свойство `arguments.callee` является единственным, которое всегда позволяет анонимной функции обращаться к самой себе, а потому — самое легкое и безопасное в применении.

СВОЙСТВО CALLER



Свойство `caller` позволяет обратиться к функции, которая активизировала функцию, выполняющуюся в данный момент. Например:

```
function calleeFunc() {
arguments.caller(); // обратный вызов
}
```

Для изучения полезных применений свойства `arguments.caller` обратитесь к описанию функции `mySuper()` в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

При использовании свойства `arguments.caller` убедитесь, что при этом не будет создан бесконечный цикл.

Несколько простых примеров рекурсии

Рекурсия возникает, когда функция обращается к самой себе. В большинстве случаев рекурсию можно создать просто путем использования имени функции внутри нее же, как в приведенном ниже примере.

```
function infiniteLoop() { // НЕ ДЕЛАЙТЕ ЭТОГО!
    infiniteLoop();
}
```

Ту же самую задачу можно выполнить с помощью свойства `arguments.callee()`, как показано ниже.

```
function infiniteLoop() { // НЕ ДЕЛАЙТЕ ЭТОГО!
    arguments.callee();
}
```

Цикл будет бесконечным, даже если вставить предложение `return`, как показано ниже.

```
function infiniteLoop() { // НЕ ДЕЛАЙТЕ ЭТОГО!
    return arguments.callee();
}
```

В последнем случае при попытке интерпретирующей программы вычислить возвращаемое значение исполняется функция `infiniteLoop()`, т.е. цикл будет бесконечным.

Функция `infiniteLoop()` является ярким примером рекурсии, потому что все, что она делает, — это обращается к самой себе снова и снова. Это не только бесполезно, но и вредно, поскольку может привести к сбою приложений и компьютеров.



Flash MX автоматически прерывает такой сценарий и выводит сообщение об ошибке после выполнения 256 циклов. Плеер Flash 6 прерывает сценарий **без** вывода сообщения об ошибке. Flash 5 и соответствующий плеер отображают сообщение об ошибке и позволяют пользователю прервать сценарий.

При использовании браузеров возможность прерывания зависит от разных факторов, например, от используемой операционной системы. Но совершенно четко можно сказать, что никакие из результатов не будут полезными, и все может закончиться перезагрузкой компьютера.

Однако сама по себе рекурсия может быть полезной. Необходимо только обеспечить возможность выхода из нее!

Пример полезной рекурсии представлен в разделе "Flash за работой: расчет факториалов — пример рекурсии" в конце этой главы.

ИЗВЛЕЧЕНИЕ РЕЗУЛЬТАТОВ с помощью ВОЗВРАЩАЕМЫХ ЗНАЧЕНИЙ

Иногда требуется, чтобы функция не только выполняла задачу, но и возвращала значение. Например, вы пишете функцию для удвоения чисел, но хотите, чтобы она не только выполняла эту операцию, но и возвращала удвоенные числа для последующего использования.

Функция возвращает значения благодаря предложению `return`. Предложение `return` с возвращаемым значением передает его вызывающей функции. Так, в приведенном ниже примере функция возвращает удвоенное переданное ей значение.

```
function doubleIt(numberToBeDoubled) {  
    return numberToBeDoubled * 2;  
}
```

Функция не обязательно должна включать в себя предложение `return`. При его отсутствии функция будет просто последовательно, сверху вниз, исполнять предложения и завершится по достижении последней фигурной скобки. Предложение `return` завершит функцию до того, как она достигнет последней фигурной скобки.

Предложение `return` без возвращаемого значения вернет значение `undefined`. Например, приведенная ниже функция `myFunc()`, если ей не было передано аргументов, возвращает значение `undefined`. В противном случае она возвращает сам аргумент.

```
function myFunc(arg) {  
    if (!arg) return;  
    return (arg);  
}  
result = myFunc();  
trace (result); // undefined
```

Совет

Подозреваете, что часть функции "отключена"? Возможно, это происходит из-за преждевременного возвращения. Обратитесь к разделу "Возможные проблемы" в конце этой главы.

Существуют два способа применения возвращаемого значения.

- Присвоить возвращаемое значение переменной и каким-либо образом использовать ее.
- Применить обращение функции к ней самой и в качестве эквивалента возвращаемого значения использовать переменную.

Например, приведенная ниже функция `doubleIt()` иллюстрирует первый подход.

```
twicefour = doubleIt(4);  
trace (twicefour); // отображается 8
```

Л этот пример демонстрирует второй подход:

```
trace(doubleIt(4)); // отображается 8
```

Оба примера иллюстрируют выполнение одной и той же задачи. Второй код в два раза короче первого как в исходном коде, так и в SWF-файле. Однако во втором случае не создается переменная, которую можно продолжать использовать, тогда как в первом случае это происходит,

Совет

Не получили ожидаемого возвращаемого значения? Обратитесь к разделу "Возможные проблемы" в конце этой главы.

ФУНКЦИИ КАК КЛАССЫ: ФУНКЦИИ-КОНСТРУКТОРЫ

Одним из самых важных типов функций во Flash является функция-конструктор. Функция-конструктор в комбинации с оператором `new` является способом создания новых объектов во Flash.

Функция-конструктор выглядит так же, как и обычная. Фактически ею она и является. Однако используется функция-конструктор очень специфическим образом, что позволяет извлечь максимум пользы из тех ее свойств, которые обычно оставались неиспользованными.

В частности, все функции имеют свойство `prototype`, но, как правило, оно не используется. Однако в функциях-конструкторах оно является ключевым, поскольку позволяет всем объектам класса иметь общие свойства, а также позволяет одному классу наследовать свойства других.

В этом разделе рассмотрены три основные темы.

- Создание новых объектов с помощью оператора `new`.
- Использование свойства `prototype` для коллективного использования свойств.
- Задание цепочек наследования с помощью оператора `new`.

Эти три темы представляют собой основу методики объектно-ориентированного программирования (ООП).

Совет

Более широко вопросы, связанные с объектно-ориентированным программированием, рассматриваются в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

СОЗДАНИЕ ОБЪЕКТОВ с помощью

ОПЕРАТОРА NEW

Для создания объекта сначала необходимо задать функцию-конструктор, а затем использовать ключевое слово `new`. Приведенная ниже функция-конструктор `Ball()` назначает только одно свойство `col` (цвет) для создаваемых объектов.

```
function Ball (col) {  
    this.col = col;  
}
```

Для создания объекта класса `Ball` активизируется функция `Ball()` с помощью оператора `new`.

```
myBall = new Ball ("blue");
```

За исключением ключевого слова `new` это предложение выглядит как обычное обращение к функции. Следовательно, сразу же можно обратить внимание, что данное предложение вызывает функцию `Ball()`, принимает возвращаемое значение и присваивает его переменной `myBall`. Так ведет себя обычная функция, и в данном случае происходит то же самое.

Однако здесь есть одна странность. Вы активизируете функцию так, как если бы она возвращала какое-то значение, но при этом оператор `return` отсутствует! Какое же значение возвращается? Ключевое слово `new` говорит функции о том, что следует создать новый объект. Этот объект и есть возвращаемое значение, которое присваивается переменной `myBall`.

Несмотря на то что строка кода `myBall = new Ball ("blue")` выглядит очень просто, она инициирует активизацию достаточно сложной цепочки событий.

- Сначала интерпретирующая программа **ActionScript** создает новый "родовой" объект и передает его функции `Ball()`. На данном этапе представим себе, что этот объект является пустым, не обладающим никакими свойствами.
- Внутри функции `Ball()` объект (который до сих пор является "анонимным" или безымянным) называется `this`. Каково предназначение предложения `this.col = col`? Элемент `col` в правой части предложения относится к аргументу, переданному функции, т.е. является строкой "blue". Левая часть предложения, `this.col`, создает для анонимного объекта новое свойство с именем `col`. Следовательно, результатом всего предложения будет объект `{col: "blue"}`.
- Функция `Ball()` возвращает все еще безымянный объект, обладающий новым свойством `col`. Эта часть процесса, в некотором роде, "скрыта", поскольку функция-

конструктор не имеет предложения `return`. Вместо этого оператор `new` иницирует возвращение объекта. В этом месте анонимному объекту наконец-то присваивается имя `myBall`.

- Интерпретирующая программа **ActionScript** задает скрытое свойство объекта `myBall`, свойство `_proto_`, для того, чтобы сослаться на объект `prototype` функции-конструктора `Ball()`. Это свойство позволяет объекту `myBall` найти в объекте `prototype` коллективно используемые свойства.

Совет

Свойство `_proto_` описывается в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

Теперь можно проверить, что же только что было сделано:

```
trace (typeof myBall); // "object"
for (a in myBall) trace(a); // перечисляет свойства: отображает "col"
trace(myBall.col); // "blue"
```

Все выглядит отлично!

Свойство `col` содержится в объекте `myBall` локально. Для каждого объекта, созданного функцией-конструктором `Ball()`, будет задано свойство `col`, и каждое из этих свойств будет локальным по отношению к каждому созданному объекту. Это означает, что при изменении значения свойства `col` в одном из объектов оно не повлияет на значения `col` других объектов. Это свойство не является коллективно используемым.

СВОЙСТВО PROTOTYPE

При создании каждой функции интерпретирующая программа **ActionScript** автоматически назначает для нее свойство `prototype`. Свойство `prototype` является объектом, т.е. может иметь свои собственные свойства. При создании нового объекта с помощью оператора `new` (как описано в предыдущем подразделе) любые свойства объекта `prototype` функции-конструктора используются вновь созданным объектом.

Предположим, что создается класс `Dog`, реализуемый в функции `Dog()`:

```
function Dog () {}
```

Этот пример не слишком похож на функцию, но, тем не менее, имеет объект `prototype`. Теперь назовем несколько свойств объекта `prototype` класса `Dog`:

```
Dog.prototype = {legs: 4, says: "bow wow", food: ["bones", "steak"]};
```

В переводе это означает, что прототипная собака имеет четыре лапы, лает "bow wow", ест кости и мясо.

Те же самые свойства для объекта `prototype` можно назначить и по-другому:

```
Dog.prototype.legs = 4;
Dog.prototype.says = "bow wow";
Dog.prototype.food = ["bones", "steak"];
```

На заметку

Имена классов традиционно пишутся с прописной буквы.

В качестве свойств объект `prototype` может иметь и функции. Функция, являющаяся свойством объекта, называется *методом*.

В этом конкретном примере никакие из свойств методами не являются. Методы мы рассмотрим позднее, поскольку при их назначении для объекта `prototype` необходимо принять во внимание несколько специальных моментов.

После создания указанных выше трех свойств объекта `Dog.prototype` класса `Dog` ими автоматически будет обладать любой другой объект этого класса. Можно создать 100 объектов класса `Dog`, и каждый из них будет иметь четыре ноги, лаять "bow wow", есть кости и мясо. Но эти данные необходимо хранить только в одном месте.

Разработчик Flash Робин Дебреуил (Robin Debreuil), на Web-узле которого размещен превосходный учебник по объектно-ориентированному программированию в `ActionScript`, привел хорошую аналогию этой ситуации. Предположим, вы смотрите через окно с двойным стеклом. И на первом, и на втором стекле есть несколько царапин. Но глядя через стекло, вы не можете отличить их. Вы просто видите ряд царапин. Первое стекло является объектом, а царапины — его свойствами. Второе стекло является объектом `prototype` функции-конструктора, а царапины на нем — свойства этого объекта. Когда вы, будучи программистом `ActionScript`, получаете доступ к свойству, то для вас не имеет значения, является ли оно локальным свойством (первого стекла) или свойством `prototype` (второго стекла). Эта аналогия разрушается, когда речь идет о нескольких объектах одного и того же класса (за которыми каким-то образом будет располагаться второе окно), но в случае с одним объектом она работает превосходно.

На самом деле происходит вот что: когда интерпретирующей программе `ActionScript` нужно найти свойство объекта (поскольку в программе имеется ссылка на нечто подобное `myDog.col`), то сначала поиск проводится в объекте (`myDog`). Если объект не содержит нужного свойства, интерпретирующая программа ищет прототип функции-конструктора данного объекта (`Dog.prototype`). Как правило, для программиста этот процесс прозрачен. При обращении к объекту `myDog.col` нет необходимости указывать, является ли свойство локальным по отношению к объекту или хранится в прототипе конструктора.

Пока что объект `Ball.prototype` не имеет никакого содержимого, т.е. для этого класса еще нет коллективно используемых свойств. Теперь создадим коллективное свойство:

```
Ball.prototype.className = "Ball"; // добавление свойства className
for (a in myBall) trace(a); // отображается "className" и "col"
trace(myBall.className); // "Ball"
```

Обратите внимание, что синтаксис обращения к свойству `myBall.className` является таким же, как и в обращении к `myBall.col`, даже несмотря на то, что `className` является локальным свойством объекта `prototype`, а свойство `col` — локальным для `myBall`.

НАЗНАЧЕНИЕ МЕТОДОВ для КЛАССА

Теоретически метод для класса можно назначить одним из четырех способов, приведенных в табл. 15.1.

ТАБЛИЦА 15.1. ЧЕТЫРЕ СПОСОБА НАЗНАЧЕНИЯ МЕТОДА для КЛАССА

ЛОКАЛЬНЫЙ (в КАЖДОМ ЭКЗЕМПЛЯРЕ)	КОЛЛЕКТИВНЫЙ (в ПРОТОТИПЕ)
Объявление функции	Объявление функции
Функциональный литерал	Функциональный литерал

Как видно из табл. 15.1, метод назначается одним из двух способов: либо путем объявления функции, либо с помощью функционального литерала. Каждый из способов можно использовать как для создания локального метода, хранящегося в **каждом** экземпляре класса, так и для создания коллективного метода, сохраняющегося только один раз в объекте **prototype**.

На практике почти никогда не существует причины, по которой каждый объект должен был бы обладать своей собственной копией метода. Объект должен иметь собственную копию свойства, только если ему необходимо иметь для него индивидуальное значение. Например, пусть существует много объектов различных цветов, принадлежащих классу **Dog**, и вы не хотите, чтобы всем членам этого класса было предоставлено в коллективное использование только одно свойство **col**. Фактически это означало бы наличие только одного значения этого свойства, т.е. одного цвета. Однако для всех объектов класса метод почти всегда является идентичным. Когда ситуация является подходящей, метод может использоваться коллективно и принадлежать объекту **prototype** функции-конструктора. В этой главе рассматривается только назначение методов для объекта **prototype**.

Совет

Процедура назначения свойств отдельным экземплярам продемонстрирована в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

Как видно из табл. 15.1, метод можно назначить посредством объявления функции либо с помощью функционального литерала. Давайте сначала рассмотрим синтаксис **каждого** из этих способов, а потом обсудим, в каких случаях лучше применять тот или иной подход.

НАЗНАЧЕНИЕ МЕТОДА ПОСРЕДСТВОМ ОБЪЯВЛЕНИЯ ФУНКЦИИ

Назначение метода для объекта **prototype**, начинающегося с объявления обычной функции, проходит в два этапа. Сначала выполняется объявление функции (в нашем примере это функция **whatColorAmI()**). Затем для свойства объекта конструктора **prototype** (в нашем примере это **getColor**) назначается функция.

```
// сначала объявляется функция
function whatColorAmI () {
return this.col;
}
// затем для свойства в прототипе назначается идентификатор функции
Ball.prototype.getColor = whatColorAmI;
// это работает!
trace(myBall.getColor()); // отображается "blue"
```

Обратите внимание, что при назначении функции не используются круглые скобки:

```
Ball.prototype.getColor = whatColorAmI;
```

Круглые скобки используются только при *исполнении* функции:

```
myBall.getColor();
```

Кроме того, обратите внимание, что в функции можно использовать ключевое слово **this**, которое автоматически будет относиться к объекту, метод которого активизируется. Это означает, что можно без труда назначать одну и ту же функцию для нескольких объектов. Например, в приведенном ниже листинге создается функция-конструктор **Cat**, в прототипе **Cat** создается свойство **getColor**, для которого назначается функция **whatColorAmI()**. Обратите внимание, что это та же самая функция, которая использовалась для класса **Ball**:

```
function Cat (col) {
    this.col = col;
}
```

```
Cat.prototype.getColor = whatColorAmI;
// создание Объекта Cat серого цвета
myCat = new Cat("gray");
// проверить цвет myCat
trace(myCat.getColor()); // отображается "gray"
```

НАЗНАЧЕНИЕ МЕТОДА с помощью ФУНКЦИОНАЛЬНОГО ЛИТЕРАЛА

Назначить метод для объекта `prototype` можно с помощью всего одного шага — используя функциональный литерал. Рассмотрим, например, такой функциональный литерал:

```
function () {
    return this.col;
};
```

То же самое можно записать одной строкой:

```
function O {return this.col;};
```

Сразу после создания функциональный литерал нужно где-то сохранить, иначе он будет потерян. В данном случае "где-то" является свойством объекта `prototype`:

```
// назначение функционального литерала для свойства getColor
Ball.prototype.getColor = function () {return this.col;};
// это работает!
trace(myBall.getColor()); // отображается "blue"
```

Ниже приведено несколько других примеров назначения свойств с помощью функциональных литералов. Во втором примере имеется аргумент `col`, который позволяет задать новый цвет мяча.

```
Ball.prototype.getClassName = function () {return this.className;};
Ball.prototype.setColor = function (col) {this.col = col;};
trace(myBall.getClassName()); // отображается "Ball"
myBall.setColor("red");
trace(myBall.getColor()); // отображается "red"
```

СЛУЧАИ ПРИМЕНЕНИЯ ФУНКЦИОНАЛЬНОГО ЛИТЕРАЛА И ОБЪЯВЛЕНИЯ ФУНКЦИИ

Если вы собираетесь назначить функцию только для одного класса и не используете ее в любом другом контексте, то использование функционального литерала будет более эффективным. В этом случае объявление функции только добавит строки кода в программу.

С другой стороны, если вы назначаете одну и ту же функцию для нескольких классов или используете ее в каком-либо другом контексте, лучше объявить функцию в одном месте и обращаться к ней каждый раз, когда ее необходимо использовать. Если в такой ситуации воспользоваться функциональным литералом, будет задействовано дополнительное количество памяти, что не является необходимым, поскольку в каждом классе будет храниться своя копия функции.

ПРИМЕНЕНИЕ НАСЛЕДОВАНИЯ для СОЗДАНИЯ ПОВТОРНО ИСПОЛЬЗУЕМОГО КОДА

Возможность одного объекта (такого как `myBall`) эффективно заимствовать свойства другого объекта (объекта `prototype` конструктора) лежит в основе ключевой характеристики объектно-ориентированного программирования — наследования.

Основная идея наследования очень проста: поскольку объект, подобный `myBall`, может эффективно заимствовать свойства из объекта `prototype` функции-конструктора, то и один класс может заимствовать свойства другого. В **ActionScript** функция-конструктор является

олицетворением класса, поэтому наследование является процедурой заимствования свойств одной функцией-конструктором у другой. Говоря точнее, она заключается в том, что один объект `prototype` одной функции-конструктора заимствует свойства другого объекта `prototype` у другой функции.

Несмотря на схожесть способов, которыми заимствуют свойства объекты и классы, обратите внимание на то, что нигде не говорится о том, что объект "наследует" свойства класса. В языках, имеющих статически определенные классы, таких как Java, следует избегать фразы о том, что объект "наследует" свойства класса. Наследование — это нечто, происходящее только между классами. Когда объект является *реализованным* (созданным как экземпляр класса), функция-конструктор назначает для него различные свойства. Некоторые свойства могут быть унаследованы классом до того, как он был назначен для объекта. Но сами по себе объекты ничего не "наследуют".

Это совершенно четкое отличие не очень хорошо отражает реалии языка `ActionScript`. В `ActionScript` механизм, согласно которому объект (такой как `myBall`) получает какие-то полезные свойства без конкретного выполнения задачи, а просто благодаря обстоятельствам своего рождения, является тем же самым, что и механизм передачи свойств от класса к классу. Этим механизмом является объект `prototype`. Экземпляры и классы получают "заимствованные" свойства одним и тем же способом, поэтому иногда использование термина *наследование* в обеих ситуациях является естественным.

Взаимосвязь между классами, которые заимствуют свойства, и классом, который их передает, является отношением типа "родитель-потомок". Объект `prototype` функции-конструктора заимствует свойства экземплярам класса аналогичным способом. В `ActionScript` взаимосвязь типа "прототип-экземпляр" удобнее называть связью типа "родитель-потомок".

Любое отношение типа "родитель-потомок" является двухуровневой иерархической структурой. На верхнем уровне находится родительский объект, на нижнем уровне — один или несколько дочерних объектов. В `ActionScript` родительский объект может иметь любое количество дочерних объектов, но каждый дочерний объект может иметь только один родительский.

Можно создать цепочки наследования, содержащие несколько поколений. Родительские объекты могут иметь свои родительские объекты, а дочерние — свои дочерние объекты. Чуть позже вы узнаете, как это сделать.

Однако сначала подумаем, для чего мы хотим создать такую цепочку. На минутку вернувшись к функции `Ball()`, подумайте, почему вы хотите использовать функцию-конструктор для создания мячей? Ведь для того, чтобы просто создать мяч, обладающий рядом свойств, придется затратить массу усилий. **Фактически**, если требуется просто создать парочку мячей с несколькими свойствами, применение объектно-ориентированного программирования, вероятно, является излишним. С другой стороны, чем больше мячей необходимо создать, тем удобнее будет сделать это с помощью одной строки кода для каждого мяча. Иными словами, ценность функции-конструктора увеличивается по мере увеличения ее повторного использования в программе.

Если вам понадобятся мячи в другой программе, то время и усилия, потраченные на создание функции-конструктора `Ball()`, будут вполне оправданы. Таким образом, ценность функции-конструктора увеличивается и по мере увеличения ее повторного использования в других программах.

Вы можете позволить другим программистам использовать свою функцию в их программах. При этом программистам не нужно будет анализировать всю внутреннюю разработку функции-конструктора. **Все**, что им нужно, — это понимать простые взаимодействия, реализованные в методе. Таким образом, затратив время и силы на исходном этапе, вы создадите доступный для неоднократного использования и, следовательно, ценный объект.

Подтверждением последней концепции неоднократного использования является то, что компания `Macromedia` проделала с `ActionScript`: она создала ряд различных классов, которые, скорее всего, будут востребованы широким кругом `Flash`-программистов. Теперь вы, как программист, можете взять эти классы и использовать их в своих собственных программах.

Классы можно использовать в том виде, в каком они есть. Можно также изменить или расширить их в соответствии со своими потребностями. Неоднократное использование объектов такого рода является наиболее упоминаемым преимуществом объектно-ориентированного программирования.

Однако не все классы имеют равный потенциал повторного использования. Следовательно, у программистов есть стимул для создания широко применяемых классов. Это и является основной мотивацией создания цепочек наследования, состоящих из нескольких поколений, поскольку это позволяет создавать обобщенные классы, широко доступные для неоднократного использования.

Хорошими примерами этой стратегии являются компоненты пользовательского интерфейса, такие как полоса прокрутки, комбинированное окно и поле списка, которые Macromedia предоставляет в комплекте Flash MX. Например, если перетащить объект `ListBox` в рабочее поле и воспользоваться панелью `Movie Explorer` (Проводник фильма) для просмотра его кода `ActionScript`, вы обнаружите комментарии, подобные приведенным ниже.

```
FSelectableListClass
    EXTENDS FUIComponentClass
CLASS FSelectableItemClass
    EXTENDS FUIComponentClass
FListBoxClass
    EXTENDS FSelectableListClass
    IMPLEMENTS FListItemClass
FScrollSelectListClass
    EXTENDS FSelectableListClass
FListItemClass
    EXTENDS FSelectableItemClass
```

Эти комментарии отображают трехъярусную иерархию наследования.

- Ярус 1 — вверху находится `FUIComponentClass`.
- Ярус 2 — `FSelectableListClass` и `FSelectableItemClass` наследуют свойства `FUIComponentClass`.
- Ярус 3 — `FScrollSelectListClass` наследует свойства `FSelectableListClass`, а `FListItemClass` — свойства `FSelectableItemClass`.

Свойства, необходимые для всех компонентов, можно поместить в `FUIComponentClass` и предоставить в коллективное использование всем классам данной иерархической структуры. Свойства, необходимые для всех списков, в которых нужно выбирать элементы, можно поместить в `FSelectableListClass` и предоставить в коллективное использование объектам класса `FScrollSelectListClass`. И наконец, свойства, необходимые для всех выбираемых элементов, можно поместить в `FSelectableItemClass` и предоставить в коллективное использование объектам класса `FListItemClass`.

Объект, созданный с помощью функции-конструктора класса, будет иметь общие свойства по всей цепочке наследования, обратно до класса `FUIComponentClass`. Например, объект класса `FListItemClass` будет иметь свойства, общие с `FSelectableItemClass` и `FUIComponentClass`.

ИСПОЛЬЗОВАНИЕ ОПЕРАТОРА NEW для СОЗДАНИЯ ЦЕПОЧЕК

НАСЛЕДОВАНИЯ

Иерархию наследования можно задавать по собственному усмотрению, создавая классы или повторно используя уже существующие. Например, если вы хотите создать собственные компоненты, то можете взять за основу и повторно использовать существующие классы Macromedia.

Иерархические структуры из нескольких поколений можно создавать двумя способами. Если рассматривать объекты `prototype` в качестве связующих звеньев между родительскими и дочерними элементами, то рассматриваемый в этой главе подход заключается в создании новых цепочек. Это стандартный и рекомендуемый способ начала процесса.

Совет

Другой подход, который заключается в выполнении повторной компоновки существующих цепочек, рассматривается в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

Предположим, что имеются четыре функции-конструктора: `Bird` (Птица), `Raptor` (Хищник), `Eagle` (Орел) и `BaldEagle` (Белоголовый орел). Очевидно, что эти четыре класса находятся в иерархической взаимосвязи. Как связать их в структуру типа "родитель—потомок"?

Вы уже видели, каким образом объекты привязываются к объекту `prototype` их функций-конструкторов. Это происходит автоматически, при создании экземпляра объекта с помощью оператора `new`. Но вы еще не пытались создавать никаких экземпляров. Вы только хотите объединить существующие классы в иерархическую структуру.

Начните с рассмотрения двух верхних классов иерархии: `Bird` и `Raptor`. На минутку сделайте вид, что хотите создать несколько экземпляров в классе `Raptor`, и хотите, чтобы каждый из них обладал всеми свойствами, присущими классу `Bird`. Этого можно достичь, если поместить все свойства класса `Bird` в объект `prototype` класса `Raptor`. По существу, это означает, что вы хотите, чтобы объект `Raptor.prototype` стал членом класса `Bird`. Иными словами, вы хотите сказать, что

```
Raptor.prototype = new Bird();
```

Будет ли это работать? Да! Этот подход может показаться немного странным, потому что до сих пор мы рассматривали прототип только как нечто, создаваемое автоматически при создании функции. В данном же случае новый объект-прототип создается как член существующего класса.

Теперь разберемся в деталях. Ниже приведены функция-конструктор `Bird` и предложение, назначающее свойство `movement` для объекта `Bird.prototype`.

```
function Bird ( ) {  
    // локальное свойство "covering"  
    this.covering = "feathers";  
}  
// коллективное свойство "movement"  
Bird.prototype.movement="fly";
```

Любой объект, созданный с помощью функции `Bird`, будет иметь два свойства:

- локальное свойство, `covering = "feathers"`;
- коллективное свойство, `movement="fly"`.

Ниже приведена функция-конструктор `Raptor`:

```
function Raptor ( ) {  
    // здесь указывается предложение  
}
```

В данный момент функция-конструктор `Raptor` представляет собой всего лишь пустую оболочку. Несмотря на это, как и все остальные функции, `Raptor` уже имеет объект прототипа, который был задан при создании функции. Macromedia не сделала этот факт очевидным. Так, например, в цикле `for-in` объект `prototype` не отображается:

```
for a ( in Raptor) trace(a) ,• // ничего не отображается
```

Несмотря на это, объект-прототип имеется, и это можно подтвердить, используя недокументированную функцию `ASSetPropFlags()` (установить метки свойств `ActionScript`):

```
ASSetPropFlags(Raptor, null, 8, 1); // "открыть" скрытые свойства
for (a in Raptor) trace(a); // отображается "prototype"
```

В объекте `Raptor.prototype` пока что ничего не появилось. Это можно подтвердить путем применения цикла `for-in` в следующем образом:

```
for a ( in Raptor.prototype) trace(a); // ничего не отображается
```

Совет

Подробная информация о скрытых свойствах прототипа приведена в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

Необходимо создать новый объект-прототип для функции `Raptor` — тот, что обладает всеми свойствами класса `Bird`. С этой целью используется предложение `Raptor.prototype = new Bird()`. Теперь, если проверить содержимое объекта `Raptor.prototype`, даже без применения функции `ASSetPropFlags()`, вы обнаружите два свойства, унаследованных от класса `Bird`:

```
for (a in Raptor.prototype) trace(a); // "movement" "covering"
```

Поставленная цель достигнута! Теперь любой объект, созданный с помощью функции-конструктора `Raptor`, будет обладать свойствами класса `Bird`.

Вспомните, что вы намеревались создать экземпляры класса `Raptor`. В реальности `Raptor` является все еще слишком обобщенным классом для того, чтобы иметь экземпляры. Сначала необходимо установить связь класса `Eagle` с классом `Raptor` таким же образом, как только что задавалась связь между `Raptor` и `Bird`. Затем надо установить связь `BaldEagle` с `Eagle`. И наконец, создаются экземпляры класса `BaldEagle`. Ниже приводится костяк программы, которая выполняет все эти операции и выполняет проверку работы кода.

```
// функция-конструктор Bird
function Bird ( ) {
    // локальное свойство "covering"
    this.covering = "feathers";
}
Bird.prototype.movement="fly"; // коллективное свойство "movement"

// функция-конструктор Raptor
function Raptor ( ) {
    // здесь указывается предложение
}
Raptor.prototype = new Bird();
// Raptor наследует свойства класса Bird

// функция-конструктор Eagle
function Eagle ( ) {
    // здесь указывается предложение
}
Eagle.prototype = new Raptor();
// Eagle наследует свойства класса Raptor

// функция-конструктор BaldEagle
function BaldEagle ( ) {
    // здесь указывается предложение
}
BaldEagle.prototype = new Eagle();
// BaldEagle наследует свойства класса Eagle
```

```
baldEagle1 = new BaldEagle(); // создание экземпляра BaldEagle
for (a in baldEagle1) trace(a); // отображает "movement" и "covering"
trace(baldEagle1.movement); // отображает "fly"
trace(baldEagle1.covering); // отображает "feathers"
```

Последние три строки программы показывают, как была достигнута поставленная цель: все свойства класса `Bird` переданы вниз по цепочке `Raptor`, `Eagle` и `BaldEagle` и появляются в экземплярах класса `BaldEagle`.

Данная цепочка наследования является всего лишь основой, поскольку внутри трех нижних классов не добавлены или не удалены никакие свойства. В реальной программе каждый класс для подтверждения ценности своего существования должен был бы выполнять что-то полезное. В данном примере можно было бы добавить какие-то свойства, характерные для того или иного подкласса. Например, хищники (`Raptor`) обладают признаками, которые не являются характерными для всех птиц (`Bird`), орлы (`Eagle`) обладают признаками, которые не являются присущими для всех хищников, а белоголовые орлы (`BaldEagle`) обладают признаками, не являющиеся характерными для всех орлов.

Совет

Цепочки наследования рассматриваются также в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

ФУНКЦИИ КАК ДАННЫЕ

Каждая функция является объектом и, следовательно, может иметь свойства. Так, приведенный ниже пример показывает, как можно объявить функцию `myFunc()` и задать ее свойство `counter`, которое устанавливается в значение, равное 0.

```
function myFunc () {}
myFunc.counter = 0;
```

Как правило, свойства функций не назначаются, но они могут понадобиться при работе с классами. Естественным местом хранения данных о классе является функция-конструктор. В приведенном ниже примере `myClass()` является функцией-конструктором. Свойство `myClass.counter` может, например, использоваться для подсчета количества созданных экземпляров этого класса:

```
function MyClass () {}
MyClass.counter = 0;
```

Совет

Методика хранения данных в функции-конструкторе описывается в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

МЕТОДЫ `FUNCTION.APPLY()` И `FUNCTION.CALL()`



Методы `apply()` и `call()`, определенные для всех функций, позволяют обратиться к функции или методу, как если бы это был метод любого назначенного объекта. Объект указывается как первый аргумент метода `apply()` или `call()`. Внутри активизируемой функции назначенный объект обозначает ключевое слово `this`. При обращении к функции или методу без использования `apply()` или `call()` вы будете привязаны к назначенному для `this` значению по умолчанию. Например, при обращении к методу `this` относится к объекту, к которому принадлежит данный метод. Если выполняется вызов функции на временной шкале, то `this` относится к этой временной шкале.

С помощью методов `apply()` и `call()` можно сделать так, что ключевое слово **this** будет относиться к любому необходимому объекту.

Вызываемой функции можно передавать любое количество параметров. Единственное отличие между методами `apply()` и `call()` заключается в том, что при использовании метода `apply()` параметры передаются в виде массива, а при использовании метода `call()` выполняется передача списка, разделенного запятыми.

Формат методов выглядит следующим образом:

```
myFunction.apply(myObject, myArray);
myFunction.call(myObject, myArg1, myArg2, myArg3);
```

Методы `apply()` и `call()` будут получать значения, которые функция возвращает при обычных условиях.

Если первым аргументом является `null`, то будет использоваться значение **this**, заданное по умолчанию. В этом случае методы `apply()` и `call()` ведут себя как обычные обращения к функции за исключением того, что метод `apply()` передает параметры в массиве.

Чтобы показать, что вы сознательно не передаете никаких параметров, второй аргумент также можно задать равным `null`. При использовании этой методики достигается тот же эффект, что и при полном пропуске второго аргумента.

В приведенном ниже примере метод `apply()` используется для сброса счетчиков в трех классах.

Сначала объявим функцию `resetCounter()`, предназначенную для установки свойства `counter` (счетчик) какого-то объекта (представленного ключевым словом **this**) в значение 0.

Затем создадим три класса — `Class1`, `Class2` и `Class3` — и зададим для каждого из них свойство `counter`. Установим свойства `counter` в ненулевые значения так, чтобы функция `resetCounter()` могла выполнить необходимую операцию.

И наконец, применим цикл `for` и метод `apply()` для передачи классов, одного за другим, в функцию `resetCounter()`. Второй аргумент метода `apply()` установим равным `null`, поскольку функция `resetCounter()` не принимает никаких аргументов.

```
// объявление функции resetCounter
function resetCounter () {
    this.counter = 0;
}
// создание трех классов, каждый из которых обладает свойством
"counter"
function Class1() {}
Class1.counter = 1;
function Class2() {}
Class2.counter = 2;
function Class3() {}
Class3.counter = 3;
// использование метода apply() для передачи классов,
одного за другим, в
функцию resetCounter()
for (i = 1; i < 4; i++) {
    className = eval("Class"+i); // образование имени класса
    resetCounter.apply(className, null);
}
```


В приведенном ниже примере в каждом классе создаются три свойства `counter`. Свойство `counter1` устанавливается в значение 1, свойство `counter2` — в значение 2, а `counter3` — в значение 3. Значения счетчиков передаются в функцию `setCounters()` в виде массива в качестве второго аргумента метода `apply()`. Функция `setCounters()` извлекает эти значения с помощью свойства `arguments`, в котором хранятся переданные функции аргументы.

Совет

Объект `arguments` был рассмотрен ранее в этой главе, в разделе "Передача данных в функции с помощью аргументов".

```
// определение функции
function setCounters () {
    this.counter1 = arguments[0];
    this.counter2 = arguments[1];
    this.counter3 = arguments[2];
}
// определение классов
function Class1() {}
function Class2() {}
function Class3() {}
// создание и инициализация трех счетчиков в каждом классе
for (i = 1; i < 4; i++) {
    className = eval("Class"+i);
    setCounters.apply(className, [1,2,3]);
}
```

Метод `apply()` не поддерживает ассоциативных массивов в качестве своего второго аргумента. Именованные свойства ассоциативного массива будут утеряны, а вызываемая функция получит только элементы с числовыми индексами.

Совет

Дополнительная информация об ассоциативных массивах представлена в главе 19 "Использование встроенных базовых объектов".

Свойство `arguments` функции можно использовать в качестве второго аргумента метода `apply()`. Так, в приведенном ниже примере сначала вызывается функция `myHandler()` с двумя аргументами: 1 и 2. Функция `myHandler()`, в свою очередь, вызывает функцию `myDelegate()`. С помощью свойства `arguments` функция `myHandler()` "пересылает" свои аргументы в качестве второго аргумента метода `apply()`.

Обратите внимание на то, как возвращается значение от функции `myDelegate()` к функции `myHandler()` и обратно.

Ключевое слово `this` функции `myDelegate()` относится к временной шкале, поскольку первым аргументом метода `apply()` было значение `null`. Таким образом, в последнем действии `trace` величины `p0` и `p1` показаны как обычные переменные временной шкалы.

```
function myDelegate () {
    this.p0 = arguments[0];
    this.p1 = arguments[1];
    return true;
}
function myHandler() {
    ret = myDelegate.apply (null, arguments); // "пересылка" аргументов
    return ret;
}
result = myHandler(1,2);
trace (result); // true
trace (p0 + " " + p1); // 1 2
```

ЯВНЫЙ ОБЗОР ДАННЫХ

Обзор данных представляет собой процедуру сообщения интерпретирующей программе **ActionScript** о том, где следует искать переменную, видеоклип или массив. Интерпретирующая программа может выполнять поиск в *функции*, в *объекте*, во *временной шкале*, в объекте *Object* или в идентификаторе `_global`. Можно непосредственно указать определенное место поиска, и интерпретирующая программа будет осуществлять его именно там. В этом и заключается *явный обзор данных*, являющийся темой данного подраздела. Если сознательно не сообщать интерпретирующей программе, где следует выполнять поиск, то она будет следовать собственным правилам автоматического обзора данных. Этот тип поиска рассмотрен ниже, в разделе "Автоматический обзор данных".

На заметку

В данном подразделе термины *временная шкала* или *видеоклип* обозначают временную шкалу видеоклипа и любые присоединенные обработчики событий и кнопки.

Область действия данных похожа на сложную файловую систему. Ее можно использовать как для поиска данных, так и для сохранения их конфиденциальности. К сожалению, случается и пропуск файлов: когда интерпретирующая программа не может найти элемент или обнаруживает что-либо неверное, могут возникать неявные ошибки.

Совет

Подробная информация об идентификаторе `_global` приводится в главе 12 "Управление переменными, данными и типами данных".

Один из способов предотвращения подобной проблемы заключается в применении явного обзора данных для сообщения интерпретирующей программе **ActionScript** точного места поиска. При этом для сообщения интерпретирующей программе **ActionScript** места поиска элемента данных, к которому производится обращение, используется синтаксис с применением точек (или исключенный синтаксис с применением символов косой черты). Например:

```
_root.myClip.myFunc(); // сообщает интерпретирующей программе, где искать
                        // видеоклип "myClip"
_global.myVar = 10; // искать в _global
```

ОГРАНИЧЕНИЯ ЖЕСТКОГО КОДИРОВАНИЯ

Примером *жесткого кодирования* является использование имени видеоклипа, функции, объекта или идентификатора `_global` для указания *пути* к элементу данных, как в приведенных выше примерах. Однако четкие и легкие в употреблении жестко закодированные пути делают код менее пригодным к неоднократному использованию. Рассмотрим *пример* с указанием пути `_root.myClip.myFunc()`; . Если необходимо использовать функцию `myFunc()` в каком-то другом проекте, но не с видеоклипом `_root.myClip`, код нужно будет несколько видоизменить. То же самое справедливо и для второго примера, т.е. если вы хотите повторно использовать переменную `myVar`, но не хотите, чтобы она была глобальной.

Избежать жесткого кодирования можно с помощью *относительного обращения* или ключевого слова `this`. Еще одна возможность заключается в применении автоматического, а не явного обзора данных.

ЯВНЫЙ ОБЗОР ДАННЫХ С ПОМОЩЬЮ ОТНОСИТЕЛЬНЫХ ОБРАЩЕНИЙ

Если областью обзора данных является видеоклип, то наиболее распространенный способ повышения возможности неоднократного использования кода заключается в применении *относительных обращений*. Это означает, что для указания видеоклипа, в котором должен выполняться поиск, используется свойство `_parent` текущего видеоклипа. Например:

```
_parent.myFunc(); // выполнять поиск в родительском клипе  
_parent._parent.myFunc(); // выполнять поиск в "прародительском" клипе
```

Предположим, например, что функция `myFunc()` находится в видеоклипе `mc1` основной временной шкалы. При этом абсолютный путь к клипу будет следующим: `_root.mc1`. (Любой путь, начинающийся с префикса `_root`, называется "абсолютным путем", потому что он указывает местоположение видеоклипа, начиная с основной временной шкалы, а не относительно другого видеоклипа.) Теперь, если необходимо обратиться к функции `myFunc()` из дочернего клипа, скажем, из `_root.mc1.mc2`, можно указать путь `_parent.myFunc()`. При обращении из прадочернего клипа, например из `_root.mc1.mc2.mc3`, можно указать путь `_parent._parent.myFunc()`. Если использовать этот код с видеоклипами `myClip1`, `myClip2` и `myClip3`, то, чтобы применить его к другим именам экземпляров видеоклипов, менять уже ничего не нужно.

ЯВНЫЙ ОБЗОР ДАННЫХ С ПОМОЩЬЮ КЛЮЧЕВОГО СЛОВА `THIS`

В третьем типе явного обзора данных используется ключевое слово `this`, за единственным исключением: `this` относится к видеоклипу, в котором располагается предложение. В следующих подразделах приведено несколько примеров общего правила. После этого мы рассмотрим исключение.

ИСПОЛЬЗОВАНИЕ КЛЮЧЕВОГО СЛОВА `THIS`: ОСНОВЫ

Ключевое слово `this` обычно относится к временной шкале (в том числе с присоединенными обработчиками событий и кнопками). В видеоклипах ключевое слово `this` можно использовать для того, чтобы интерпретирующая программа `ActionScript` распознала, что выполняется обращение к методу `MovieClip`, а не к глобальной функции с тем же именем. Например, если поместить приведенный ниже код во временную шкалу, то интерпретирующая программа `ActionScript` посчитает, что вы обращаетесь к глобальной функции `duplicateMocieClip()`, и сгенерирует ошибку.

```
duplicateMovieClip("myClip", 1); // ОШИБКА!!
```

При вставке ключевого слова `this` будет активизироваться метод `MovieClip`, и код будет работать корректно:

```
this.duplicateMovieClip("myClip", 1); // все в порядке
```

Совет

Подробная информация о функции `duplicateMovieClip()` приведена в главе 17 "Демонстрация мощи видеоклипов".

В приведенном ниже примере показаны три варианта использования ключевого слова `this` во временной шкале для обзора данных.

- Переменная находится в свойстве объекта на временной шкале.
- Обращение к переменной выполняется внутри функции.
- Обращение к переменной производится в видеоклипе.

Три предложения `trace` в конце кода подтверждают, что в каждом случае интерпретирующая программа рассматривает `this` как временную шкалу. Если с равно 300, то можно сделать вывод о том, что данное предложение "видит" `a` и `b` на временной шкале. Если бы она видела свойства `a` и `b`, к примеру, в объекте `myObj`, то с равнялось бы 3, а не 300.

```

a = 100; // переменная объявлена во временной шкале видеоклипа
b = 200; // переменная объявлена во временной шкале видеоклипа
c = a + b; // переменная объявлена во временной шкале видеоклипа
myObj = {a : 1, b : 2, c : this.a + this.b} // #1 "this" в свойстве
function myFunc() {
    trace(this.c); // #2 "this" в функции
}
trace(this.c); // # 3 "this" во временной шкале - отображается 300
trace(myObj.c); // отображается 300 - в объекте
myFunc(); // отображается 300 - в функции

```

Областью действия переменной является временная шкала, в которой она находится, даже если она используется внутри объекта или функции. Кажется удивительным, но интерпретирующая программа не выполняет сначала проверку внутри объекта наличия свойств, совпадающих с вызываемым именем.

ИСКЛЮЧЕНИЕ: ОБЗОР ДАННЫХ В ОБЪЕКТЕ

В предыдущем подразделе мы увидели, что в большинстве ситуаций, если ключевое слово **this** указывалось в качестве пути к переменной на временной шкале, то под ним подразумевалась **временная** шкала. Из этого правила есть одно важное исключение — *в пределах метода объекта* ключевое слово **this** привязывается к объекту.

За исключением этого момента правила обзора данных для функций такие же, как и для переменных, т.е. на временной шкале (или соответствующих обработчиках событий и кнопках) или в пределах функции на временной шкале областью действия **this.myFunc()** является временная шкала.

Приведенный ниже код выполняет, в сущности, ту же операцию, что и код в предыдущем примере, но только вместо переменных используются функции. Четыре вызова функции, стоящие в конце кода (строки 10–13), говорят о том, что происходит. В строке 10 показано, что при обращении к **this.a** на временной шкале интерпретирующая программа ищет на временной шкале функцию **a()**. Аналогичным образом, интерпретирующая программа ищет на временной шкале функцию **a()** при обращении к ней внутри **b()** (строка 11). Третий вызов функции (строка 12) показывает, что активизация **this.a()** внутри функции **myFunc()** на временной шкале ничего не меняет. Интерпретирующая программа все равно ищет функцию **a()** на временной шкале. Однако последний вызов функции показывает, что когда **this.a()** активизируется внутри метода, интерпретирующая программа ищет функцию **a()** в объекте, которому принадлежит данный метод.

```

1 function a 0 {trace ("function a");}
2: function b 0 {this.a();} // областью действия "this" является
                           // временная шкала
3: myObj = {
4: a : function () {trace ("object function a");},
5: b : function () {this.a();} // областью действия "this" является объект!
6: };
7: function myFunc() {
8: this.a(); // областью действия "this" является временная шкала
9: }
10: this.a(); // отображается "function a" - областью действия "this"
           // является временная шкала
11: b(); //отображается "function a"
12: myFunc(); // отображается "function a"
13: myObj.b(); // отображается "object function a"

```

ПСЕВДОНИМЫ ФУНКЦИЙ СОХРАНЯЮТ СВОЮ СОБСТВЕННУЮ ОБЛАСТЬ ДЕЙСТВИЯ

Если задать переменную равной функции или методу, то она сохранит свою собственную область действия. Она *не* принимает область действия функции или метода. Так, в приведенном ниже примере переменная `myAliasFunc` устанавливается равной `myObj.myFunc`. Исполнение `myObj.myFunc()` отображает `prop1` корректно, поскольку используется метод, областью действия которого является объект. Однако при исполнении функции `myAliasFunc()` получим значение `undefined` (или пустую строку во Flash 5), так как областью действия переменной `myAliasFunc` является *временная* шкала, в которой нет такой переменной, как `prop1`.

```
myObj = new Object();
myObj.myFunc = function () {
    trace(this.prop1);
}
myObj.prop1 = 1;
myAliasFunc = myObj.myFunc;
myObj.myFunc(); // отображается "1"
myAliasFunc(); // undefined, свойство this.prop1 во временной шкале не
                // определено
```

АВТОМАТИЧЕСКИЙ ОБЗОР ДАННЫХ

При автоматическом обзоре данных вы просто указываете прямую ссылку на переменную, функцию, объект или массив, без применения точечного синтаксиса. При этом интерпретирующая программа ActionScript выполняет обзор данных автоматически.

Например, ниже приводится прямая ссылка.

```
myFunc(); // интерпретирующей программе не сообщается, где следует искать
          // функцию myFunc()
```

В этом случае интерпретирующая программа начинает поиск там, где сделана ссылка на данные. Оттуда может выполняться поиск во временной шкале видеоклипа (в том числе в связанных с ней обработчиках событий и кнопках), в функциях или объектах, в том числе и в цепочках наследования. И, наконец, последнее место, в котором проводится поиск, — это идентификатор `_global`.

По умолчанию *цепочки обзора* ActionScript очень просты. Цепочка обзора представляет собой список объектов, в которых следует проводить поиск. Если с помощью методик объектно-ориентированного программирования, описанных ранее в этой главе, не была создана более сложная структура, то, как правило, цепочкой обзора является последовательность `_global-Object-временная шкала` или `_global-Object-временная шкала-функция`. Поиск в цепочках выполняется справа налево, т.е. сначала выполняется поиск в функции, затем — во временной шкале, потом — в объекте `Object` и наконец — в `_global`.

Встречаются и более длинные цепочки, такие как `_global-Object-временная шкала-внешняя функция-внутренняя функция`, когда внутренняя функция объявляется внутри внешней.

Цепочка обзора временно удлинится при использовании предложения `with`, которое добавляет еще один объект на самом нижнем уровне цепочки. Этот добавленный объект (объект внутри предложения `with`) всегда проверяется первым внутри предложения `with`. По завершении выполнения этого предложения он удаляется из цепочки обзора.

В следующих трех примерах показаны начальная точка (в которой объявлена переменная или функция) и типовая итоговая цепочка обзора.

- Начало на *временной шкале* (или в присоединенном к ней обработчике событий или кнопке).

- Пример:

```
myFunc(); // на временной шкале
```

- Цепочка обзора: `_global-Object-временная шкала`.

- Примечание: в цепочку обзора не входят другие временные зависимости.

- Начало в *свойстве объекта* или *элементе массива* на временной шкале.

- В этом примере переменная `myVar` объявлена на временной шкале и ссылается на свойство `myObj.myProp1`. Затем это свойство отображается двумя способами: с помощью метода объекта, `myMethod()`, определенного в строке 4 и вызываемого в строке 5, и с помощью предложения `trace` во временной шкале, в строке 6.

```
1: myVar = "main";
2: myObj = { myVar : "myVarObj",
3:           myProp1 : myVar,
4:           myMethod : function () {trace(this.myProp1);}};
5: myObj.myMethod(); // отображается "main"
6: trace(myObj.myProp1); // отображается "main"
```

- Цепочка обзора: `_global-Object-временная шкала`.

- Примечание 1. Если значением `myProp1` в строке 3 является `this.myVar` (обратите внимание на добавление ключевого слова `this`), то цепочкой обзора все равно будет последовательность `_global-Object-временная шкала`.

- Примечание 2. В цепочку обзора не входят "подчиненные" свойства или элементы. Например, в строке 2 содержится свойство `myObj.myVar`, но оно игнорируется.

- Начало в объявлении *функции* или *метода*.

- В этом примере переменная `myVar`, будучи объявленной во временной шкале (строка 1), сначала упоминается в объявлении функции во временной шкале (строка 3), а затем — в методе `myObj.myFunc()` (строка 5).

```
1: myVar = 10;
2: function myFunc() {
3:   trace(myVar);
4: }
5: myObj = {myVar : 99 , myFunc : function() { trace(myVar);}
}

6: myFunc(); // 10
7: myObj.myFunc(); // 10
```

- Цепочка обзора: `_global-Object-временная шкала-функция`.

- Примечание: в цепочку обзора не входят "подчиненные" свойства или элементы. Например, интерпретирующая программа игнорирует свойство `myVar : 99` (строка 5).



Последнее предложение, `myObj.myFunc()` (строка 7), работает идентично и во Flash 5, и во Flash MX, при условии, что оно находится в той же временной шкале, что и объявление объекта `myObj` над ним (строка 5). Однако при попытке выполнить метод `myObj.myFunc()` из другой временной шкалы, во Flash 5 код работать не будет (или же будет отображена переменная `myVar` другой временной шкалы, если таковая существует). Во Flash MX метод `myObj.myFunc()` можно выполнить из другой временной шкалы, и при этом будет отображена переменная `myVar` той временной шкалы, в которой был определен объект `myObj`.

Предположим, например, что приведенный выше код располагается в основной временной шкале и в ней же имеется клип `circle`, во временной шкале которого имеются следующие предложения:

```
myVar = "circle";
_parent.myObj.myFunc(); // во Flash 5 отображается "circle", а во
Flash MX - 10
```

При работе с Flash 5 в результате выполнения второго предложения отобразится "circle", а при использовании Flash MX— 10. Во Flash MX областью действия метода все время является временная шкала, в которой он был определен, и ею никогда не является временная шкала, из которой выполнялся вызов метода. (В этом месте Flash MX соответствует стандарту ECMA-262, а Flash 5 отходит от него.)



Нововведением Flash MX также является то, что функциональный литерал, заданный внутри функции, принимает область действия внешней функции. Во Flash 5 его областью действия являлась временная шкала.

В приведенном ниже примере функция `innerFunction()` относится к функциональному литералу внутри функции `outerFunction()`. Во Flash MX областью действия функционального литерала является функция `outerFunction`, что обеспечивает доступ к параметру `arg`. Следовательно, когда функциональный литерал возвращается в строку 4, он включает в себя аргумент, который отображается при выполнении функции в строке 5. Во Flash 5 областью действия функции `innerFunction()` является временная шкала, в которой аргумент `arg` не определен, поэтому в строке 5 будет отображено "Argument was: ".

```
1: function outerFunction (arg) {
2:   return innerFunction = function () {trace ("Argument was:
   "+arg);}
3: }
4: myFunc = outerFunction("myArg");
5: myFunc(); // во Flash MX отображается "Argument was: myArg"
```

Это пример цепочки обзора, состоящей из пяти звеньев: `_global—Object—временная шкала—внешняя функция—внутренняя функция`.

Приведенная ниже функция `makeHandler` показывает, как использовать эту новую функциональность для присоединения программным образом обработчиков событий к видеоклипам. Во Flash MX функция `makeHandler()` возвращает аргумент `name`. Во Flash 5 вернулось бы значение `undefined` (предполагается, что во временной шкале не объявлена переменная `name`), потому что внешняя функция не является областью действия внутренней функции.

```
// спасибо Gary Grossman, главному инженеру коллектива Macromedia
Flash!
function makeHandler(name) {
  return function () {
    trace("Handler "+ name +" invoked.");
  };
}
function makeHandlers(mc, names) {
  for (var i=0; i<names.length; i++) {
    mc[names[i]] = makeHandler(names[i]);
    mc[names[i]](); // отображается "Handler onPress invoked."
  }
}
etc.

makeHandlers(mc, ["onPress", "onRelease", "onReleaseOutside",
  "onRollOver", "onRollOut", "onDragOver", "onDragOut"]);
```

В разделе "Возможные проблемы" обсуждается такая возможность размещения на временной шкале, чтобы переменная или функция была не доступна даже в пределах одной временной шкалы.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему я не могу получить доступ к переменной или функции на одной и той же временной шкале?



Если воспроизводящая головка ни разу не доходила до кадра, содержащего определенную переменную или функцию, то интерпретирующая программа ActionScript о ней ничего не знает.

Пусть, например, в кадр 1 помещен следующий обработчик события:

```
_root.onEnterFrame = function() {
    trace(x);
};
```

Пусть в кадр 2 той же временной шкалы помещено предложение `x = 10`. В этом случае при первом исполнении (в первом кадре) предложение `trace(x)` в обработчике события будет отображать `undefined`, потому что переменная `x` еще не объявлена. После того как интерпретирующая программа "увидит" переменную `x` (т.е. после того, как воспроизводящая головка достигнет второго кадра), программа "запомнит" ее, и, начиная с этого момента и в дальнейшем, предложение `trace(x)` в обработчике события будет отображать "10" при достижении любого кадра, в том числе и кадра 1.

Знайте, что интерпретирующей программе ActionScript становятся известны все функции кадра, как только она его достигнет. Таким образом, функцию можно выполнять еще до ее объявления, в пределах одного кадра:

```
test(); // отображается "hi"
function test() {
    trace("hi");
}
```

Почему я получаю возвращаемое значение `undefined`?

Функции возвращают значение `undefined` в случае отсутствия предложения `return` или при его наличии, но без указания возвращаемого значения. Проверьте программу на наличие таких ситуаций. В обоих следующих случаях необходимо использовать предложение `return result`;

Нет предложения `return`:

```
function doubleIt(numberToBeDoubled) {
    result = numberToBeDoubled * 2;
}
```

Нет возвращаемого значения:

```
function doubleIt(numberToBeDoubled) {
    result = numberToBeDoubled * 2;
    return;
}
```


Часть моей функции никогда не выполняется. Почему?

Если часть функции никогда не выполняется, то, вероятно, это может быть потому, что в код внесено предложение `return`. Вот простой пример подобной проблемы:

```
1: function showRoomNumber(floor, num) {  
2:   if (num < 0)  
3:     err = "negative numbers not supported";  
4:     return(err);  
5:   return ("Your room number is "+floor+num); // эта строка никогда  
не выполняется  
6: }
```

Проблема возникает из-за того, что вокруг двух предложений, которые должны были бы являться кодом возврата ошибки (строки 3 и 4), нет фигурных скобок. Таким образом, строка 4 выполняется всегда, что приводит к возврату функции. Строка 3 пропускается, если `num` больше или равно 0. Если строка 3 не выполняется, то ошибка (`err`) не определяется и поэтому строка 4 возвращает величину `undefined`. При этом строка 5, которая должна вернуть номер комнаты (`room number`), вообще никогда не выполняется.

Функция должна выглядеть следующим образом:

```
function showRoomNumber(floor, num) {  
  if (num < 0) {  
    err = "negative numbers not supported";  
    return(err)  
  }  
  return ("Your room number is "+floor+num);  
}
```

FLASH ЗА РАБОТОЙ: РАСЧЕТ ФАКТОРИАЛОВ — ПРИМЕР РЕКУРСИИ

Рекурсию часто демонстрируют на примере функции, выполняющей расчет факториалов. Факториалы часто используются для того, чтобы определить, сколько различных способов имеется для упорядочивания или систематизации ряда событий.

Факториал числа вычисляется по формуле

(число) x (число - 1) x (число - 2)...

и т.д. до тех пор, пока не будет достигнут последний множитель, равный 1. Таким образом, факториал числа 4 равен:

$4 \times 3 \times 2 \times 1$,

или 24.

По определению факториалы 0 и 1 равны 1.

В виде псевдокода функцию `factorial()` можно представить следующим образом:

```
if (число < нуля) отклонить его!  
if (число не целое) округлить его до целого в сторону уменьшения  
if (числом является 0 или 1) факториал = 1  
if (число > 1) факториал = число x факториал(число - 1)
```

Последняя строка делает функцию рекурсивной. Чтобы получить факториал числа, это число нужно умножить на факториал числа, меньшего на 1. Таким образом, цикл начинается снова.

Вот как функция расчета факториала выглядит на языке **ActionScript**:

```

1: function factorial(myNumber){
2:   myNumber = Math.floor(myNumber); // округлить до ближайшего цело-
го в сторону уменьшения.
3:   if (myNumber < 0){ // Если число < 0, отклонить его.
4:     return ("факториалы отрицательных чисел не определяются");
5:   }
6:   if (myNumber < 2 ) { // Если числом является 0 или 1
7:     return 1; // факториал = 1. Также разрывается цикл рекурсии!
8:   }
9:   else return (myNumber * factorial(myNumber - 1)); // рекурсия
10: }
11: trace(factorial(4) ); // отображается "24"

```

Когда функция `factorial()` вызывается с аргументом 4, то первая строка кода, которая что-то делает, — это строка 9. Она начинает возвращать `4*factorial(3)`, но, естественно, интерпретирующая программа, прежде чем она сможет вернуть значение, должна вычислить `factorial(3)`.

Когда функция `factorial()` вызывается с аргументом 3, снова выполняется предложение `else` в строке 9. Функция начинает возвращать `3*factorial(2)`, но интерпретирующая программа теперь должна вычислить `factorial(2)`.

Функция `factorial()` вызывается с аргументом 2. В предложении `else` в строке 9 интерпретирующая программа возвращает `2*factorial(1)`, но сначала ей нужно рассчитать `factorial(1)`.

И наконец, когда интерпретирующая программа выполняет функцию `factorial(1)`, то предложение `if (myNumber < 2)` в строке 3 становится истинным. Функция возвращает значение 1.

Теперь интерпретирующая программа выполняет предложение `return`, которое было оставлено "в подвешенном состоянии". Самый последний цикл дает значение 2. Это приводит к тому, что предыдущий цикл дает значение 6, а цикл перед ним — 24, которое и является значением, возвращаемым функцией `factorial()`.

ВЗАИМОДЕЙСТВИЕ, СОБЫТИЯ И УСТАНОВЛЕНИЕ ПОСЛЕДОВАТЕЛЬНОСТИ

В ЭТОЙ ГЛАВЕ...

Иницилирующее действие: временная шкала и обработчики событий	329
Создание сценариев на основе временной шкалы	330
Создание сценариев на основе событий	331
Установление последовательности действий с помощью порядка выполнения	353
Возможные проблемы	354
Flash за работой: универсальные процессоры событий Flash	354

ИНИЦИИРУЮЩЕЕ ДЕЙСТВИЕ: ВРЕМЕННАЯ ШКАЛА И ОБРАБОТЧИКИ СОБЫТИЙ

Интерпретирующая программа **ActionScript** по своей природе линейна. Если ей не давать никаких специальных распоряжений, она просто будет последовательно выполнять предложения. В предыдущей главе мы увидели, что вызовы функций позволяют нарушить такую последовательность. Они позволяют прервать последовательный ход выполнения программы, выполнить ряд предложений, определенных где-то в другом месте, а затем вернуться в то место, где последовательное выполнение программы было прервано.

Совет

Подробная информация о вызове функций приведена в главе 15 "Объединение предложений в функции".

В данной главе рассматриваются два других способа изменения хода выполнения программы. В одном из них используется **временная шкала**, а в другом — **события**. Каждый объект имеет определенные события, наступления которых он ожидает. Например, всякий раз, когда пользователь **щелкает** на кнопке, она генерирует событие `onPress`. Чтобы отследить событие, необходимы специальные функции, именуемые *обработчиками событий*. Схема, работающая на основе временной шкалы и также называемая "кадровой", основана на том, что при воспроизведении кадра интерпретирующая программа исполняет находящийся в нем код.

Совет

Еще один способ изменения хода выполнения программы заключается в использовании глобальной функции `setInterval()`, которая описывается в главе 17 "Демонстрация мощи видеоклипов".

Источником возможной путаницы является тот факт, что наиболее распространенные события также часто бывают связаны с кадрами. Так, всякий раз по достижении кадра видеоклип генерирует событие `onEnterFrame`. Событие `onEnterFrame` позволяет выполнять один и тот же код всякий раз, когда определенный видеоклип достигает кадра, *без* помещения этого кода в каждый кадр. В отличие от такого подхода, при использовании кадрового программирования выполняется тот код, который *находится* в кадре, когда воспроизводящая головка достигает его.

Совет

Отличия между кадровым и событийным управлением ходом выполнения программы описаны в главе 11 "Знакомство с ActionScript".

Большая часть этой главы посвящена программированию, основанному на событиях. Однако сначала коротко рассмотрим методику программирования, основанного на использовании временной шкалы.

СОЗДАНИЕ СЦЕНАРИЕВ НА ОСНОВЕ ВРЕМЕННОЙ ШКАЛЫ

Иногда **временная шкала** является самым естественным средством систематизации программ. Хорошим примером является **раскрывающееся** меню, которое в свернутом состоянии находится в одном кадре (например, в кадре 1), а в развернутом — в другом (например, в кадре 2). Примером такого меню может служить файл `timeline.fl`, находящийся на прилагаемом к книге компакт-диске. На рис. 16.1 и 16.2 показано развернутое состояние меню в том **виде**, в каком оно отображается соответственно в SWF- и во FLA-файлах.

Совет

Временная шкала описывается в главе 2 "Интерфейс Flash".

Если действие связано с одним из состояний меню (свернутым или развернутым), то имеет смысл поместить его в кадр с соответствующим состоянием.

Например, на рис. 16.1 показано приложение "*Картинная галерея*", в котором каждый элемент меню позволяет пользователю выбрать и отобразить определенную репродукцию картины. Когда меню разворачивается, отображаемая в данный момент **репродукция** "затеняется" (коэффициент прозрачности `_alpha` устанавливается равным 50) так, что она меньше отвлекает внимание.

На рис. 16.2 изображен файл FLA этого же приложения. Временная шкала содержит **двух-кадровое** всплывающее меню, а воспроизводящая головка находится в кадре 2 слоя 1. Свернутое состояние меню помещено в кадр 1, а развернутое — в кадр 2. Когда меню разворачивается (путем перехода к кадру 2), происходит активизация кода в этом кадре, т.е. `currentPicture._alpha = 50;`, и текущая репродукция затеняется.

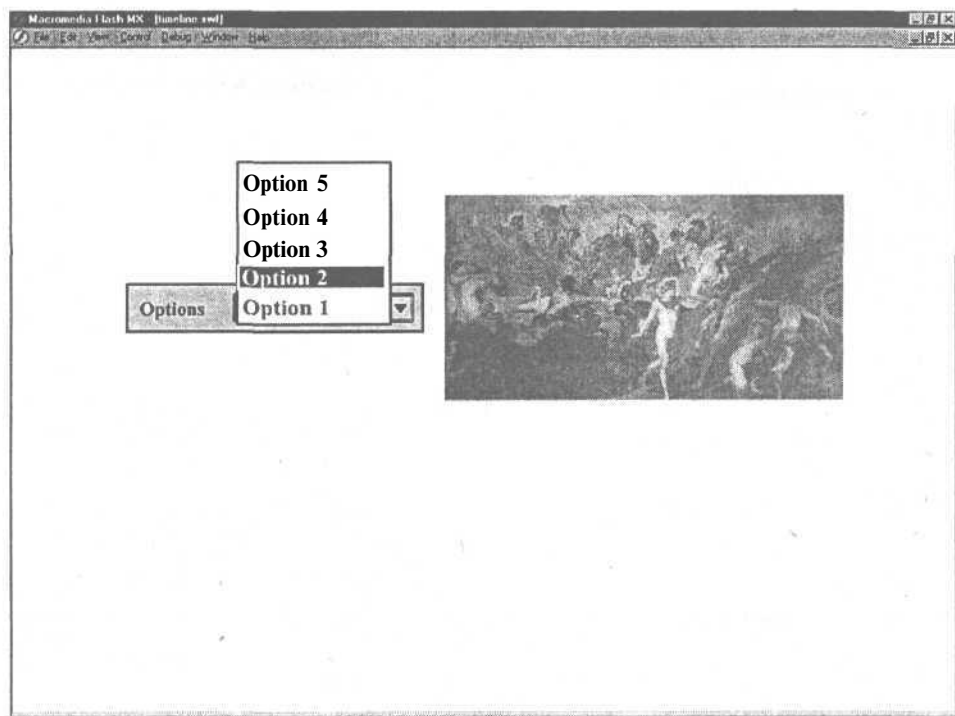


Рис. 16.1. При использовании кадрового программирования развернутое состояние раскрывающегося меню достигается за счет перемещения воспроизводящей головки в кадр 2

Кроме того, при наличии кода, выполняющегося последовательно, но логически состоящего из двух или нескольких отдельных частей, для его компоновки можно использовать временную шкалу. Если каждую часть кода поместить в отдельный кадр верхнего слоя и пометить каждый кадр, несущий код, то размещение кода и его базовая структура будут сразу же понятны любому пользователю, открывшему файл. На рис. 16.3 изображена временная шкала файла `MovieClass.fla`, автором которого является разработчик Flash Робин Дебреуил (Robin Debreuil). (Не беспокойтесь о том, для чего предназначен файл `MovieClass.fla`. Он приведен здесь только для того, чтобы продемонстрировать компоновку кода. Если вы хотите поближе познакомиться с этим файлом, то найдете его на компакт-диске, прилагаемом к книге.)

Помещая код в различные кадры (в идеальном случае они должны иметь описательные метки) и отправляя воспроизводящую головку в определенный кадр в нужный момент времени, можно создавать достаточно сложные приложения. До выхода версии Flash 5 это был единственный способ моделирования вызовов функций во Flash. С появлением реальных функций в версии Flash 5 управление ходом выполнения программы на основе временной шкалы стало использоваться гораздо меньше. Однако два способа программирования не являются несовместимыми и их можно эффективно применять в комбинации.

СОЗДАНИЕ СЦЕНАРИЕВ НА ОСНОВЕ СОБЫТИЙ

Flash-плеер генерирует событие для того, чтобы сигнализировать о том, что с объектом в программе происходит нечто потенциально существенное. Например, пользователь мог щелкнуть на кнопке или изменить размеры рабочего поля; могла завершиться загрузка видеоклипа, XML-файла или звука; пользователь мог изменить содержимое текстового поля и т.п.

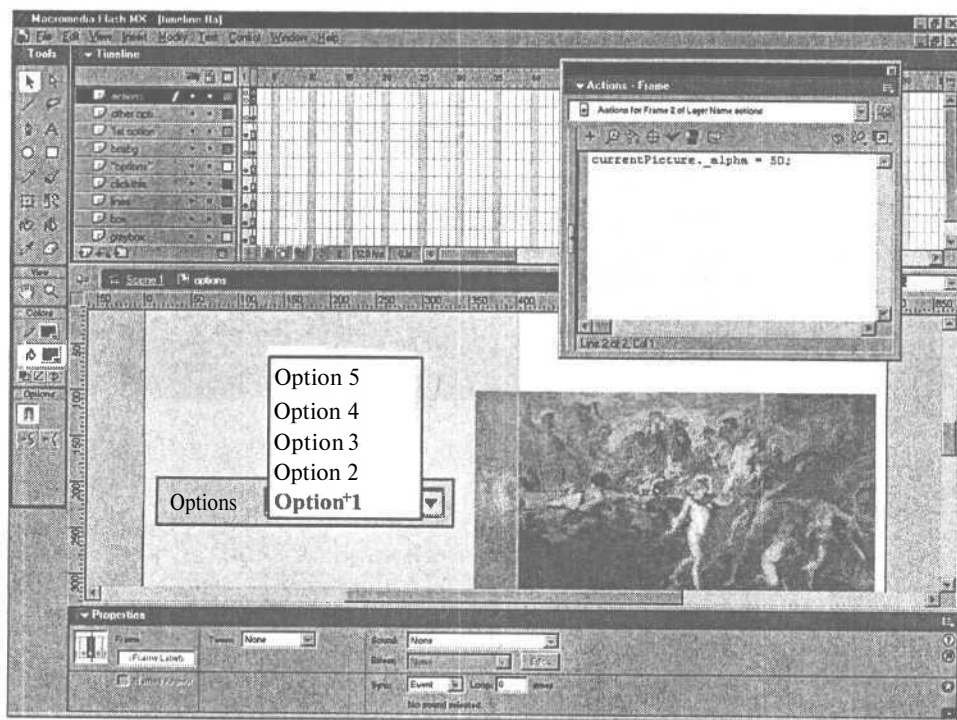


Рис. 16.2. Файл FLA раскрывающегося меню, показанного на рис. 16.1. Воспроизводящая головка находится в кадре 2

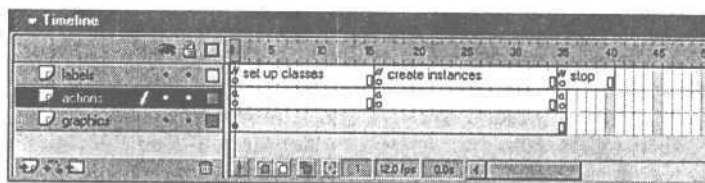


Рис. 16.3. Использование временной шкалы для компоновки кода, который логически состоит из двух или нескольких отдельных частей

Когда плеер генерирует событие, с фильмом ничего не происходит. Чтобы с ним что-то начало происходить, необходимо *захватить* событие путем создания обработчика событий или создать для него функцию *обратного вызова*. Тогда при наступлении события Flash-плеер вызовет эту функцию.

Например, функция обратного вызова может отображать новое изображение при нажатии кнопки, изменять расположение элементов на рабочем поле при изменении его размеров, проверять значение переменной после загрузки содержащего ее видеоклипа, проверять содержимое измененного текстового поля и предпринимать определенные действия на основе введенного текста.

Все функции обратного вызова являются методами или объектами. Если объект является видеоклипом, кнопкой или текстовым полем, функция обратного вызова будет выполняться только в том случае, если этот объект в данный момент находится в рабочем поле.

Имена функций обратного вызова являются предварительно заданными: они всегда начинаются с *on*, как, например, *onEnterFrame*. Обстоятельства, при которых они активизируются, также являются предварительно заданными. Однако решение о создании определен-

ного обработчика для определенного объекта и о том, какие предложения нужно поместить в функцию обратного вызова, целиком зависит от программиста.

Например, вы можете задать обработчик события `onEnterFrame` для видеоклипа `myClip`, поместив строку кода в кадр временной шкалы, содержащей `myClip`:

```
myClip.onEnterFrame = function () {trace("just testing");}
```

Flash-плеер будет активизировать функцию обратного вызова всякий раз, когда видеоклип достигнет нового кадра. В этом случае для каждого кадра в выходном окне будет отображаться фраза "just testing".

РАСШИРЕНИЕ ОБРАБОТКИ СОБЫТИЙ во FLASH MX

Во Flash 5 к кнопкам, видеоклипам и экземплярам классов `XML` и `XMLSocket` можно было присоединять только документированные события. Обработчики событий кнопок и видеоклипов должны были быть действиями объектов, а не кадров.

Совет

В главе 11 "Знакомство с ActionScript" рассматривались отличия между действиями объектов и действиями кадров. В этой же главе говорилось о том, как следует создавать обработчики событий видеоклипов с помощью действий объектов.

Во Flash 5 присоединять события к кнопке или видеоклипу необходимо было при разработке фильма. Этот метод определения событий все еще применяется. В рабочем поле необходимо щелкнуть на кнопке или видеоклипе, а затем воспользоваться панелью Actions (действия) для создания обработчика событий `on` (для кнопок) или `onClipEvent` (для видеоклипов). Во Flash 5 нельзя было присоединять обработчики событий кнопок и видеоклипов программным образом при выполнении фильма. Кроме того, при выполнении фильма их нельзя было изменить или удалить. Обработчики событий были *статическими*, или непреложными.

В противоположность этому, и во Flash 5, и во Flash MX события, связанные с экземплярами классов `XML` и `XMLSocket`, присоединяют программным образом при выполнении фильма. Например, для использования обработчика событий `onLoad` для объекта `XML` сначала программным образом создается объект, а затем к нему присоединяют обработчик события, тоже программным путем, как показано ниже.

```
myXML = new XML O; // создание объекта XML
myXML.onLoad = function(success) {trace("loaded !"); // присоединение
// обработчика
```

Совет

Подробная информация об XML представлена в главе 28 "XML-данные". Обработчики событий, созданные программным образом, при выполнении кода можно также программно изменить или удалить. Они являются динамическими.



Обработчики событий `on`, `onClipEvent` и обработчики событий, связанные с `XML`, во Flash MX и во Flash 5 работают одинаково. Кроме того, во Flash MX внесен ряд существенных расширений возможностей программирования на основе событий.

- Теперь создаваемые пользовательские объекты могут иметь обработчики событий и получать уведомления о событиях. Объект, явным образом зарегистрированный для получения событий определенного класса, называется *приемником*. Если вы хотите, чтобы встроенный объект получал события, которые он обычно получать не может, зарегистрируйте его как приемник.

- Обработчик можно создавать динамически для любого события, подлежащего захвату, независимо от того, связано ли оно с классом `Button`, `LoadVars`, `MovieClip`, `Sound`, `Stage`, `TextField`, `XML` или `XMLSocket`. Любое статическое событие `on` и `onClipEvent` имеет динамический эквивалент. Например, `onEnterFrame` является динамическим эквивалентом события `onClipEvent (enterFrame)`.

Совет

Класс `LoadVars` описан в главе 27 "Взаимосвязь с сервером".

- Теперь видеоклипы могут принимать все те же события и иметь все те же поведения, что и кнопки, отличаясь лишь областью действия. *Видеоклип кнопки* — это видеоклип, функционирующий так же, как и кнопка, но гораздо более гибкий и сохраняющий свою область действия, а не ограничивающий ее только той временной шкалой, на которой он находится. Некоторые пользователи Flash MX заявили, что они больше никогда не будут пользоваться "старыми" кнопками.
- Теперь можно задавать обработчики событий для четырех классов объектов, которые никогда раньше не получали событий. Это классы `LoadVars`, `Sound`, `TextField` и `Stage`. `LoadVars`, `TextField` и `Stage`, которые являются новыми типами объектов Flash MX.
- Вместе с новыми типами объектов появились и новые события: `onLoad` для объектов `LoadVars`, `resize` — для `Stage`, а также `onChanged`, `onScroller`, `onSetFocus` и `onKillFocus` — для объектов `TextField`.
- Теперь текстовые поля и кнопки имеют имена экземпляров. Это требуется для назначения им динамических обработчиков событий.

Совет

Подробная информация об объекте `TextField` представлена в главе 20 "Использование встроенных объектов фильмов".

- Для существующего объекта `Sound` (для которого во Flash 5 не было определено обработчиков событий) теперь имеются два обработчика событий: `onLoad` и `onSoundComplete`.
 - Новый класс `Button` делает кнопки "реальными" объектами и позволяет применять к ним все технологии объектно-ориентированного программирования. Класс `Button` имеет динамические эквиваленты всех статических событий `on`, которые имели кнопки. Например, класс `Button` имеет метод `onPress`, эквивалентный событию `on (press)`. Кроме того, класс `Button` имеет два новых события: `onSetFocus` и `onKillFocus`. Теперь кнопки также имеют имена экземпляров, что является требованием для присоединения динамических событий.
 - Класс `MovieClip` тоже поддерживает события `onSetFocus` и `onKillFocus`, а также динамические эквиваленты всех статических событий `onClipEvent`. Например, метод `onEnterFrame` является эквивалентом события `onClipEvent (enterFrame)`.
- В табл. 16.1 представлены все события, которые плеер Flash 6 считает потенциально важными.

СИСТЕМНЫЕ СОБЫТИЯ И СОБЫТИЯ ВВОДА ПОЛЬЗОВАТЕЛЕМ

Во Flash существуют два основных типа событий: *системные* и события *ввода пользователем*. Системные события генерируются фильмом. К ним относится, например, достижение видеоклипом определенного кадра или завершение процесса загрузки внешних данных.

ТАБЛИЦА 16.1. СОБЫТИЯ FLASH MX

	BUTTON	KEY	LOADVARS	MOVIE CLIP	MOUSE	SELEC- TION	TEXT FIELD	SOUND	STAGE	XML	XML SO- CKET
on	E										
onResize									Цп)		
onChanged							E(n), L(n)				
onScroller	E	x		E			E(n), L(n)				
onSetFocus	E			E		Цп)	E(n)				
onKillFocus	E			E			E(n)				
onRollOver	E			E							
onRollOut	E			E							
onPress	E			E							
onRelease	E			E							
onReleaseOutside	E			E							
onDragOut	E			E							
onDragOver	£			E							
onMouseUp				E	Цп)						
onMouseDown				E	Цп)						
onMouseMove				E	Цп)						
onKeyUp		L(n)		E							
onKeyDown		Цп)		E							
onClipEvent				E							
onUnload				E							
onEnterFrame				E							
onLoad			E(n)	E				E(n)		E	
onData				E						E	E
onConnect										E	E
onClose										E	E
onSoundComplete								E(n)			

E — событие; *L* — приемник; *(n)* — новый во Flash MX

События ввода пользователем начинаются после щелчка мышью или нажатия клавиши на клавиатуре. Точнее говоря, события наступают посредством интерфейса мыши или клавиатуры, к которому могут относиться и другие способы ввода данных, например, голосом. К событиям ввода пользователем относятся все события, связанные с классами Button, Key, Mouse, Stage и TextField. Многие события видеоклипов также являются событиями ввода пользователем, поскольку они дублируют функциональность событий классов Button, Key, Mouse, Stage и TextField.

РЕГИСТРАЦИЯ ПРИЕМНИКОВ

Каждая функция динамического обработчика событий является свойством объекта. Объект может принадлежать одному из следующих встроенных классов: `Button`, `LoadVars`, `MovieClip`, `Sound`, `TextField`, `Stage`, `XML` или `XMLSocket`. В этом случае объект по умолчанию получает одно или несколько событий. Пользователю достаточно просто создать функцию обратного вызова для определенного экземпляра объекта.

Если вы хотите, чтобы объект, принадлежащий определенному встроенному классу, получал события, которые он не получает по умолчанию, или если вы желаете, чтобы события получал созданный вами пользовательский объект, необходимо выполнить еще одно действие: зарегистрировать объект как приемник. События, которые может получать зарегистрированный объект, генерируют пять классов: `Key`, `Mouse`, `Selection`, `Stage` и `TextField`.

Объект регистрируется как приемник с помощью метода `addListener` того класса, в котором производится регистрация. Например, видеоклип обычно получает события `onKeyUp` и `onKeyDown` только тогда, когда он имеет клавиатурную фокусировку. Чтобы объект получал эти события при любых обстоятельствах, необходимо создать функцию обратного вызова и зарегистрировать видеоклип как приемник в классе `Key`, как показано ниже.

```
myClip.onKeyUp = function () {trace("onKeyUp fired");}
myClip.onKeyDown = function () {trace("onKeyDown fired");}
Mouse.addListener(myObj);
```

Приведенный ниже код создает объект `myObj`, вставляет в него обработчик события `onMouseUp` (отпускание кнопки мыши), а затем регистрирует объект как приемник класса `Mouse`.

```
myObj = new Object();
myObj.onMouseUp = function() {trace("mouseUp");};
Mouse.addListener(myObj);
```

Начиная с этого места при отпускании кнопки мыши будет запускаться событие `myObj.onMouseUp`.

Того же самого результата можно достичь, используя пустой видеоклип со статическим обработчиком события `onClipEvent (mouseup)`. Однако динамический обработчик имеет ряд преимуществ.

Во-первых, динамический подход более гибкий, поскольку позволяет программным образом при исполнении фильма решить, какие приемники следует регистрировать и для каких событий следует создавать обработчики. При статическом подходе принять решение о том, какие объекты получат те или иные обработчики событий, необходимо при разработке фильма.

Во-вторых, динамический подход позволяет поместить код в любое нужное место. Его не обязательно присоединять именно к видеоклипу. Весь код можно сосредоточить, например, в основной временной шкале, что облегчит его поиск и редактирование.

В-третьих, динамические обработчики событий являются реальными функциями, тогда как статические обработчики к ним не относятся. Это означает, что в отличие от статических, в динамических обработчиках событий можно определять локальные переменные, вызывать их явным образом и передавать в них параметры.

В-четвертых, существует только 9 статических событий видеоклипов, внедряемых в событие `onClipEvent` в качестве параметров. При этом динамических событий видеоклипов насчитывается 18.

В-пятых, для видеоклипов и кнопок не существует статических эквивалентов событий `onSetFocus` и `onKillFocus`. Таким образом, невозможно программным образом управлять способом связи статических обработчиков событий с клавиатурной фокусировкой.

В-шестых, динамические обработчики событий `onPress` и `onRelease` позволяют видеоклипу выступать в роли кнопки, запуская его события и отображая его пиктограмму только

тогда, когда видеоклип находится в активном состоянии. Чтобы достигнуть того же результата с помощью статических событий `mouseenter` и `mousedown`, придется затратить гораздо больше усилий.

И **наконец**, использование видеоклипов только для захвата событий подобно использованию дорогого ножа с десятью лезвиями только в качестве штопора. Пользовательский объект занимает гораздо меньше памяти, чем пустой видеоклип, поскольку он не обладает свойствами видеоклипа, не относящимися к захвату событий.

Использование статических событий клипов является обоснованным только для событий `onClipEvent (load)`, присоединяемых к видеоклипам, которые вручную помещены в рабочее поле при разработке фильма. В этом случае динамическое событие `onLoad` работать не будет.

Совет

Подробная информация о событиях `onLoad` и `onClipEvent (load)` приведена ниже, в подразделе "События, связанные с видеоклипами".

ОБЗОР ДАННЫХ И ИСПОЛЬЗОВАНИЕ КЛЮЧЕВОГО СЛОВА `this` С ОБРАБОТЧИКАМИ СОБЫТИЙ

Внутри статических обработчиков событий `on` и `onClipEvent` и ключевое слово `this`, и прямые ссылки относятся к временной шкале клипа, к которому присоединен обработчик событий. Например, приведенные ниже выражения эквивалентны.

```
onClipEvent (enterFrame) {
    _x++;
}
onClipEvent (enterFrame) {
    this._x++;
}
```

Однако при использовании динамических обработчиков событий ситуация усложняется. Каждый динамический обработчик событий является методом объекта, и внутри обработчика ключевое слово `this` относится к объекту. С другой стороны, областью действия прямых ссылок является временная шкала, в которой определена функция обработчика событий. Таким образом, если приведенный ниже код помещен в основную временную шкалу, то переменная `this.myVar` относится к объекту `meClip.myVar`, тогда как `myVar` относится к

```
_root.myVar;
_root.createEmptyMovieClip("myClip", 1);
myVar = "main timeline";
myClip.myVar = "myClip";
myClip.onMouseUp = function() {
    trace(this.myVar); // "myClip"
    trace(myVar); // "main timeline"
```

Как и в случае с другими методами, в динамических обработчиках событий можно определять локальные переменные. Например, в качестве первой строки кода функции `myClip.onMouseUp`, представленной в предыдущем примере, можно добавить следующее выражение:

```
var myVar = "local";
```

Тогда предложение `trace (myVar)` в последней строке кода будет отображать `"local"`, а не `"main timeline"`.

В `myClip.onMouseUp` можно указать и другой обработчик события объекта, и тогда значение переменной `this` внутри `myClip.onMouseUp` изменится на другой объект. С другой стороны, областью действия прямой ссылки на переменную `myVar` будет временная шкала, в которой определен функциональный литерал.

Если областью действия переменной или другого элемента данных является та область действия, в которой он (элемент данных) определен, то говорят, что он задан лексически.

Предположим, что вы создали еще один клип с именем `myClip2`, определили для него собственное свойство `myVar`, а также задали свойство `onMouseDown`, указывающее на обработчик события `myClip.onMouseUp`, определенный в предыдущем примере. Вот как будет выглядеть код:

```
myClip2.myVar = "myClip2"
meClip2.onMouseDown = _root.myClip.onMouseUp
```

Несмотря на то что `myClip.onMouseUp` и `myClip2.onMouseDown` относятся к одному и тому же функциональному литералу, `myClip2.onMouseDown` активизируется, когда пользователь *нажимает* кнопку мыши, а слово `this` означает `myClip2` внутри функции. Отображается "`myClip2`" и "`main timeline`". С другой стороны, `myClip.onMouseUp` активизируется, когда пользователь *отпускает* кнопку мыши, а слово `this` означает `myClip`, т.е. отображается "`myClip`" и "`main timeline`". В обоих случаях прямая ссылка все так же относится к основной временной шкале. Подводя итог, можно сказать, что значение переменной `this` является объектом, связанным с вызывающим методом, а прямая ссылка на переменную `myVar` задана лексически.

ЯВНОЕ ОБРАЩЕНИЕ К ОБРАБОТЧИКАМ СОБЫТИЙ

Хотя обработчики событий автоматически вызываются Flash-плеером при предварительно заданных условиях, тем не менее разрешается активизировать их и в явном виде. Такая необходимость может возникнуть, например, если обработчик события, созданный в одном объекте, потребуется выполнить в другом объекте. Методы `Function.apply()` и `Function.call()` позволяют указать **значение** ключевого слова `this` в пределах обращения.

Расширим код, приведенный в предыдущем подразделе. Так, в приведенном ниже коде создается объект с именем `myObj` (строка 1), задается его свойство `myVar` (строка 2), указывается обработчик события `onMouseDown`, который сначала выполняет функцию `myClip.onMouseUp`, а затем выполняет одну дополнительную строку кода (строки 3-6), наконец, (строка 7) объект `myObj` регистрируется как приемник объекта класса `MovieClip` так, что событие `myObj.onMouseDown` будет активизировано после щелчка мышью:

```
1: myObj = new Object();
2: myObj.myVar = "myObj";
3: myObj.onMouseDown = function() {
4:   myClip.onMouseUp.apply(myObj); // отображается "myObj", "main
   timeline"
5:   trace("this " + this.myVar); // отображается "myObj"
6: }
7: MovieClip.addListener(myObj);
```

Предложение `this.myVar` в пределах функции `myObj.onMouseDown` всегда отображает "`myObj`". Если вы хотите, чтобы при явном обращении к `myClip.onMouseUp` ключевое слово `this` относилось к объекту `myObj` внутри этой функции, необходимо применить метод `Function.apply()` или `Function.call()`. Если активизировать функцию `myClip.onMouseUp` с помощью обычного вызова, ключевое слово `this` всегда будет относиться к объекту `myClip`.

Расширяя код, представленный в предыдущем подразделе, представим себе, что приведенный ниже код был помещен в третий видеоклип.

```
// on myOtherClip
onClipEvent(load) {
    _root.myClip.onMouseDown();
}
```

В результате выполнения этого кода отображается "myClip" и "main timeline", поскольку обращение к нему производится как к методу объекта myClip. Поэтому он выполняется в контексте myClip.

Вызов обработчиков событий явным образом открывает третью возможность, отличную от обычного вызова и использования методов `apply()` или `call()`. Эта возможность заключается в указании переменной в методе и последующего использования переменной для выполнения метода. В этом случае `this` будет относиться к временной шкале, в которой определена переменная. Прямая ссылка на переменную myVar в myClip.onMouseDown все так же задается лексически. Это место является неизменным во всех трех способах вызова метода. Эта третья возможность проиллюстрирована в приведенном ниже коде, который относится к временной шкале клипа myClip2:

```
myVar = "my Var in myClip2";
f = _root.myClip.onMouseDown;
f(); // отображается "myVar in myClip2", "main timeline"
```

Совет

Подробная информация об областях действия функций приведена в главе 15 "Объединение предложений в функции".

ОБРАБОТЧИКИ СОБЫТИЙ И ФОКУСИРОВКА

И текстовые поля, и кнопки, и видеоклипы кнопок поддерживают события `onSetFocus` и `onKillFocus`. По умолчанию обработчики событий `onKeyUp` и `onKeyDown`, связанные с этими объектами, не будут активизированы до тех пор, пока объект не будет иметь клавиатурной фокусировки, т.е. пока курсор не будет находиться на объекте. (Статические эквиваленты, `onClipEvent(keyUp)` и `onClipEvent(keyDown)`, активизируются независимо от клавиатурной фокусировки.)

Кроме того, чтобы зарегистрировать объект для приема событий определенного класса, можно использовать предложение `addListener`. В этом случае динамические обработчики событий этого класса всегда будут активизироваться, независимо от того, имеет ли объект фокусировку.

Пусть имеется видеоклип myClip, и вы хотите, чтобы он захватывал события `onKeyUp` и `onKeyDown`. Сначала создадим функции обратного вызова:

```
myClip.onKeyUp = function() {trace("onKeyUp");};
myClip.onKeyDown = function() {trace("onKeyDown");};
```

У нас есть два варианта. Один из них заключается в том, чтобы сделать объект myClip видеоклипом кнопки. Функция обратного вызова активизируется, когда объект myClip имеет клавиатурную фокусировку.

Совет

Процесс создания видеоклипа кнопки будет описан далее в этой главе, в подразделе "События, связанные с видеоклипами".

Другая возможность заключается в том, чтобы сделать объект myClip приемником класса Key, как показано ниже.

```
Key.addListener(myClip);
```

Функции обратного вызова активизируются все время.

ОТКЛЮЧЕНИЕ И УДАЛЕНИЕ ОБРАБОТЧИКОВ СОБЫТИЙ

Удалить обработчик события легко, поскольку он является обычной переменной. Например:

```
delete myClip.onKeyDown;
```

Если вы зарегистрировали приемник для класса, в котором не собираетесь больше захватывать никаких событий, то удалить следует приемник, как показано ниже.

```
Key.removeListener(myClip);
```

Обработчик события можно отключить временно, чтобы он не задействовал циклы процессора в то время, когда не выполняет никакой полезной функции. Например, некоторое количество обработчиков событий `onEnterFrame`, выполняющих только проверку условия `if` и возврат, могут существенно замедлить программу.

Одна из возможностей заключается в том, чтобы установить сам обработчик события в значение `null`. Предположим, что обработчик события `onEnterFrame` определен следующим образом:

```
myClip.myFunc = function0 {  
    // предложения функции  
};  
myClip.onEnterFrame = myFunc;
```

Если вы хотите отключить обработчик события, можно использовать следующее предложение:

```
myClip.onEnterFrame = null;
```

Эта методика применена в фильме `ondata.fl`, имеющемся на прилагаемом компакт-диске. Найдите 25-ю строку программы:

```
else (_root.attachTarget.onEnterFrame = null;}
```

Чтобы снова запустить обработчик события, используйте следующее предложение:

```
myClip.onEnterFrame = myFunc;
```

На заметку

Чтобы включить нумерацию строк на панели Actions (Действия), щелкните в правом верхнем углу панели и из выпадающего меню выберите пункт View Line Numbers (Показывать номера строк). Включение нумерации строк облегчает поиск определенной строки кода.

Если установка обработчика события `onEnterFrame` в значение `null` дает нежелательные сторонние эффекты, то практически того же самого результата можно достичь, присвоив переменной значение `myFunc` и использовав переменную для выполнения функции `myFunc()` в обработчике события `onEnterFrame`. Для включения и отключения обработчика событий можно переключать переменную между значениями `null` и `myFunc`:

```
myClip.myFunc = function 0 {  
    // предложения функции  
};  
doNow = myClip.myFunc;  
myClip.onEnterFrame = function 0 {  
    doNow();  
};
```

Теперь, когда обработчик события необходимо отключить, используйте такое предложение:

```
doNow = null;
```

Когда обработчик события снова требуется запустить, используйте предложение

```
doNow = myClip.myFunc;
```

СОБЫТИЯ, СВЯЗАННЫЕ С КНОПКАМИ

Во Flash имеется восемь статических событий кнопок: `press`, `release`, `releaseOutside`, `rollOver`, `rollOut`, `dragOut`, `dragOver` и `keyPress`.



Имеется также девять динамических событий кнопок: `onPress`, `onRelease`, `onReleaseOutside`, `onRollOver`, `onRollOut`, `onDragOut`, `onDragOver`, `onKillFocus` и `onSetFocus`.

Обратите внимание на то, что кнопки не получают динамические события класса `Key` по умолчанию. Однако кнопки можно зарегистрировать как приемник класса `Key`, и тогда они смогут получать события `onKeyUp` и `onKeyDown` — динамические эквиваленты `keyPress`.

Все события кнопок активируются только в ответ на нажатие главной кнопки мыши. В ответ на нажатие вторичной кнопки мыши никакой реакции нет.

Все события кнопок, связанные с мышью (к ним относятся все события кнопок за исключением `keyPress`), активируются только один раз, в точке перехода. Например, если пользователь нажал и удерживает кнопку мыши, событие `press` активируется только единожды, когда кнопка была нажата в первый раз.

За исключением первого из следующих ниже подразделов, посвященного событиям, связанным с фокусировкой, во всех остальных описывается по одному событию, которое может быть захвачено как статическим, так и динамическим обработчиком. Таким образом, фраза "события `press` и `onPress`" сродни фразе "радио- и теленовости" — события одни и те же, но доступны по двум различным каналам. Можно также сказать: "события `press`", и это будет означать "событие, возникающее при нажатии кнопки и захватываемое либо статически, либо динамически".

ONKILLFOCUS И ONSETFOCUS

Событие `onSetFocus` возникает, когда кнопка получает клавиатурную фокусировку. Клавиатурная фокусировка изменяется, когда пользователь нажимает клавиши `<Tab>` или `<Shift+Tab>` для перемещения по текстовым полям и кнопкам страницы. Событие `onKillFocus` возникает, когда кнопка утрачивает клавиатурную фокусировку. События фокусировки были введены, в первую очередь, для текстовых полей ввода данных. Они представляют собой удобный прием при любой необходимой предварительной или последующей обработки данных, введенных пользователем в поля. Эти события также могут оказаться полезными для кнопок и видеоклипов кнопок. Например, если вы пользуетесь предназначенной для отправки данных кнопкой `Submit`, то когда кнопка становится активной, можно проверить правильность заполнения полей пользователем. Когда кнопка теряет фокусировку, вы можете сгенерировать сообщение типа "Спасибо".

PRESS И ONPRESS

События `press` и `onPress` возникают при нажатии пользователем главной кнопки мыши, в то время когда указатель мыши находится в пределах активной области кнопки. Данное событие обеспечивает максимально быструю реакцию, поскольку оно активируется сразу же при нажатии кнопки мыши. С другой стороны, это событие не позволяет пользователю изменить свое мнение после нажатия кнопки мыши. Это событие очень хорошо подходит для игр (где время реакции является решающим фактором) и для переключателей, где пользователь может отменить сделанный им выбор, повторно щелкнув на кнопке.

RELEASE И ONRELEASE

События `release` и `onRelease` возникают, когда пользователь нажал и отпустил главную кнопку мыши, в то время как указатель мыши находится в пределах активной области кнопки. Эти события позволяют пользователю изменить свое мнение относительно сделанного выбора, просто сместив указатель мыши с активной области до того, как отпустить кнопку мыши.

RELEASEOUTSIDE И ONRELEASEOUTSIDE

События `releaseOutside` и `onReleaseOutside` наступают при нажатии пользователем кнопки мыши, когда указатель находится в пределах активной области, последующего перемещения указателя за ее пределы и отпускания кнопки мыши. Если пользователю нужно, к примеру, перетащить какой-то элемент из одного места в другое, то в эти обработчики событий можно поместить соответствующие действия с тем, чтобы определить, находится ли указатель в нужном месте, когда пользователь отпускает кнопку мыши.

ROLLOVER И ONROLLOVER

События `rollover` и `onRollOver` активизируются, когда указатель мыши перемещается в пределы активной области при не нажатой кнопке мыши (находящейся в состоянии `Up`). Несмотря на то что состояние `Up` кнопки и соответствующий кадр `_up` видеоклипа кнопки описывают визуальный аспект **роллвера**, обработчик события позволяет также выполнить в этом месте программным образом какую-либо операцию. Например, можете воспользоваться этими событиями, если хотите сделать что-то непосредственно перед тем, как пользователь нажмет кнопку.

ROLLOUT И ONROLLOUT

События `rollout` и `onRollOut` активизируются, когда указатель мыши смещается за пределы активной области кнопки, а сама кнопка *не нажата*. Аналогично предыдущему случаю, состояние `Up` кнопки и соответствующий кадр `_up` видеоклипа кнопки описывают визуальный аспект процесса, а обработчики событий позволяют выполнить что-либо программным образом. Это событие можно использовать, к примеру, для того, чтобы выполнить какое-либо действие после того, как пользователь завершил работу с определенной кнопкой.

DRAGOUT И ONDRAGOUT

События `dragOut` и `onDragOut` активизируются, когда указатель мыши смещается за пределы активной области кнопки, а сама кнопка *нажата*.

DRAGOVER И ONDRAGOVER

События `dragOver` и `onDragOver` активизируются, когда пользователь выполняет действие типа `dragOut`, не отпускает кнопку мыши за пределами активной области кнопки, а затем перемещает указатель обратно в пределы активной области и отпускает кнопку мыши.

KEYPRESS

Событие `keyPress` возникает, когда пользователь нажимает какую-либо клавишу на клавиатуре. Событие имеет следующий формат:

```
on (keyPress клавиша) {  
    предложение1;  
    предложение2;  
}
```

где *клавиша* обозначает заключенную в кавычки строку, представляющую нажатую клавишу. Для буквенно-цифровых клавиш *клавиша* обозначает буквальное представление, например:

```
on (keyPress "a") {  
    trace ("a pressed");  
}
```

Кроме того, выражение *клавиша* может обозначать одно из 14 специальных ключевых слов, используемых для представления не буквенно-цифровых клавиш, таких как клавиши со стрелками, пробел и `<Enter>`:

<Backspace>	<Delete>	<Down>	<End>	<Enter>
<Home>	<Insert>	<Left>	<PgDn>	<PgUp>
<Right>	<Space><Tab>	<Up>		

Эти клавиши можно представить так:

```
on (keypress "<Space>") {
    trace("space bar pressed");
}
```

В отличие от событий, связанных с кнопками мыши, событие **keyPress** обычно повторяется многократно, если пользователь удерживает клавишу нажатой, хотя процесс происходит по-разному для разных клавиатур и операционных систем.

На заметку

Поскольку поведение клавиатурной фокусировки отличается для разных сред проектирования, Flash-плеера и браузеров, то любой фильм, в котором задействована клавиатурная фокусировка, необходимо протестировать во всех окружениях, в которых он может использоваться.

Обработчик события **keyPress** необходимо присоединить к экземпляру кнопки. Кроме того, чтобы обработчики события, связанные с клавишами клавиатуры, активизировались, Flash-плеер должен иметь фокусировку мыши. Фильмы автоматически получают фокусировку мыши во **Flash-плеере** или в проектировщике, но не в браузере. Следовательно, для достижения совместимости с браузером пользователь должен будет сначала щелкнуть на кнопке мыши, а затем вводить данные с клавиатуры.

На заметку

В среде Flash для задания клавиатурной фокусировки фильма может возникнуть необходимость щелкнуть в текстовом поле ввода данных.

Практически во всех ситуациях событие **keyPress** является более ограниченным и менее гибким по сравнению с событиями **onKeyUp** и **onKeyDown**, связанными с объектом **Key**. Например, событие **keyPress** не поддерживает функциональные клавиши, а также клавиши **<Caps Lock>**, **<Cmd>** (Macintosh) или **<Ctrl>** (Windows). Оно также не поддерживает приемников. И что наиболее важно, обработчик **keyPress** требует нажатия определенной клавиши. Он не позволяет получить событие при нажатии произвольной клавиши, а затем определить, какая это была клавиша. Все эти ограничения можно обойти с помощью объекта **Key**.

Событие **keyPress** обладает одной уникальной возможностью, которая не присуща объекту **Key**: оно может отключить клавишу **<Tab>** для предотвращения смещения фокусировки. Чтобы предотвратить использование пользователем клавиши **<Tab>** для смещения фокусировки в отдельном плеере, проектировщике или браузере, к кнопке необходимо присоединить следующий код:

```
on(keyPress "<Tab>") {
    // здесь должно быть какое-то предложение
    dummy = null;
}
```

Затем, при желании, можно воспользоваться объектом **Key** для выявления клавиши **<Tab>** и функцией **Selection.setFocus** — для изменения фокусировки явным образом.

С другой стороны, события объекта **Key** позволяют охватить клавишу **<Tab>** без ее отключения для смещения фокусировки.

СОБЫТИЯ, СВЯЗАННЫЕ С НАЖАТИЕМ КЛАВИШ

Во Flash 5 был введен объект **Key** (Клавиша) со связанными событиями видеоклипов **onClipEvent(keyDown)** и **onClipEvent(keyUp)**. Событие **keyDown** активизируется, когда пользователь нажимает произвольную клавишу на клавиатуре. Событие **keyUp** активизируется, когда пользователь отпускает клавишу.

С объектом `Key` связаны четыре метода, позволяющие определить, какая клавиша была нажата.

- `Key.getCode()` — возвращает код последней нажатой клавиши.
- `Key.getAscii()` — возвращает ASCII-код последней нажатой клавиши.
- `Key.isDown(код_клавиши)` — возвращает значение `true`, если указанная клавиша в данный момент нажата.
- `Key.isToggled(код_клавиши)` — возвращает значение `true`, если указанная клавиша (<Caps Lock> или <Num Lock>) в данный момент включена.

Метод `Key.isDown()` можно сам по себе использовать в обработчике события `enterFrame` для проверки клавиши в каждом кадре. Например, приведенный ниже код проверяет нажатие клавиши <Tab> в каждом кадре.

```
myClip.onEnterFrame = function () {  
    if (Key.isDown(Key.TAB))  
        trace("Tab key pressed");  
}
```

Такой код можно использовать, например, в играх, первоочередной задачей которых является максимально быстрое нажатие клавиш.

Обратите внимание на то, что константа `TAB` набрана прописными буквами.

На заметку

Константы, представляющие собой наиболее используемые клавиши, перечислены в списке инструментов в левой части панели **Actions (Действия)**, в группах **Objects, Movie, Key, Constants**.

За исключением процедуры проверки клавиши в каждом кадре, использование одного только метода `Key.isDown()` является ненадежным способом выявления нажатия клавиш, потому что пользователь может нажать клавишу в том кадре, в котором проверка не выполняется. Чтобы обойти эту проблему, воспользуйтесь обработчиком события `Key` для обнаружения нажатия клавиши и одним из методов `Key`, чтобы выяснить, какая клавиша была нажата. В приведенном ниже примере используется обработчик события `KeyDown` с методом `Key.getAscii()`, а также метод `String.fromCharCode()` для трансляции ASCII-кода в буквенно-цифровой символ.

```
myClip.onKeyDown = function() {  
    trace( String.fromCharCode(Key.getAscii()) );  
};
```



Во Flash MX не добавлялось новых событий или методов, но зато были введены динамические обработчики (`onKeyUp` и `onKeyDown`) и функции приемников, позволяющие получать события `Key` объектам, не являющимся видеоклипами.

Если Flash-плеер имеет фокусировку, то событие `on(keypress)` всегда будет активизироваться при нажатии соответствующей клавиши. Однако события, связанные с объектом `Key`, могут и не активизироваться. Это зависит от двух факторов: клавиатурной фокусировки и регистрации приемника.

На заметку

Чтобы обработчики события, связанные с нажатием клавиш (в том числе и `on(keyPress)`), активизировались, Flash-плеер должен иметь фокусировку мыши.

Совет

Фокусировка, приемники и события были рассмотрены ранее в этой главе, в подразделе "Обработчики событий и фокусировка".

- Существуют два способа активизации обработчиков событий, связанных с объектом Key.
- Любые обработчики событий, связанные с объектом Key, будут активизироваться независимо от клавиатурной фокусировки, если объект зарегистрирован как приемник объекта Key, как показано ниже.

```
myClip.onKeyDown = function () {trace("onKeyDown fired");};
Key.addListener (myClip); // работает для любого объекта,
                           // клавиатурная фокусировка игнорируется
```

- Если кнопка или видеоклип кнопки имеет клавиатурную фокусировку, то обработчик события, связанный с нажатием кнопки, будет активизироваться. При этом не обязательно, чтобы он был зарегистрирован как приемник объекта Key.

Два этих правила сведены в табл. 16.2.

ТАБЛИЦА 16.2. АКТИВИЗАЦИЯ СОБЫТИЙ, СВЯЗАННЫХ с ОБЪЕКТОМ KEY

	ОБЪЕКТ ЗАРЕГИСТРИРОВАН КАК ПРИЕМНИК KEY	ОБЪЕКТ НЕ ЗАРЕГИСТРИРОВАН КАК ПРИЕМНИК KEY
Объект имеет клавиатурную фокусировку	Для любого объекта будут активизироваться события, связанные с нажатием клавиш	События, связанные с нажатием клавиш, будут активизироваться для кнопок и видеоклипов кнопок
Объект не имеет клавиатурной фокусировки	Для любого объекта будут активизироваться события, связанные с нажатием клавиш	События, связанные с нажатием клавиш, активизируются не будут

СОБЫТИЯ, СВЯЗАННЫЕ С МЫШЬЮ

Во Flash 5 были введены объект Mouse (Мышь) и связанные с ним события видеоклипов `onClipEvent (mouseDown)`, `onClipEvent (mouseUp)` и `onClipEvent (mouseMove)`. Событие `mouseDown` активизируется, когда пользователь нажимает кнопку мыши. Событие `mouseUp` активизируется, когда пользователь отпускает кнопку мыши. Событие `mouseMove` активизируется в любой момент движения мыши. Оно может активизироваться и при загрузке фильма.

Во Flash MX добавлены динамические версии этих событий: `onMouseUp`, `onMouseDown` и `onMouseMove`. Кроме того, добавлена регистрация приемников так, чтобы эти события могли получать и объекты, не являющиеся видеоклипами. Несмотря на возможность поставить события мыши в соответствие кнопкам, они будут активизироваться независимо от того, находится ли указатель мыши в пределах рабочего поля, игнорируя клавиатурную фокусировку и активные области кнопок. Кроме того, в отличие от событий, связанных с кнопками, события, связанные с мышью, не изменяют вид указателя мыши.

Одним из предназначений событий `mouseMove` и `onMouseMove` является "пробуждение" программы после периода простоя. Если данные события используются в подобном этому ограниченном контексте, то пока они не используются, их можно установить равными `null`. В противном случае они могут генерировать большое количество событий за короткий промежуток времени, задействуя много циклов процессора.

Совет

Подробная информация об установке обработчиков событий в значение `null` представлена выше, в подразделе "Отключение и удаление обработчиков событий".

Одним из наиболее распространенных применений событий, связанных с мышью (в сочетании с двумя методами, `show ()` и `hide ()`, делающими указатель мыши видимым или невидимым), является создание пользовательского указателя мыши. Вы скрываете обычный ука-

затель, используете операторы `_root._xmouse` и `_root._ymouse` для проверки положения мыши при наступлении события, связанного с мышью, а затем отображаете в этом положении пользовательский указатель мыши.

СОБЫТИЯ, СВЯЗАННЫЕ С ВИДЕОКЛИПАМИ

Видеоклипы получают все события объектов классов `Button` (Кнопка), `Key` (Клавиша) и `Mouse` (Мышь). Кроме того, видеоклипы получают события, связанные с входом в кадр, загрузкой, выгрузкой и данными.



Нововведением Flash MX являются связанные с основной временной шкалой динамические формы этих событий (`onEnterFrame`, `onLoad`, `onUnload` и `onData`). Статические формы можно присоединять к видеоклипам, но не к основной временной шкале. События **видеоклипов**, присоединенные к основной временной шкале, не открывают никаких новых возможностей, но иногда способствуют упрощению операций. Например, они устраняют необходимость помещения кода в видеоклип только из-за того, что нужно выполнить какое-то действие в каждом кадре.

ENTERFRAME

Событие `enterFrame` уже много раз упоминалось в этой книге. Оно возникает каждый раз, когда воспроизводящая головка достигает кадра и является наиболее распространенным событием во Flash.

DATA И ONDATA

События `data` и `onData` возникают, когда внешние данные загружаются в видеоклип в результате выполнения функции `loadVariables()` или `loadMovie()`.

Совет

Класс `XML` также имеет событие `onData`, которое рассматривается в главе 28 "XML-данные".

Функция `loadVariables()` активизирует одно событие `data` по завершении загрузки всего пакета переменных. В отличие от нее, функция `loadMovie()` активизирует ряд событий `data` при загрузке фильма, максимально по одному на кадр. Таким образом, при использовании функции `loadVariables()` событие `data` говорит о том, что имеются все необходимые данные и с ними можно начинать работать. При использовании функции `loadMovie()` событие `data` сигнализирует о том, что поступила определенная часть загружаемого фильма, но при этом может быть необходимо проверить, достаточно ли этой порции для выполнения тех или иных задач. Для выполнения проверки можно воспользоваться функциями `getBytesLoaded()` и `getBytesTotal()`, свойствами видеоклипа `_framesloaded` и `_totalframes` или (для совместимости с Flash 4) исключенной функцией `ifFrameLoaded()`.

Совет

Подробная информация о свойстве `_framesloaded` и функции `ifFrameLoaded()` приведена в главе 14 "Работа с данными: использование предложений".

Простой пример использования события `onData` с функцией `loadVariables()` приведен в фильме `ondata fla`, содержащемся на прилагаемом к книге компакт-диске.

Совет

Подробная информация о событии `data` представлена в главе 27 "Взаимосвязь с сервером".

LOADиONLOAD

События `load` и `onLoad` активизируются при первоначальной загрузке видеоклипа в фильм.

Совет

Событие `onLoad` также связано с объектом `loadVars`. За подробной информацией обратитесь к главе 27 "Взаимосвязь с сервером".

Совет

Объект `Sound` также имеет событие `onLoad`, которое описывается ниже, в подразделе "События, связанные со звуком".

Клипы можно создавать вручную с помощью инструментальных средств разработки путем дублирования другого клипа с использованием функции `duplicateMovieClip()` или загрузки из библиотеки путем выполнения функции `attachMovie()`. Кроме того, их можно загрузить как внешний SWF-файл с помощью функции `loadMovie()`.

Flash MX: отсутствие события `Load` при выгрузке клипа



Во Flash 5 функция `unloadMovie()`, как ни парадоксально это кажется, также приводила к активизации события `load`, поскольку на место, образовавшееся при выгрузке фильма, загружался пустой видеоклип. Предположим, что в библиотеке имеется клип с идентификатором связи `myMC`, и этот код мы помещаем в основную временную шкалу:

```
attachMovie("myMC", "myMC", 1);
```

Предположим также, что имеется кнопка, к которой присоединен следующий код:

```
on(press) {  
    _root.myMC.unloadMovie();  
}
```

При нажатии кнопки отображается сообщение об ошибке `Error opening URL`... , поскольку Flash пытается загрузить безымянный видеоклип.

Теперь все это не так. Во Flash MX кнопка будет работать без сообщений об ошибке. Очевидно, что Flash MX не генерирует событие `load` при выгрузке клипа.

В большинстве случаев динамические и статические обработчики событий функционируют практически идентично. Динамические обработчики обеспечивают большую гибкость, облегчают централизацию кода и локализуют область действия.

Совет

Более подробная информация о преимуществах динамических обработчиков событий над статическими приведена выше, в подразделе "Регистрация приемников".

Все, что можно сделать с динамическими обработчиками событий, обычно можно применить и к статическим обработчикам, и наоборот. Но это не касается событий `load` и `onLoad`. Фактически первая же попытка применить событие `onLoad` оказывается бесполезной. Почему?

Проблема заключается в том, что обработчик события `onLoad` должен быть создан до того, как загрузится клип. Однако, если клип не загрузился, то нечего ставить в соответствие обработчику `onLoad`. Таким образом, если на рабочем поле вручную создан клип `myClip`, а в

первый кадр основной временной шкалы помещен приведенный ниже код, то ко времени получения кода интерпретирующей программой клип уже будет загружен:

```
MyClip.onLoad = function () {trace("myClip loads");}; // никогда не активизируется
```

С другой стороны, если вы собираетесь загрузить клип программным образом с помощью функции `attachMovieClip()`, то возникает вопрос: когда нужно объявлять обработчик события `onLoad`: до или после присоединения? Если до, то клип еще не существует и для него нельзя определить обработчик события. Если после, то клип уже загружен и объявлять обработчик события уже слишком поздно.

Получается, что существует только один способ успешного применения динамического события `onLoad`: событие должно быть унаследовано из объекта `prototype` функции-конструктора, т.е. из класса. Когда интерпретирующая программа создает новый объект, то есть промежуток времени, когда объект еще не имеет имени и "не рожден", но уже обладает всеми коллективными методами и свойствами объекта `prototype`.

Совет

Процесс создания нового объекта более полно описан в главе 15 "Объединение предложений в функции".

Если одним из коллективных методов, которыми обладает "рожденный" объект, является обработчик события `onLoad` (он будет активизироваться).

Это продемонстрировано в файле `onload fla`, часть кода которого приведена ниже. Основной идеей программы является помещение обработчика события `onLoad` в объект `prototype` класса `MyMCClass` и последующее использование предложения `Object.registerClass` для регистрации видеоклипа в библиотеке класса `MyMCClass`.

Совет

Подробная информация о предложении `Object.registerClass` приведена в главе 17 "Демонстрация мощи видеоклипов".

Когда видеоклип загружается из библиотеки с помощью предложения `attachMovie`, он ставится в соответствие классу `MyMCClass` и обработчик события `onLoad` активизируется.

```
myMCclass = function O {} // определение класса
myMCclass.prototype = new MovieClip(); // наследование из MovieClip
myMCclass.prototype.onLoad = function() { // помещение onLoad в прототип
    trace(this+" loaded");
}
Object.registerClass("mc", myMCclass); // регистрация видеоклипа
_root.attachMovie("mc", "mc", 2);
```

В представленном на компакт-диске файле `onload2 fla`, созданном Эрикой Нортон (Erica Norton), ведущим специалистом по гарантии качества компании Macromedia, продемонстрирована та же самая стратегия — помещение кода между командами компилятора `ttinitclip` и `#endinitclip`. Эти команды превращают предложения в действия по *инициализации клипа*, которые выполняются перед помещением любого кода в кадр 1.

Совет

Инициализация клипа часто используется вместе с компонентами. Об этом говорится в главе 22 "Компоненты".

UNLOAD И ONUNLOAD

События `unload` и `onUnLoad` активизируются при использовании функций `unloadMovie()` или `unloadMovieNum O` для выгрузки загруженного клипа.

Событие unload является хорошим средством освобождения ресурсов (таких как приемники), которые были связаны с видеоклипом. Например:

```
onClipEvent(load) {  
    Mouse.addListener(this);  
}  
onClipEvent(unload) {  
    Mouse.removeListener(this);  
}
```

Совет

Не можете заставить работать динамическое событие видеоклипа? Обратитесь к разделу "Возможные проблемы" в конце этой главы.

СОЗДАНИЕ ВИДЕОКЛИПА КНОПКИ



Во Flash 5 видеоклипы не могли получать события кнопок. Во Flash MX они по умолчанию получают эти события. Этот факт является основой создания видеоклипа кнопки. Существуют еще три функциональные возможности, присущие кнопкам.

- *Курсор в виде руки*, позволяющий пользователю получить визуальное представление о том, когда можно щелкать на кнопке. По умолчанию, когда обработчик события кнопки ставится в соответствие видеоклипу, клип при наведении на него указателя мыши начинает отображать курсор в виде руки.
- *Активная область* — область рабочего поля, в пределах которой можно щелкать на кнопке. Теперь видеоклипы обладают свойством `hitArea`, позволяющим назначить любой видеоклип как активную область видеоклипа кнопки.

Совет

Подробная информация о курсоре, принимающем вид руки, и свойстве `hitArea` видеоклипа приведена в главе 17 "Демонстрация мощи видеоклипов".

- Состояния Up, Over и Down, позволяющие кнопке изменять свой внешний вид, когда пользователь перемещает курсор мыши в активную область кнопки или когда он нажимает ее. Во временной шкале видеоклипа кнопки можно пометить кадры, отвечающие разным состояниям кнопки (`_up`, `_over` и `_down`), и они автоматически будут обеспечивать нужные поведения. В качестве альтернативного способа внедрения поведений можно воспользоваться обработчиками событий `onRollOver`, `onRollOut` и `onRelease` или `onPress`.

В файле `buttonMovie.fla`, содержащемся на компакт-диске, продемонстрирован клип, использующий свойство `hitArea` и кадры с метками `_up`, `_over` и `_down`.

В приведенном на компакт-диске файле `dynamicButtonMovie.fla`, разработанном Web-дизайнером и владельцем узла *i-Technica* (<http://i-Technica.com>) Хелен Триоло (Helen Triolo), продемонстрирован альтернативный подход, заключающийся в использовании заданной по умолчанию активной области и обработчиков событий `onRollOver`, `onRollOut` и `onRelease`. В этой программе видеоклип кнопки создается полностью программным образом.

Сначала с помощью прикладного программного интерфейса рисования и функции `createEmptyMovie()` был создан видеоклип с именем `triangle`. Функциональность кнопки `triangle` в файле `dynamicButtonMovie.fla` задается следующими строками кода:

```
triangle.c = new Color(triangle);  
triangle.onRollOver = function0 {
```



```

        this.c.setRGB(0xff0000); // красный цвет при наведении мыши
    }
    triangle.onRollOut = function() {
        this.c.setRGB(0x0000ff); // первоначальный синий цвет
    }
    triangle.onRelease = function() {
        trace("button clicked");
    }
}

```

В первой строке для клипа `triangle` задается свойство `triangle.c`, принадлежащее классу `Color` (цвет) и используемое в последующих обработчиках событий `onRollOver`, `onRollOut` и `onRelease` для изменения цвета видеоклипа в состояниях `Up`, `Down` и `Over`. Обработчик события `onRelease` делает кнопку доступной для нажатия.

СОБЫТИЯ, СВЯЗАННЫЕ С ВЫБОРОМ

Объект `Selection` предназначен в первую очередь для того, чтобы помочь в управлении клавиатурной фокусировкой.

Если с использованием предложения `addListener` зарегистрировать объект в классе `Selection` и определить обработчик события `onSetFocus`, то этот объект будет получать уведомления обо всех изменениях фокусировки.

Обработчик события `Selection.onSetFocus` имеет следующий формат:

```

onSetFocus (старый_фокус, новый_фокус) {
    предложения
}

```

Приведенный ниже код демонстрирует создание объекта и делает его приемником всех событий `onSetFocus`.

```

myObj = new Object();
myObj.onSetFocus = function() {trace("focus set event occurred");};
Selection.addListener(myObj);

```

Этот обработчик отличается от обработчиков событий `onSetFocus`, заданных по умолчанию (без использования предложения `addListener`) для экземпляров классов `Button`, `MovieClip` и `TextField`. Обработчик события `onSetFocus`, заданный по умолчанию, активизируется только в случае, если получает фокусировку *экземпляра, которому он принадлежит*. Следовательно, заданному по умолчанию обработчику событий `onSetFocus` необходим только один аргумент, каковым является старый фокус объекта. Новым фокусом всегда является экземпляр, которому принадлежит данный обработчик. Таким образом, заданный по умолчанию обработчик события `onSetFocus` имеет такой формат:

```

onSetFocus (старый_фокус) {
    предложения
}

```

Если экземпляр класса `Button`, `MovieClip` или `TextField` с помощью предложения `addListener` зарегистрирован в классе `Selection`, то его обработчик события `onSetFocus` будет получать уведомления, присущие обоим классам. Такой "двойственный" обработчик события `onSetFocus` будет активизироваться *дважды*, когда фокусировку получает экземпляр, и *единожды*, когда фокусировку получает любой другой объект.

Эти аспекты продемонстрированы в файле `selection fla`.

Объект `Selection` и фокусировка рассматривались ранее в этой главе, в подразделах "Обработчики событий и фокусировка" и "События, связанные с кнопками".

СОБЫТИЯ, СВЯЗАННЫЕ СО ЗВУКОМ



Объект `Sound` (звук) имеет два события. Событие `onLoad` активизируется при загрузке звука, а событие `onSoundComplete` — по завершении воспроизведения звука.

В приведенном ниже коде, взятом из файла `sound.fla`, продемонстрировано создание объекта `Sound`, определение обработчиков событий и загрузка звука.

На заметку

Очень важно определить обработчики событий перед загрузкой звука.

```
music = new Sound(this); // создание объекта sound
music.onLoad = function () {
    this.start(); // начало воспроизведения звука
                // когда звук загружается
};
music.onSoundComplete = function () {
    trace("music finished playing");
};
music.loadSound ("electro.mp3", true); // "true" обозначает потоковый
                                     // режим
```

Подробная информация об объекте `Sound` приведена в главе 20 "Использование встроенных объектов фильмов".

СОБЫТИЕ ONRESIZE, СВЯЗАННОЕ С РАБОЧИМ ПОЛЕМ



Новой возможностью Flash MX является способность манипулирования рабочим полем как объектом. Объект `stage` (Рабочее поле) имеет только одно событие, `onResize`, которое происходит, когда пользователь изменяет размеры рабочего поля. Для изменения размеров рабочего поля во Flash-плейере или в браузере пользователь может щелкнуть на маркерах, расположенных в его левом верхнем (Macintosh) или правом верхнем (Windows) углу. Пользователь может развернуть рабочее поле или свернуть его. Эти действия приводят к активизации события `onResize`. Еще одним способом изменения размеров является перетаскивание краев окна плейера или браузера. При этом будет активизировано несколько событий `onResize`, поскольку размеры окна изменяются непрерывно. Во Flash-плейере или в инструментальной среде разработки выполнение команды `View⇒Magnification⇒100%` также приводит к активизации события `onResize`, тогда как при минимизации или масштабировании размеров рабочего поля оно не активизируется. Кроме того, при первоначальной загрузке фильма, будь то в браузере, во Flash-плейере или в среде разработки, может активизироваться одно или несколько событий `Stage.resize`.

Свойство `scaleMode` объекта `Stage` определяет, каким образом масштабируется и обрезается графика при изменении размеров рабочего поля. В браузере на масштабирование и обрезку также влияют параметры HTML-файла.

Подробная информация о свойстве `scaleMode` приведена в главе 20 "Использование встроенных объектов фильмов".

Существует очень много комбинаций свойства `scaleMode` и параметров HTML. Рассмотрим вариант, который в целом дает неплохие результаты.

Выполните команду **File⇒Publish Settings** (Файл⇒Параметры публикации). В открывшемся диалоговом окне **Publish Settings** перейдите ко вкладке **HTML**, из раскрывающегося списка **Dimensions** (Размеры) выберите пункт **Percent** (Проценты), а в полях **Width** (Ширина) и **Height** (Высота) оставьте заданные по умолчанию размеры, равные 100%. В этой же вкладке из раскрывающегося списка **Scale** (Масштаб) выберите пункт **Default (Show All)** (По умолчанию (Показывать все)).

Задайте свойство `Stage.scaleMode` равным `"noScale"`, а свойство `Stage.align` — равным `"TL"`. В приведенном ниже коде показано, как эти свойства задаются программным образом.

Любой объект может быть приемником события `Stage.resize`. В приведенном ниже простом примере объект `Stage` зарегистрирован как приемник в своем же классе, что устраняет необходимость создания отдельного объекта для приема данного события.

```
Stage.scaleMode = "noScale";
Stage.align = "TL";
Stage.onResize = function() {
    // здесь приводится код!
};
Stage.addListener(Stage);
```

Событие `Stage.resize` используется, как правило, для корректировки размеров, компоновки или содержимого фильма в соответствии с новыми размерами рабочего поля.

СОБЫТИЯ, СВЯЗАННЫЕ С ТЕКСТОВЫМИ ПОЛЯМИ

Во Flash используются четыре события `TextField`.

- Событие `onSetFocus` происходит, когда текстовое поле не является активным, а пользователь переносит на него фокусировку либо щелкая в нем, либо нажимая клавиши `<Tab>` или `<Shift+Tab>`. Это событие имеет один аргумент — предыдущий фокус.
- Событие `onKillFocus` происходит, когда текстовое поле является активным, а пользователь переносит фокусировку с него на другой элемент, либо щелкая за пределами текстового поля, либо нажимая клавиши `<Tab>` или `<Shift+Tab>`. Это событие имеет один аргумент — новый фокус.

Совет

Подробная информация о событиях `onSetFocus` и `onKillFocus` приведена выше, в разделе "События, связанные с выбором".

Совет

Нажимаете клавишу `<Tab>` для изменения фокусировки, но ничего не происходит? Обратитесь к разделу "Возможные проблемы" в конце этой главы.

- Событие `onChanged` активизируется всякий раз при изменении текста в поле (например, когда пользователь вводит или удаляет букву). Вырезание или вставка блока текста приводит к активизации только одного события — `onChanged`. Это событие имеет только один аргумент — имя экземпляра текстового поля.
- Событие `onScroller` активизируется при перемещении полосы прокрутки в текстовом поле. Это может происходить либо после щелчка на самой полосе прокрутки, либо автоматически, при вводе текста пользователем. Данное событие имеет только один аргумент — имя экземпляра текстового поля.

Каждое из указанных событий продемонстрировано в приведенном на компакт-диске фильме `textfieldEvents.flx`. Этот файл содержит одно текстовое поле `myText` с компонентом `ScrollBar`, добавленным из окна `Components`. (Чтобы открыть это окно, выполните команду `Window⇨Components`.) События `onChanged` и `onScroller` дают приращение переменных (`myScrollBarChanged` и `myScrollBarScroller`), которые отображаются в динамических текстовых полях. Ниже приведен код.

```
myScrollBarChanged = 0;
myScrollBarScrolled = 0;
myText.onSetFocus = function(oldFocus) {
    trace(oldFocus);
    trace("setfocus");
};
myText.onKillFocus = function(newFocus) {
    trace(newFocus);
    trace("kill focus");
};
myText.onChanged = function(textfieldName) {
    trace(textfieldName);
    myScrollBarChanged++;
};
myText.onScroller = function(textfieldName) {
    trace(textfieldName);
    myScrollBarScroller++;
};
```

УСТАНОВЛЕНИЕ ПОСЛЕДОВАТЕЛЬНОСТИ ДЕЙСТВИЙ С ПОМОЩЬЮ ПОРЯДКА ВЫПОЛНЕНИЯ

Системные события выполняются в определенном порядке относительно основной временной шкалы и временной шкалы любого дочернего видеоклипа. Необходимо принять во внимание три основных фактора: во-первых, относительное расположение слоев, в которых происходят события; во-вторых, положение клипа, содержащего событие, в иерархической структуре (основной фильм, дочерний клип); и, в-третьих, тип события (например, загрузка данных с использованием функций `attachMovie()`, `loadMovie()` или `duplicateMovie()`, инициализация клипа путем выполнения команд `#initclip` и `#endinitclip`, загрузка и выгрузка клипа с помощью обработчиков событий `onClipEvent(load)` и `onClipEvent(unload)`, события `onLoad` и `onUnload`, достижение кадра с использованием обработчиков событий `onClipEvent(enterFrame)` и `onEnterFrame`).

Ниже приведено несколько основных правил управления последовательностью выполнения системных событий.

- При загрузке фильма любой код инициализации клипа выполняется до любого другого кода, содержащегося в этом же кадре.
- Статические события загрузки (`onClipEvent(load)`) выполняются в том кадре, в который загружается клип, и до выполнения любого кода временной шкалы этого клипа.
- Динамические события загрузки (`onLoad`) выполняются в кадре, следующем *после* кадра, в который загружается клип, и *после* выполнения любого кода временной шкалы этого клипа.
- Загрузка и выгрузка фильма предшествует событию, происходящему при достижении фильмом определенного кадра. Таким образом, события `enterFrame` и

`onEnterFrame` не начнутся до тех пор, пока не будет достигнут кадр, следующий после кадра, содержащего фильм.

- В пределах одной временной шкалы код выполняется от верхнего слоя к нижнему.
- Код родительской временной шкалы, не относящийся к событию, выполняется до не относящегося к событию кода, расположенного в дочерней временной шкале.
- Если в один и тот же кадр загружается несколько клипов, то первоначально их код выполняется в соответствии с порядком загрузки. Для задания порядка загрузки выполните команду **File⇒Publish Settings**, перейдите ко вкладке **Flash** и из раскрывающегося меню **Load Order** выберите пункт **Bottom Up** (Снизу **вверх**), который задан по умолчанию, или **Top Down** (Сверху вниз). В последующих кадрах порядок выполнения кода будет противоположным.

При взаимодействии всех этих правил порядок выполнения кода может стать очень сложным. Это еще одна причина, по которой нужно постараться как можно сильнее централизовать код и ограничить число дочерних клипов, содержащих код.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему не активизируется мой обработчик события?

Одна из причин может заключаться в том, что вместо имени динамического обработчика события используется имя статического параметра. Например, `mouseDown` является статическим параметром, который нужно использовать со статическим обработчиком события, как показано ниже.

```
onClipEvent(mouseDown) {  
    // предложения  
}
```

Если во Flash 5 вы часто использовали статические обработчики событий, то можете написать нечто подобное и во Flash MX:

```
myClip.mouseDown = functionO { // ЭТО НЕ ОБРАБОТЧИК СОБЫТИЯ!  
    // предложения  
}
```

Нужно использовать не `mouseDown`, а `onMouseDown`:

```
myClip.onMouseDown = function() {  
    // предложения  
}
```

Я нажимаю клавишу <Tab>, чтобы изменить фокусировку, но ничего не происходит.

Это стандартная ситуация в среде проектирования. Попробуйте протестировать SWF-файл в браузере и во Flash-плеере.

FLASH ЗА РАБОТОЙ: УНИВЕРСАЛЬНЫЕ ПРОЦЕССОРЫ СОБЫТИЙ FLASH

Во Flash MX документированная модель событий позволяет принимать лишь ограниченный набор последних: нажатие клавиш, щелчки мышью, изменения текстовых полей и размеров рабочего поля. Однако могут возникать ситуации, требующие приема других событий,

связанных либо со встроенными, либо с созданными объектами. Это можно сделать с помощью универсального процессора событий.

Предположим, что у нас есть фильм, в котором есть кукла и "оживляющий" ее кукольник. Пользователь управляет действиями кукольника. Действия кукольника, в свою очередь, вызывают отклики самой куклы. Вы хотите определить события, связанные с кукольником, и хотите, чтобы кукла принимала эти события. Эта задача состоит из четырех основных этапов.

- Создайте пользовательские объекты, представляющие куклу и кукольника (они также могут быть представлены видеоклипами или другими встроенными объектами).
- Определите способ управления пользователем кукольником.
- Задайте для объекта, представляющего кукольника, необходимые свойства и методы отслеживания, добавления и удаления приемников, а также для отправления уведомлений об одном или нескольких событиях.
- Для куклы задайте соответствующие функции обратного вызова (совпадающие с теми, что были заданы для объекта, представляющего кукольника) и сделайте куклу приемником кукольника.

Выполнение третьего шага требует наличия универсального процессора событий. Некоторые из них можно бесплатно загрузить из Web. Ссылки на узлы загрузки имеются на Web-узле <http://www.flashoop.com>.

Кроме того, во Flash MX встроен недокументированный универсальный процессор событий ASBROADCASTER. В приведенной ниже программе `puppets.fla` процессор событий ASBROADCASTER используется для того, чтобы на действия кукольника откликались две куклы. Основная идея программы заключается в том, что объект-кукольник принимает событие `onKeyDown` и активизирует событие `assumeDefaultPosition`, когда пользователь нажимает клавишу <Пробел>. Две куклы, зарегистрированные как приемники объекта-кукольника и для которых определены функции обратного вызова `assumeDefaultPosition()`, принимают событие `assumeDefaultPosition`. При выполнении функции обратного вызова в выходном окне отображаются следующие строки:

```
puppet1 assumes default position (кукла 1 предполагает положение по умолчанию);  
puppet2 assumes default position (кукла 2 предполагает положение по умолчанию).
```

Ниже описано, как соотносится с описанными выше четырем основным этапам каждой строка программы.

Я Объект-кукольник создается в строке 1, а объекты-куклы — в строках 12 и 13.

- В строках 2-7 создается функция обратного вызова (`onKeyDown()`) для объекта-кукольника, чтобы кукольник активизировал событие `assumeDefaultPosition` (строка 5) при нажатии пользователем клавиши <Пробел>.
- Предложение `initialize` в строке 8 наделяет объект `puppetMaster` (Кукольник) четырьмя свойствами: массивом `_listeners`, методами `addListener()`, `removeListener` и `broadcastMessage()`. Методы `addListener()` и `removeListener` добавляют и удаляют объекты из массива `_listeners`. Метод `broadcastMessage()` выполняет функцию обратного вызова во всех объектах массива `_listeners`.
- Функция `defaultPosition()`, определенная в строках 9–11, используется объектами-куклами как функция обратного вызова. Строки 14 и 15 "заставляют" двух кукол принимать события кукольника. Строки 16 и 17 задают для каждой из кукол метод `assumeDefaultPosition()`.

```
1: puppetMaster = {};  
2: Key.addListener(puppetMaster);
```

```

3: puppetMaster.onKeyDown = function(){
4:   if (Key.isDown(Key.SPACE)){
5:     this.broadcastMessage("assumeDefaultPosition");
6:   }
7: }
8: ASBroadcaster.initialize(puppetMaster);
9: function defaultPosition (name){
10:  trace(this.name+" assumes default position");
11: }
12: puppet1 = {name: "puppet1"};
13: puppet2 = {name: "puppet2"};
14: puppetMaster.addListener(puppet1);
15: puppetMaster.addListener(puppet2);
16: puppet1.assumeDefaultPosition = defaultPosition;
17: puppet2.assumeDefaultPosition = defaultPosition;

```

Первым аргументом метода `ASBroadcaster.broadcastMessage()` является имя события. Кроме того, можно добавить любое количество других аргументов. В приведенной ниже программе `puppets2.fla`, содержащейся на прилагаемом к книге компакт-диске, объект-кукольник откликается на любую из трех клавиш: <Пробел>, <K> и <P> (в английской раскладке клавиатуры). В зависимости от того, какую клавишу нажимает пользователь, куклольник выполняет метод `ASBroadcaster.broadcastMessage()` с различным вторым параметром или (при нажатии клавиши <Пробел>) без второго параметра. Второй параметр (или его отсутствие) говорит куклам о том, какую позицию следует занять.

```

1: puppetMaster = {};
2: Key.addListener(puppetMaster);
3: puppetMaster.onKeyDown = function(){
4:   if (Key.isDown(Key.SPACE)){
5:     this.broadcastMessage("assumePosition");
6:   }
7:   if (Key.isDown(75)){ // 75 это код клавиши <K>
8:     this.broadcastMessage("assumePosition", "kneeling");
9:   }
10:  if (Key.isDown(80)){ // 80 это код клавиши <P>
11:    this.broadcastMessage("assumePosition", "prone");
12:  }
13: }
14: ASBroadcaster.initialize(puppetMaster);
15: function assumePosition (arg){
16:  if (arg == undefined) arg = "default";
17:  trace(this.name+" assumes "+arg+" position");
18: }
19: puppet1 = {name: "puppet1"};
20: puppet2 = {name: "puppet2"};
21: puppetMaster.addListener(puppet1);
22: puppetMaster.addListener(puppet2);
23: puppet1.assumePosition = assumePosition;
24: puppet2.assumePosition = assumePosition;

```

Я узнал о существовании `ASBroadcaster` из списка рассылки для программистов Flash. Проверьте архивы или подпишитесь на рассылку на узле <http://chattyfig.figleaf.com>.

Внимание! Поскольку `ASBroadcaster` является недокументированной функцией, то в следующих версиях Flash она может измениться или вообще не поддерживаться. В ActionScript вставить новый код, заменяющий эту функцию, достаточно легко. Подробная информация об `ASBroadcaster` приведена на узле <http://chattyfig.figleaf.com/flashcoders-wiki> (зайдя на узел, щелкните на ссылках *Undocumented Features* и *Flash MX Features*).

ДЕМОНСТРАЦИЯ МОЩИ ВИДЕОКЛИПОВ

В ЭТОЙ ГЛАВЕ...

Представление видеоклипов	357
Новые свойства видеоклипов во Flash MX	363
Создание и удаление видеоклипов	367
Загрузка и выгрузка внешнего содержимого	369
Управление визуальным порядком наложения видеоклипов	372
Использование исходных объектов для задания свойств новых видеоклипов	377
Использование метода <code>Object.registerClass()</code> для выделения нового видеоклипа в подкласс	378
Проверка перекрытия видеоклипов	382
Повторный вызов функции с помощью глобальной функции <code>setInterval()</code>	385
Перетаскивание видеоклипов	387
Динамические маски	389
Возможные проблемы	391
Flash за работой: игра "счастливый случай"	392

ПРЕДСТАВЛЕНИЕ ВИДЕОКЛИПОВ

Видеоклип, безусловно, — самый важный и распространенный тип объектов Macromedia Flash. Видеоклипы являются уникальными объектами. Несмотря на создание класса `MovieClip`, призванного привести видеоклипы к рамкам объектно-ориентированного программирования, введенного в версии Flash 5, они все же не похожи на другие типы объектов и требуют особого обращения.

Например, в отличие от большинства других классов, для создания экземпляров класса `MovieClip` нельзя использовать оператор `new`. Приведенное ниже предложение не создает новый видеоклип.

```
myClip = new MovieClip(); // НЕ создает новый видеоклип!
```

Тем не менее предложение `new MovieClip()` может оказаться полезным. Например, его можно использовать для создания подкласса, наследующего класс `MovieClip`. Но, в отличие от других функций-конструкторов, сам по себе класс `MovieClip` не может служить полноценным средством создания видеоклипов.

Совет

Подробная информация о создании подкласса, наследующего класс `MovieClip`, приведена ниже, в разделе "Использование метода `Object.registerClass()` для выделения нового видеоклипа в подкласс".



Фактически во Flash 5 невозможно было создать совершенно новый видеоклип программным образом. Во Flash MX эту функцию выполняет метод `createEmptyMovieClip()` класса `MovieClip`. Этот метод необходим в связи с тем, что хотя посредством класса `MovieClip` видеоклип попадает в рамки объектно-ориентированного программирования, введенного во Flash 5, его основы были заложены еще до появления ООП, и многие базовые свойства (такие как `_x`, `_y`, `_xscale`, `_yscale`) не имеют никакой связи с классом `MovieClip`, но при этом встроены в интерпретирующую программу.

Видеоклипы уникальны еще и в том, что они имеют собственную *временную шкалу* с одним или несколькими *ключевыми кадрами*, т.е. кадрами, в которые можно либо добавить код `ActionScript`, либо добавить или изменить графику, кнопку, текстовое поле или еще один видеоклип. Многие методы видеоклипов (такие как `gotoAndPlay()`, `stop()` и `prevFrame()`), три свойства видеоклипов (`_currentframe`, `_framesloaded` и `totalframes()`) и вездесущее событие `onEnterFrame` связаны с временной шкалой.

На заметку

Кнопки тоже имеют собственную временную шкалу, но она не поддерживает ни `ActionScript`, ни другие связанные с ней методы, свойства или события.

Помимо рабочего поля, единственными объектами во Flash, которые могут иметь графическое содержимое, являются кнопки и видеоклипы. Графику могут содержать и графические символы, но они не являются объектами.

В табл. 17.1 перечислены все методы класса `MovieClip`, за исключением методов рисования.

Совет

Методы рисования рассматриваются в главе 18 "Процедура рисования с помощью `ActionScript`".

ТАБЛИЦА 17.1. МЕТОДЫ КЛАССА `MOVIECLIP`

МЕТОД	ФОРМАТ	ОПИСАНИЕ
<code>attachAudio</code>	<code>myClip.attachAudio(поток);</code>	Присоединяет аудиопоток к видеоклипу
<code>AttachMovie</code>	<code>myClip.attachMovie (idИмя, новоеИмя, глубина [, initObject])</code>	Присоединяет фильм библиотеки к объекту <code>myClip</code> с указанной глубиной. Присоединенный фильм обладает всеми локальными свойствами объекта <code>init</code> . Возвращает ссылку на присоединенный фильм

МЕТОД	ФОРМАТ	ОПИСАНИЕ
AttachVideo	Нет	Нет
<code>createEmptyMovieClip</code>	<code>myClip.createEmptyMovieClip(имя_экземпляра, глубина)</code>	Создает пустой видеоклип как дочерний для объекта <code>myClip</code> , с указанной глубиной
CreateTextField	<code>myClip.createTextField(имя_экземпляра, глубина, x, y, ширина, высота)</code>	Создает пустое текстовое поле с указанным именем экземпляра, глубиной, координатами <code>x</code> и <code>y</code> , шириной и высотой
<code>DuplicateMovieClip</code>	<code>myClip.duplicateMovieClip(новоеИмя, глубина[, initObject])</code>	Дублирует объект <code>myClip</code> , создавая ответвление указанной глубины. Новый фильм обладает всеми локальными свойствами исходного объекта
<code>GetBounds</code>	<code>myClip.getBounds(targetCoordinateSpace)</code>	Возвращает минимальные и максимальные координаты <code>x</code> и <code>y</code> объекта <code>myClip</code> по отношению к указанному координатному пространству
<code>GetBytesLoaded</code>	<code>myClip.getBytesLoaded()</code>	Возвращает число байтов, загруженных для объекта <code>myClip</code>
<code>getBytesTotal</code>	<code>myClip.getBytesTotal()</code>	Возвращает размер объекта <code>myClip</code> в байтах
<code>GetDepth</code>	<code>myClip.getDepth()</code>	Возвращает глубину объекта <code>myClip</code>
<code>getURL</code>	<code>myClip.getURL(URL[, окно, переменные])</code>	Загружает документ в окно браузера из указанного URL, передавая все переменные из источника <code>myClip</code> с помощью методов <code>GET</code> или <code>POST</code> , указанных в строке переменные
<code>GlobalToLocal</code>	<code>myClip.globalToLocal(point)</code>	Преобразовывает объект <code>point</code> из координат рабочего поля основного фильма в локальные координаты объекта <code>myClip</code>
<code>gotoAndPlay</code>	<code>myClip.gotoAndPlay(кадр)</code>	Отправляет воспроизводящую головку к определенному кадру объекта <code>myClip</code> и затем начинает воспроизводить фильм
<code>GotoAndStop</code>	<code>myClip.gotoAndStop(кадр)</code>	Отправляет воспроизводящую головку к определенному кадру объекта <code>myClip</code> и затем останавливает фильм

МЕТОД	ФОРМАТ	ОПИСАНИЕ
HitTest	<code>rayClip.hitTest (x, y, shapeFlag)</code>	Возвращает значение true , если окно объекта <code>myClip</code> перекрывает точку, заданную координатами X и y. <code>ShapeFlag</code> — это булево значение, которое определяет, была ли оценена (true) общая форма объекта <code>myClip</code> или только его окно (false)
	<code>myClip.hitTest (target)</code>	Возвращает значение true , если окно объекта <code>myClip</code> пересекается с окном целевого видеоклипа
loadMovie	<code>myClip.loadMovie ("url" [, переменные])</code>	Загружает фильм или JPEG-файл, указанный в "url", в объект <code>myClip</code> с помощью метода GET или POST, указанного в строке переменные
LoadVariables	<code>myClip.loadVariables ("url" , переменные)</code>	Загружает переменные из URL или другого места в объект <code>myClip</code>
localToGlobal	<code>myClip.localToGlobal (point)</code>	Преобразует объект <code>point</code> из локальных координат объекта <code>myClip</code> в глобальные координаты рабочего поля
nextFrame	<code>myClip.nextFrame ()</code>	Отправляет воспроизводящую головку к следующему кадру объекта <code>myClip</code>
play	<code>my.Clip.play ()</code>	Воспроизводит объект <code>myClip</code>
prevFrame	<code>myClip.prevFrame ()</code>	Отправляет воспроизводящую головку к предыдущему кадру объекта <code>myClip</code>
removeMovieClip	<code>myClip.removeMovieClip ()</code>	Удаляет объект <code>myClip</code> из временной шкалы, если он был создан с помощью методов <code>attachMovieO</code> , <code>duplicateMovieClip ()</code> или <code>createEmptyMovieClip ()</code>
SetMask	<code>myClip.setMask (maskMovieClip)</code>	Задаёт видеоклип как маску объекта <code>myClip</code>
Stop	<code>myClip.stop ()</code>	Приостанавливает воспроизведение объекта <code>myClip</code>

МЕТОД	ФОРМАТ	ОПИСАНИЕ
<code>startDrag</code>	<code>myClip.startDrag([lock, [left, top, right, bottom]])</code>	Делает объект <code>myClip</code> доступным для перетаскивания и начинает его перетаскивание. Если значение <code>lock</code> истинно, то указатель мыши остается в точке регистрации объекта <code>myClip</code> . Значения <code>left</code> , <code>top</code> , <code>right</code> и <code>bottom</code> являются координатами, за пределы которых объект <code>myClip</code> перетаскивать нельзя
<code>stopDrag</code>	<code>myClip.stopDrag()</code>	Останавливает перетаскивание любого видеоклипа (не только объекта <code>myClip</code>). Является точным эквивалентом глобальной функции <code>stopDrag()</code>
<code>SwapDepths</code>	<code>myClip.swapDepths(глубина)</code>	Помещает объект <code>myClip</code> на указанную глубину. Если на этой глубине уже есть какой-либо видеоклип, то объект <code>myClip</code> будет помещен в слой предыдущего уровня
	<code>myClip.swapDepths(target)</code>	Меняет уровни глубины объекта <code>myClip</code> и целевого видеоклипа
<code>unloadMovie</code>	<code>myClip.unloadMovie()</code>	Удаляет объект <code>myClip</code> , если он был загружен с помощью метода <code>loadMovie()</code> . Применим и для JPEG-файла, загруженного с помощью метода <code>loadMovie()</code>

Нет — недокументированный метод.

В табл. 17.2 перечислены все свойства класса `MovieClip`.

ТАБЛИЦА 17.2 СВОЙСТВА КЛАССА `MOVIECLIP`

Свойство	ОПИСАНИЕ
<code>_alpha</code>	Прозрачность/непрозрачность; целое число от 0 до 100
<code>_currentframe</code>	Номер текущего кадра; целое число; только для чтения
<code>_droptarget</code>	Строка, предназначенная только для чтения, содержащая абсолютный путь (написанный с использованием косой черты), по которому перетаскивался экземпляр видеоклипа
<code>enabled</code>	Булево значение, определяющее, был ли включен видеоклип кнопки
<code>focusEnabled</code>	Булево значение, определяющее, может ли видеоклип получить фокусировку
<code>_focusrect</code>	Булево значение, определяющее, обрамлен ли активный видеоклип желтым прямоугольником

свойство	ОПИСАНИЕ
<code>_framesloaded</code>	Целое число, означающее количество кадров, загруженных в память; только для чтения
<code>_height</code>	Высота экземпляра видеоклипа в пикселях; число с плавающей запятой
<code>_hitArea</code>	Видеоклип, определяющий активную область видеоклипа кнопки
<code>_quality</code>	Строка, определяющая качество отображения: LOW (Низкое), MEDIUM (Среднее), HIGH (Высокое), BEST (Наилучшее)
<code>_name</code>	Имя экземпляра; строка
<code>_parent</code>	Обращение к видеоклипу, являющемуся родительским для данного
<code>_rotation</code>	Число с плавающей запятой; число градусов поворота относительно первоначальной ориентации видеоклипа
<code>_soundbuftime</code>	Целое число; число секунд, которое звук находится в буфере перед началом воспроизведения
<code>tabChildren</code>	Булево значение, определяющее, включен ли дочерний видеоклип в автоматический порядок перехода по клавише табуляции
<code>tabEnabled</code>	Булево значение, определяющее , включен ли видеоклип в порядок перехода по клавише табуляции
<code>tabIndex</code>	Определяет для видеоклипа порядок перехода по клавише табуляции
<code>_target</code>	Путь к целевому видеоклипу
<code>_totalframes</code>	Общее число кадров в видеоклипе
<code>trackAsMenu</code>	Булево значение, определяющее, что если пользователь нажмет кнопку мыши на данном видеоклипе кнопки (или на кнопке) и отпустит ее на другом видеоклипе, то последний получит событие, соответствующее отпущенной кнопке
<code>_url</code>	Строка, предназначенная только для чтения; местоположение SWF-файла, из которого был загружен данный видеоклип
<code>useHandCursor</code>	Булево значение, включающее или отключающее для видеоклипа кнопки отображение курсора в виде руки
<code>_visible</code>	Булево значение, определяющее, является ли экземпляр видеоклипа скрытым или видимым
<code>_width</code>	Ширина экземпляра видеоклипа в пикселях; число с плавающей запятой
<code>_x</code>	Координата x экземпляра видеоклипа; число с плавающей запятой
<code>_xmouse</code>	Координата x курсора мыши в пределах экземпляра видеоклипа; число с плавающей запятой
<code>_xscale</code>	Число с плавающей запятой, указывающее процентную величину масштабирования экземпляра видеоклипа по горизонтали
<code>_y</code>	Координата y экземпляра видеоклипа; число с плавающей запятой
<code>_ymouse</code>	Координата y курсора мыши в пределах экземпляра видеоклипа; число с плавающей запятой
<code>_yscale</code>	Число с плавающей запятой, указывающее процентную величину масштабирования экземпляра видеоклипа по вертикали

НОВЫЕ СВОЙСТВА ВИДЕОКЛИПОВ во FLASH MX

Во Flash MX добавлено девять новых свойств видеоклипов. Шесть из них (`_focusrect`, `enabled`, `focusEnabled`, `hitArea`, `trackAsMenu` и `useHandCursor`) включают и отключают присущие видеоклипу или кнопке видеоклипа функциональные возможности. Во Flash 5 имелась глобальная переменная `_focusrect`, которая существует и во Flash MX. Кроме того, кнопки и видеоклипы теперь обладают свойством `_focusrect`.

Совет

Дополнительная информация о свойстве `_focusrect` приведена ниже, во врезке "Включение и отключение обозначающего фокусировку желтого прямоугольника".

Совет

Подробная информация о видеоклипах кнопок приведена в главе 16 "Взаимодействие, события и установление последовательности".

Три новых свойства видеоклипа — `tabChildren`, `tabEnabled` и `tabIndex` — управляют порядком перехода по клавише табуляции, т.е. последовательностью, в которой текстовые поля, кнопки и/или другие видеоклипы становятся активными, когда пользователь нажимает клавишу `<Tab>` или `<Shift+Tab>`. Наиболее распространенным примером применения этих свойств является заполнение формы и использование клавиши `<Tab>` для перехода от одного ее поля к другому.

Если пользователь не определяет порядок перехода по клавише табуляции, Flash устанавливает его автоматически. Это означает, что порядок, в котором элементы становятся активными, будет установлен по умолчанию. Например, при создании формы с рядом текстовых полей ввода данных, располагающимися в виде одного столбца, наиболее приемлемым порядком перехода между полями, принятым по умолчанию, будет направление сверху вниз. Помимо этого, пользовательский порядок перехода по клавише табуляции можно задать явным образом. При этом установленный автоматический порядок перехода аннулируется по всему документу, с учетом SWF-файлов, загруженных с помощью метода `loadMovie()` или `loadMovieNum()`. Пользовательский порядок перехода бывает необходим при создании сложных форм.

Включение и отключение желтого прямоугольника, обозначающего фокусировку

В среде Flash кнопка или видеоклип кнопки, имеющие в данный момент клавиатурную фокусировку, обрамляются прямоугольником желтого цвета, за исключением случаев, когда эта функция была каким-либо образом отключена. Включить и отключить обрамление желтым прямоугольником можно глобально с помощью глобального свойства `_focusrect`. Если оно установлено в значение `true` (по умолчанию), то желтый прямоугольник будет отображаться. Если свойство `_focusrect` установлено в значение `false` (как в приведенном ниже примере), то желтый прямоугольник отображаться не будет. В этом случае при получении клавиатурной фокусировки для кнопки и видеоклипа кнопки будет отображено состояние *Over*.

```
_focusrect = false; // прямоугольник фокусировки отключается глобально
```



Во Flash MX желтый прямоугольник, обозначающий фокусировку, можно включить и отключить локально для отдельного видеоклипа или экземпляра кнопки с помощью свойства `_focusrect` данного экземпляра. Если свойство `_focusrect` установлено в значение `true` (по умолчанию), то когда экземпляр получит фокусировку, желтый прямоугольник будет отображаться. Если свойство `focusrect` установлено в значение `false` (как в приведенном ниже примере), то когда экземпляр получит фокусировку, желтый прямоугольник отображаться не будет. В этом случае при получении клавиатурной фокусировки кнопка или видеоклип кнопки будут отображены в состоянии *Over*.

```
MyButton._focusrect = false; // для экземпляра MyButton прямоугольник фокусировки не отображается
```

СВОЙСТВО ENABLED

Свойство `enabled` включает и отключает для видеоклипа функциональные возможности, присущие кнопкам. По умолчанию это свойство установлено в значение `true`. Если его установить в значение `false`, как показано в приведенном ниже примере, не будет отображаться курсор в виде руки, не будут вызываться обработчики события кнопки и, кроме того, отключатся кадры `_over`, `_down` и `_up`.

```
MyButtonClip.enabled = false; // больше не функционирует как кнопка
```

Даже когда свойство `enabled` установлено в значение `false`, видеоклип все равно будет включен в порядок перехода по клавише табуляции и будут продолжать функционировать обработчики событий классов `MovieClip`, `Mouse` и `Key`.

СВОЙСТВО FOCUSENABLED

Свойство `focusEnabled` позволяет включать и отключать для видеоклипа обработчики событий, связанные с фокусировкой.

Текстовые поля, кнопки, видеоклипы кнопок и обычные видеоклипы могут иметь обработчики событий `onSetFocus` и `onKillFocus`, которые активизируются в ответ на выполнение функции `Selection.setFocus()`. Для текстовых полей, кнопок и видеоклипов кнопок эти два обработчика событий, связанные с фокусировкой, *включены* по умолчанию. Для обычных видеоклипов эти обработчики событий по умолчанию *отключены*. Если для обычного видеоклипа свойство `focusEnabled` установить в значение `true`, то его обработчики событий `onSetFocus` и `onKillFocus` будут активизироваться в ответ на выполнение функции `Selection.setFocus()`.

Приведенный ниже код взят из файла `focusenabled fla`.

```
myClip.onSetFocus = function(oldFocus) {  
    trace("focus was "+oldFocus+ ", new focus is myClip");  
};  
myClip.focusEnabled = true; // позволяет активизировать  
myClip.onSetFocus  
Selection.setFocus(myClip); // myClip.onSetFocus активизируется
```

Ниже приведены четыре факта о свойстве `focusEnabled`.

- По умолчанию оно установлено в значение `undefined`, которое является эквивалентом значения `false`.
- В нем нет необходимости, если свойство `tabEnabled` установлено в значение `true`. (Свойство `tabEnabled` включает фокусировку.)
- Оно не применимо к текстовым полям, кнопкам или видеоклипам кнопок.
- Оно включает обработчики событий, связанные с *одним* видеоклипом (в предыдущем примере это был видеоклип `myClip`). `myClip.onSetFocus` будет активизироваться только тогда, когда объект `myClip` получит фокусировку. Если вы хотите, чтобы событие `myClip.onSetFocus` активизировалось при *каждом* изменении фокусировки (для других объектов наряду с `myClip`), то вместо этого или в дополнение к установке свойства `focusEnabled` в значение `true` задайте приемник:

```
Selection.addListener(myClip);
```

Совет

Подробная информация о приемниках приведена в главе 16 "Взаимодействие, события и установление последовательности".

Если сделано и то и другое, т.е. задан приемник и свойство `focusEnabled` установлено в значение `true`, то при получении фокусировки объектом `myClip` событие `myClip.onSetFocus` будет активизироваться дважды.

Свойство `focusEnabled` можно использовать и для того, чтобы фокусировку получал видеоклип, не имеющий обработчика события `onSetFocus`. В приведенном ниже примере кода, взятого из файла `focusenabled2 fla`, содержащегося на прилагаемом к книге компакт-диске, фокусировку получает видеоклип `myMc`, а событие принимает еще один объект (`myObj`).

```
function MyClass () {}
MyClass.prototype.onSetFocus = function() {
    trace("MyClass.prototype.onSetFocus");
}
myObj = new MyClass();
myMc.focusEnabled=true;
Selection.addListener(myObj);
Selection.setFocus(myMc);
```

СВОЙСТВО HITAREA

Свойство `hitArea` позволяет назначить любой видеоклип активной областью. Для видеоклипов кнопок (как в приведенном ниже примере) при наведении мыши на видеоклип, являющийся активной областью, ее указатель примет вид руки, и на видеоклипе кнопки можно будет щелкнуть:

```
myButtonClip.hitArea = myButtonClip.myHitClip;
```

В любой момент свойство `hitArea` можно переназначить для другого видеоклипа. Можно изменить размер и форму видеоклипа, являющегося активной областью. Кроме того, его можно сделать невидимым, не изменяя доступность видеоклипа кнопки для нажатия. Если свойство `hitArea` не назначать или назначить его для фактически несуществующего объекта, то по умолчанию видеоклип кнопки сам станет активной областью.

Включение функции активной области

Само по себе назначение активной области для обычного видеоклипа не играет никакой роли. Даже добавление свойства `useHandCursor = true` (описанного ниже в этой главе) не будет иметь никакого эффекта. Чтобы кнопка стала доступной для нажатия, а курсор принимал вид руки, для видеоклипа необходимо назначить обработчик события, связанный с кнопкой, что делает его видеоклипом кнопки. В этом случае функция активной области тоже будет работать.

СВОЙСТВО TABCHILDREN

Свойство `tabChildren` позволяет исключать и впоследствии включать в порядок перехода по клавише табуляции видеоклип, являющийся дочерним по отношению к рассматриваемому видеоклипу. Это свойство может быть полезным, к примеру, если определенные поля формы нужно заполнять только при определенных условиях. Когда эти условия наступают, свойство `tabChildren` можно установить в значение `true`.

По умолчанию видеоклипы, являющиеся дочерними по отношению к включенным в порядок перехода по клавише табуляции, в эту последовательность не попадают. Таким образом, если вы хотите трактовать имеющийся компонент пользовательского интерфейса, состоящий из нескольких видеоклипов, как единый объект перехода по клавише табуляции, дополнительно предпринимать ничего не нужно.

Если вы хотите, чтобы иногда дочерние видеоклипы были включены в порядок перехода по клавише табуляции, а иногда были исключены из него, то сначала их нужно включить в

этот порядок (например, с помощью свойства `tabEnabled`). Затем, если дочерний видеоклип нужно исключить из порядка перехода по клавише табуляции, установите свойство `tabChildren` в значение `false`. Если возникнет необходимость повторного включения, установите свойство `tabChildren` в значение `true`.

Если свойство `tabChildren` установлено в значение `undefined` (по умолчанию), то дочерние видеоклипы будут включены в порядок перехода по клавише табуляции как только вы установите свойство `tabEnabled` в значение `true` или сделаете видеоклип видеоклипом кнопки.

СВОЙСТВО `TABENABLED`

Свойство `tabEnabled` включает видеоклип в автоматический или пользовательский порядок перехода по клавише табуляции, в зависимости от того, какой из них задан в текущем документе. При этом данный видеоклип сможет принимать фокусировку.

СВОЙСТВО `TABINDEX`

Изменение значения свойства `tabIndex` на положительное целое число (с установленного по умолчанию значения `undefined`) позволяет включить видеоклип в порядок перехода по клавише табуляции и переносить на него фокусировку. Место видеоклипа в порядке получения фокусировки определяется заданным числом: видеоклип с меньшим индексом табуляции получит фокусировку раньше, чем видеоклип с большим индексом.

Задание в документе хотя бы одного свойства `tabIndex` приводит к отключению автоматического порядка перехода по клавише табуляции во всем документе.

СВОЙСТВО `TRACKASMENU`

Как правило, событие `onRelease` происходит в случае, когда пользователь *нажимает и отпускает* кнопку мыши в то время, как курсор находится над какой-либо кнопкой или видеоклипом кнопки. При работе с раскрывающимся меню может возникнуть ситуация, когда пользователь должен будет щелкнуть на одной кнопке (что приведет к разворачиванию меню), а затем переместить указатель мыши к кнопке, соответствующей выбранной опции и только потом отпустить кнопку мыши. Это поведение реализуется посредством свойства `trackAsMenu` ("отслеживать как меню").

Обработчик события `onPress` присоединяется к кнопке или видеоклипу кнопки, отображающему раскрывающееся меню. Для остальных кнопок или видеоклипов кнопок назначают событие `onRelease` и поведение "отслеживания как меню".

Назначить поведение "отслеживания как меню" для кнопки можно двумя способами: посредством панели `Properties` (Свойства) и `ActionScript`, а также с помощью свойства `trackAsMenu`. Для видеоклипа кнопки можно использовать только `ActionScript`.

В файле `options.fla` для одного пункта раскрывающегося меню поведение отслеживания назначено посредством `ActionScript` (как показано в приведенной ниже строке), а для остальных пунктов — с помощью панели `Properties`. В файле `options.fla` используются видеоклипы кнопок и `ActionScript`. В файле `wapsec.fla` демонстрируется поведение отслеживания, заданное по умолчанию.

```
option1.trackAsMenu = true;
```

СВОЙСТВО `USEHANDCURSOR`

По умолчанию для видеоклипов кнопок курсор отображается в виде руки. Чтобы активировать это поведение, установите свойство `useHandCursor` в значение `false` (как пока-

зано в приведенном ниже примере), а чтобы снова включить его, установите свойство `useHandCursor` в значение `true`.

```
MyButtonClip.useHandCursor = false;
```

Совет

Пример использования свойства `useHandCursor` будет рассмотрен в файле `spacelisten.fla` далее в этой главе, в разделе "Параметры метода `startDrag()`".

Курсор не будет принимать вид руки и в том случае, если в значение `false` установлено свойство `enabled` или `tabEnabled` видеоклипа.

Если установить свойство `useHandCursor` в значение `true` для обычного видеоклипа, это не будет иметь никакого эффекта. Чтобы на видеоклипе можно было щелкать мышью и курсор мог принимать вид руки, к видеоклипу следует присоединить обработчик события, придающий ему функциональные возможности кнопки. Затем можно будет подавить такое отображение курсора, используя предложение `useHandCursor = false`.

СОЗДАНИЕ И УДАЛЕНИЕ ВИДЕОКЛИПОВ

Создавать видеоклипы можно четырьмя способами, определяющими место видеоклипа в иерархической структуре и его расположение в рабочем поле.

На заметку

Расположение видеоклипа определяется его точкой *регистрации*, помеченной символом перекрестия при редактировании видеоклипа на панели Library (Библиотека). По умолчанию точкой регистрации является центр видеоклипа, которая в основной временной шкале размещена в верхнем левом углу.

Кроме того, можно удалять, замещать видеоклипы или удалять только графическое содержимое, не затрагивая сам объект видеоклипа.

Совет

С понятием точки регистрации мы познакомились в главе 6 "Символы, экземпляры и элементы библиотек".

Совет

Подробная информация о визуальном порядке наложения представлена ниже, в разделе "Управление визуальным порядком наложения видеоклипов".

Иерархическая структура с родительскими и дочерними элементами

Во Flash иерархическая структура видеоклипов, определяющая *визуальный порядок наложения*, основана на принципе родительских и дочерних связей.

Родительско-дочерняя связь между двумя видеоклипами является отношением типа "контейнер-содержимое", т.е. родительский элемент содержит дочерний. Родительский видеоклип может иметь любое количество дочерних, но дочерний видеоклип имеет только один родительский. Два дочерних элемента, имеющие одного "родителя", являются его *потомками*.

СОЗДАНИЕ ВИДЕОКЛИПОВ

Ранее уже упоминалось о том, что существуют четыре способа создания видеоклипов. В каждом из приведенных ниже примеров создается видеоклип `clip2`, являющийся дочерним объектом видеоклипа `clip1`. В последних трех примерах `clip2` имеет *показатель глубины*, равный 1.

Более подробно показатели глубины будут рассмотрены ниже, в подразделе "Визуальное наложение элементов, являющихся потомками одного родителя: показатели глубины".

Ниже представлены четыре способа создания видеоклипов.

- **Вручную.** При работе в среде проектирования, если `clip1` и `clip2` уже существуют в библиотеке, сначала откройте `clip1` на панели Library, а затем перетащите `clip2` в `clip1`. Внутри видеоклипа `clip1` можно создать и выделить графическое изображение. Затем выполните команду **Insert**⇒**Convert to Symbol** (Вставка⇒Преобразовать в символ), в поле Name (Имя) введите `clip2`, установите переключатель в поле опции Movie Clip (Видеоклип) и щелкните на кнопке ОК, чтобы преобразовать графику в видеоклип с именем `clip2`.
- **С помощью метода `attachMovie()` класса `MovieClip`.** Если во время выполнения программы видеоклип `clip1` находится в рабочем поле, а видеоклип, находящийся в библиотеке, имеет связь ID `clip`, то приведенное ниже предложение создаст видеоклип с именем `clip2`, расположит его в рабочем поле, совместит его точку регистрации с аналогичной точкой видеоклипа `clip1` и сделает видеоклип `clip2` дочерним объектом видеоклипа `clip1`.

```
clip1.attachMovie("clip", "clip2", 1);
```

- **С помощью метода `duplicateMovieClip()` класса `MovieClip` или с помощью глобальной функции `duplicateMovieClip()`.** Если во время выполнения программы в рабочем поле располагается объект `myClip`, являющийся дочерним по отношению к видеоклипу `clip1`, то любое из приведенных ниже предложений создаст видеоклип `clip2`, разместит его в рабочем поле, совместит его точку регистрации с аналогичной точкой объекта `myClip` и сделает `clip2` дочерним объектом видеоклипа `clip1`.

```
myClip.duplicateMovieClip("clip2", 1);
duplicateMovieClip("myClip", "clip2", 1);
```

- **С помощью метода `createEmptyMovieClip()` класса `MovieClip`.** Если во время выполнения программы в рабочем поле располагается видеоклип `clip1`, то приведенное ниже предложение создаст видеоклип с именем `clip2`, разместит его в рабочем поле, совместит его точку регистрации с аналогичной точкой видеоклипа `clip1` и сделает `clip2` дочерним объектом видеоклипа `clip1`.

```
Clip1.createEmptyMovieClip("clip2", 1);
```

- Если метод `createEmptyMovieClip()` используется для создания видеоклипа в корневом каталоге, то регистрационная точка нового видеоклипа будет совмещена с верхним левым углом рабочего поля.

На заметку

Кнопки можно присоединять с помощью метода `attachMovie()`. Их нельзя дублировать программным образом и нельзя удалять. К кнопкам не применимы методы `removeMovieClip()` и `unloadMovie()`.



Методы `attachMovie()`, `duplicateMovieClip()` и `createEmptyMovieClip()` возвращают ссылку на создаваемые ими видеоклипы. Это новая недокументированная функция Flash MX.

Во Flash 5 разработчики нередко создавали ссылки на часто используемые видеоклипы, поскольку оценка ссылки задействует гораздо меньше ресурсов процессора, чем оценка строки. Например, такая ссылка создается во втором из приведенных ниже предложений:

```
clip1.attachMovie("clip", "myClip"+i, i); // присоединение фильма  
ref = clip1["myClip"+i]; // создание ссылки
```

Во Flash MX ту же самую задачу можно выполнить посредством одного предложения:

```
ref = clip1.attachMovie("clip", "clip"+i, i);
```

УДАЛЕНИЕ ВИДЕОКЛИПОВ

Метод `removeMovieClip()` класса `MovieClip` можно использовать для удаления видеоклипа, созданного с помощью методов `attachMovie()`, `duplicateMovieClip()` или `createEmptyMovieClip()`. Если вам больше не нужна графика или код, содержащиеся в видеоклипе, то их удаление позволит высвободить часть памяти, которую занимает видеоклип и (особенно при удалении большого числа видеоклипов) позволит вашей программе работать более производительнее. Ниже приведен пример:

```
myClip.removeMovieClip(); // myClip удален
```

Кроме того, любой существующий видеоклип можно заменить. Для этого необходимо создать новый дочерний видеоклип, имеющий тот же показатель глубины, что и существующий.

С помощью метода `unloadMovie()` можно удалить графическое содержимое видеоклипа, оставив объект видеоклипа "пустым". Пустой видеоклип можно использовать в качестве контейнера, например для присоединения видеоклипов из библиотеки или для загрузки внешнего содержимого.

В отличие от `unloadMovie()`, метод `removeMovieClip()` полностью удаляет весь видеоклип. Если видеоклип удален с помощью этого метода, то к нему нельзя ничего присоединить и нельзя ничего загрузить в него — видеоклип просто перестает существовать.

ЗАГРУЗКА и ВЫГРУЗКА ВНЕШНЕГО СОДЕРЖИМОГО

С появлением во Flash 5 метода `attachMovie()` загрузка внешних SWF-файлов стала гораздо меньшей необходимостью. Однако в некоторых ситуациях она все же требуется.

Предположим, что Flash-приложение предоставляет доступ к нескольким SWF- и/или JPEG-файлам, а пользователь хочет получить доступ только к небольшой части этого содержимого. Не имеет смысла заставлять каждого пользователя загружать все содержимое, как происходит в случае, когда оно хранится в библиотеке и загружается с помощью метода `attachMovie()`.

Кроме того, возможно, вы захотите создать альтернативные "оболочки" приложения, каждой из которых будет пользоваться ограниченное число пользователей.

Вас могут также попросить "создать кадр" существующего SWF-файла, добавив в него вступление, новую музыку и заключительные титры. Самый легкий способ выполнения этой задачи заключается в написании приложения, содержащего дополнительные элементы, и загрузки в него существующего SWF-файла.

Еще одна причина загрузки внешних SWF-файлов заключается в том, чтобы предоставить пользователю возможность выбора различных версий содержимого, например большого изображения с высоким разрешением или маленького — с низким.

ЗАГРУЗКА ВНЕШНИХ SWF- или JPEG-ФАЙЛОВ

Внешний SWF- или JPEG-файл можно загрузить тремя способами: с помощью глобальных функций `loadMovieNum()` и `loadMovie()`, а также с помощью метода `loadMovie()` класса `MovieClip`. Ниже приводятся примеры каждого из способов загрузки.

```
loadMovieNum("test.swf", 1); // глобальная функция
loadMovie("test.swf", "myClip"); // глобальная функция
myClip.loadMovie("test.swf"); // метод MovieClip
```

При загрузке с помощью глобальной функции `loadMovieNum()` SWF- или JPEG-файл можно либо поместить на существующий уровень, либо создать для него новый. При загрузке с помощью метода `loadMovie()` класса `MovieClip` внешний файл загружается в существующий видеоклип, т.е. принимает уровень этого видеоклипа.

Совет

Подробная информация об уровнях представлена ниже, в разделе "Управление визуальным порядком наложения видеоклипов".

Глобальную функцию `loadMovie()` можно использовать для загрузки содержимого в *видео*клип, как показано в предыдущем примере. Кроме того, ее можно использовать для загрузки *в уровень*, который указывается посредством строки или числа:

```
loadMovie("test.swf", "_level1"); // строка "_level1"
loadMovie("test.swf", 1); // число 1
```

Для указания уровня нельзя использовать переменную. Так, приведенный ниже код работать не будет.

```
x = 15;
loadMovie("test.swf", x); // ЗАГРУЗКА НЕ ВЫПОЛНЯЕТСЯ!
```

Некоторые на первый взгляд разумные нестроковые значения приводят к тому, что загрузка с помощью глобальной функции `loadMovie()` происходит на уровне 0, а это означает, что полностью заменяется содержимое фильма. Ниже приведен вполне невинно выглядящий пример массового разрушения.

```
loadMovie("test.swf", "_level0"); // ЗАГРУЗКА В УРОВЕНЬ 0!!
```

ВИДЕОКЛИПЫ с ВНЕШНИМ СОДЕРЖИМЫМ ДУБЛИРОВАТЬ НЕЛЬЗЯ

После загрузки содержимого в фильм с помощью метода `loadMovie()` этот фильм дублировать нельзя.

Из этого правила есть одно небольшое исключение. При выполнении метода `duplicateMovie()` в том же самом блоке кода, к которому применялся метод `loadMovie()`, можно использовать `duplicateMovie()` для копирования видеоклипа с *исходным содержимым* (не с тем, которое загружалось). Например, приведенный ниже код будет выполняться, но он *не* дублирует загруженное содержимое, как можно было бы ожидать.

```
loadMovie("test.swf", _root.clip1);
_root.clip1.duplicateMovieClip("dupetest", 1);
```

Ни приведенное выше правило, ни исключение из него не применимы к функции `loadMovieNum()`.

ЗАГРУЖЕННЫЕ ФИЛЬМЫ ПРИСОЕДИНЯЮТ ТОЛЬКО ИЗ ЭТИХ БИБЛИОТЕК

Если в видеоклип `myClip` с помощью метода `loadMovie()` загружено содержимое, то к объекту `myClip` больше нельзя присоединять видеоклипы из его библиотеки. Можно присоединять содержимое из библиотеки, связанной с загруженным содержимым.

Предположим, что файл `test.swf` был загружен в объект `_root.myClip` посредством такого предложения:

```
loadMovie("test.swf", _root.myClip);
```

Затем приведенную ниже строку вы помещаете в обработчик события кнопки `onRelease`.

```
_root.myClip.attachMovie("linkageID", "newClip", 1);
```

Если `linkageID` является идентификатором связи видеоклипа в той же библиотеке, в которой определен символ объекта `myClip`, то приведенное выше предложение работать не будет. С другой стороны, если `linkageID` является идентификатором связи видеоклипа в библиотеке файла `test.swf`, то после щелчка на кнопке этот видеоклип будет присоединен к объекту `myClip`, в результате чего будет создан объект

```
_root.myClip.newClip
```

Совет

При возникновении проблем, связанных с присоединением фильма из библиотеки загруженного SWF-файла или с доступом к переменной, видеоклипу или функции загруженного SWF-файла, обратитесь к разделу "Возможные проблемы" в конце этой главы.

ВЫГРУЗКА ВНЕШНЕГО SWF- или JPEG-ФАЙЛА

Внешние SWF- или JPEG-файлы, загруженные с помощью метода `loadMovie()`, можно выгрузить с помощью метода `unloadMovie()`. Для загрузки и выгрузки можно использовать глобальную функцию либо метод `MovieClip` или глобальную функцию для одной из операций, а метод `MovieClip` — для другой. Вот несколько примеров:

```
loadMovie("test.swf", _root.clip1); // загрузка "test.swf" в
                                   // _root.clip1
_root.clip2.loadMovie("test.swf"); // загрузка "test.swf" в
                                   // _root.clip2
unloadMovie("_root.clip2"); // выгрузка чего-либо, содержащегося в
                           // _root.clip2
clip1.unloadMovie(); // выгрузка чего-либо, содержащегося в clip1
```

Аналогичным образом, загрузку можно выполнять с помощью функции `loadMovieNum()`, а выгрузку — с помощью функции `unloadMovieNum()`:

```
loadMovieNum("test.swf", 1);
unloadMovieNum(1);
```

После выгрузки с помощью метода `unloadMovie()` остается пустой видеоклип, т.е. даже после выполнения предложения `clip1.unloadMovie()` объект `clip1` остается, и к нему снова можно присоединить какое-то содержимое:

```
clip1.loadMovie("new.swf");
```

Можно просто загрузить новое содержимое, не выполняя предложение `clip1.unloadMovie()`. При загрузке в видеоклип нового содержимого старое содержимое автоматически выгружается.

Метод `unloadMovie()` можно использовать для выгрузки графического содержимого из видеоклипа, созданного вручную, после чего останется пустой видеоклип.

Для указания фильма, подлежащего выгрузке с помощью метода `unloadMovie()`, лучше всего использовать строку.

Глобальная функция `unloadMovie()` в качестве параметра может также принимать ссылку на видеоклип, а не на строку. Например, приведенное ниже предложение является разрешенным.

```
unloadMovie(_root.clip2); // выгрузка чего-либо, содержащегося в
                           // _root.clip2
```

Однако при указании ссылки на несуществующий видеоклип глобальная функция `unloadMovie()` выгрузит содержимое из той временной шкалы, в которой находится предложение. Гораздо безопаснее в качестве параметра глобальной функции `unloadMovie()` использовать только строки.

УПРАВЛЕНИЕ ВИЗУАЛЬНЫМ ПОРЯДКОМ НАЛОЖЕНИЯ ВИДЕОКЛИПОВ

Во Flash-документе видеоклипы имеют такой вид, как если бы они располагались на прозрачной пленке. Это означает, что при отсутствии графического содержимого на верхнем слое будут видными элементы на нижних слоях. Процедура управления тем, какие фильмы располагаются в рабочем поле впереди или позади других фильмов, называется *визуальным порядком наложения*. Для того чтобы грамотно манипулировать порядком наложения, необходимо четко понимать иерархическую структуру видеоклипов во Flash-документе.

В основу иерархической структуры видеоклипов заложены два принципа: "бутерброда" и "дерева". В частности, структура Flash-документа представляет собой *уровни*, наложенные друг на друга подобно начинке многослойного бутерброда. В пределах каждого уровня видеоклипы образуют древовидную структуру, представляющую собой *иерархию с родительскими и дочерними элементами*, а визуальный порядок наложения потомков (дочерние видеоклипы одного родительского) определяется показателями их глубины. Таким образом, общий Flash-документ представляет собой некий "бутерброд, состоящий из деревьев". Если фильм одноуровневый, то дело придется иметь только с древовидной структурой.

Идентификатор `_root` означает основание каждого дерева. Каждый уровень имеет собственный идентификатор `_root`, или основную временную шкалу. Таким образом, две ссылки на `_root.myClip1` относятся к двум совершенно разным видеоклипам, если эти обращения возникают на двух разных уровнях. Например, один из видеоклипов может быть `_level10.myClip1`, а другой — `_level11.myClip1`. Обращения к видеоклипам, начинающиеся с индикатора уровня, являются однозначными.

Первоначальная иерархическая структура видеоклипов определяется пятью факторами:

- хронологическим порядком, в котором несколько видеоклипов были вручную помещены на один слой временной шкалы;
- относительным расположением слоев временной шкалы, содержащих видеоклипы;
- взаимосвязями типа "родитель-потомок" между видеоклипами;
- был ли создан видеоклип с помощью метода `duplicateMovie()`, `attachMovie()` или `createEmptyMovieClip()`;
- на каких уровнях располагаются видеоклипы.

УРОВНИ ВИЗУАЛЬНОГО ПОРЯДКА НАЛОЖЕНИЯ

Общая структура Flash-документа представляет собой иерархию, состоящую из одного или нескольких нумерованных *уровней*, начиная с уровня 0 (`_level0`). Иерархия уровня представляет собой простую структуру типа "бутерброд". Чем выше номер уровня, тем выше его положение в иерархической структуре. Например, `_level2` располагается на переднем плане относительно `_level1`, а `_level1` — на переднем плане относительно `_level0`.

Каждый документ должен иметь `_level0`, кроме него он может иметь любое количество уровней, максимальное количество которых составляет 16534. Содержимое не обязательно располагать на соседних уровнях. Например, оно может располагаться на уровнях 0, 100 и 1000, и между ними может не быть никакого содержимого.

При загрузке внешних SWF- или JPEG-файлов с помощью глобальной функции `loadMovieNum()` или `loadMovie()`, помимо уровня `_level0`, можно назначить (и создать) другие уровни.

Совет

Три способа загрузки внешних файлов были описаны ранее в этой главе, в подразделе "Загрузка внешних SWF- или JPEG-файлов".

Если при загрузке внешнего файла с помощью глобальной функции `loadMovieNum()` или `loadMovie()` вы специально не создадите новый уровень, то документ будет иметь только один уровень, `_level0`.

ВИЗУАЛЬНОЕ НАЛОЖЕНИЕ ЭЛЕМЕНТОВ, НЕ ЯВЛЯЮЩИХСЯ ПОТОМКАМИ ОДНОГО РОДИТЕЛЯ: ИЕРАРХИЧЕСКАЯ СТРУКТУРА С РОДИТЕЛЬСКИМИ И ДОЧЕРНИМИ ЭЛЕМЕНТАМИ

В пределах каждого уровня визуальное наложение определяется двумя факторами: иерархической структурой родительских и дочерних элементов и показателями глубины.

В иерархической структуре с родительскими и дочерними элементами каждый потомок является основателем своей "семейной ветви". Если посмотреть на семейную ветвь вертикально, то визуальный порядок наложения родительских и дочерних элементов прост: дочерний элемент всегда располагается на переднем плане относительно своего родительского.

Каждая семейная ветвь образует единый блок с целью установки визуального порядка наложения, как при задании вращения, масштабирования или местоположения. При работе с двумя видеоклипами, являющимися потомками одного родительского, не нужно думать о том, что каждый из них тоже может иметь дочерние, "прадочерные" видеоклипы и т.д. Способ определения визуального порядка наложения совместим с другими операциями Flash. Например, при вращении или перемещении видеоклипа поворачиваться либо перемещаться будет вся его семейная ветвь. Аналогичным образом, если в визуальном порядке наложения один видеоклип располагается "позади" другого, то такое размещение применимо к любой из семейных ветвей в пределах данных двух видеоклипов. Короче говоря, при работе с видеоклипом он рассматривается как единый блок.

На этом мы закончим рассмотрение иерархии родительских и дочерних элементов, определяющей визуальный порядок наложения для дочерних элементов, не являющихся потомками одного родительского.

ВИЗУАЛЬНОЕ НАЛОЖЕНИЕ ЭЛЕМЕНТОВ, ЯВЛЯЮЩИХСЯ ПОТОМКАМИ ОДНОГО РОДИТЕЛЯ: ПОКАЗАТЕЛИ ГЛУБИНЫ

Еще одним фактором, определяющим визуальный порядок наложения в пределах каждого уровня, является *показатель глубины*. Во Flash он используется для отслеживания нескольких факторов, которые все вместе определяют визуальный порядок наложения дочерних элементов, имеющих один родительский. Дочерний элемент с большим показателем глубины располагается на переднем плане относительно дочернего элемента, имеющего меньший показатель глубины.

Максимальное число показателей глубины, которые можно назначить для одной временной шкалы, составляет 16 384.

Разрешенные показатели глубины могут находиться в диапазоне от -16384 до 0x7FFFFFFD (или 2 130 706 429). Воспользуйтесь процедурой тестирования, чтобы убедиться, что вы сможете удалить видеоклипы с показателем глубины, большим 1 048 575.

Если разработчики используют очень большие показатели глубины, это наверняка является попыткой обезопасить себя от того, что для нового видеоклипа будет использован тот же показатель глубины, что и для существующего (что приведет к замене существующего видеоклипа новым).

АВТОМАТИЧЕСКОЕ И ЯВНОЕ НАЗНАЧЕНИЕ ПОКАЗАТЕЛЕЙ ГЛУБИНЫ

Показатели глубины могут назначаться автоматически или явным образом.

- Flash автоматически назначает показатели глубины для видеоклипов, которые вручную помещены в рабочее поле в процессе разработки фильма. Назначение показателей глубины зависит от трех факторов: хронологического порядка, в котором несколько видеоклипов помещено на один уровень временной шкалы; относительного расположения слоев временной шкалы, содержащих видеоклипы, и родительско-дочерних взаимосвязей между видеоклипами. Автоматически назначаемые показатели глубины всегда будут отрицательными, начиная с -16384 для корневого расположения и далее по возрастанию, возможно, с интервалами (например, -16383, -16382, -16380).

На заметку

Во Flash 5 не предусмотрен нижний предел показателей глубины, однако во Flash 6 он не может быть меньше **-16384**. Flash 6 делает исключение только для SWF-файлов старых версий.

- Для каждого видеоклипа, создаваемого с помощью методов `duplicateMovie()`, `attachMovie()` или `createEmptyMovieClip()`, показатель глубины необходимо назначить явным образом. С этой целью во Flash зарезервированы показатели от 1 до 16384, которые автоматически не присваиваются.
- Можно явным образом "заменить" показатели глубины двух существующих видеоклипов или с помощью метода `swapDepth()` класса `MovieClip` назначить для видеоклипа показатель глубины, который в данный момент не используется.
- Например, приведенная ниже строка назначает для видеоклипа `myClip` показатель глубины, равный 1. Если этот показатель глубины уже имеет какой-то дочерний элемент, то для этого элемента будет назначен текущий показатель глубины видеоклипа `myClip`.

```
myClip.swapDepths(1);
```

- Приведенная ниже строка меняет показатели глубины видеоклипов `otherClip` и `myClip`.

```
myClip.swapDepths(otherClip);
```

Совет

Во Flash MX метод `swapDepths()` является более надежным по сравнению с Flash 5.

Если явным образом назначаются только показатели глубины, большие 0, то конфликта между ними и автоматически назначаемыми никогда не будет, поскольку последние всегда меньше 0.



Использование метода `getDepth()`. Определить показатель глубины любого видеоклипа можно с помощью метода `getDepth()` класса `MovieClip`. С помощью этой информации можно, к примеру, сознательно назначить уже существующий показатель глубины (как отрицательный, так и положительный) для нового дочернего видеоклипа, что приведет к замене видеоклипа, созданного вручную или профамным способом. Приведенный ниже код, который подразумевает наличие в библиотеке символа со связью ID `clip`, уничтожает видеоклип `myClip1` в процессе создания видеоклипа `myClip2`.

```
dep = _root.myClip1.getDepth(); // получить показатель глубины
                                   // видеоклипа myClip1
_root.attachMovie("clip", "myClip2", dep); // myClip2 заменяет myClip1
```

НАЗНАЧЕНИЕ ГЛУБИНЫ `_ROOT` для ДИНАМИЧЕСКОГО СОЗДАНИЯ нового ФОНА

Позади любого содержимого, в том числе и авторского, можно динамически помещать фон, задав глубину `_root` нового видеоклипа, как показано ниже.

```
_root.attachMovie("newBackground", "myBG", -16384);
```

На заметку

Для автоматически назначаемых показателей глубины термин "большой" означает "менее отрицательный". Например, -16380 больше, чем -16382.

АВТОМАТИЧЕСКОЕ НАЗНАЧЕНИЕ ПОКАЗАТЕЛЕЙ ГЛУБИНЫ

При автоматическом назначении показателей глубины Flash следует лишь двум основным правилам.

- Дочерний видеоклип, расположенный на более высоком слое временной шкалы, получает больший показатель глубины по сравнению с дочерним видеоклипом, расположенным на более низком слое временной шкалы.
- Новый дочерний видеоклип получает более высокий показатель глубины по сравнению с уже существующим. Это означает, к примеру, что если на один слой временной шкалы вручную поместить несколько видеоклипов, то в визуальном порядке наложения видеоклипы, помещаемые позднее, будут автоматически располагаться выше.

Обратите внимание на то, что здесь ничего не говорится об относительных показателях глубины родительских и дочерних видеоклипов. Это не имеет значения. Показатель глубины определяет только то, как видеоклипы располагаются над своими дочерними элементами. Фактически при создании семейной ветви без дочерних элементов все видеоклипы ветви могут иметь один и тот же показатель глубины. Дочерние элементы, не являющиеся потомками одного родителя, могут иметь один и тот же показатель глубины из-за того, что их визуальный порядок наложения определяется родительско-дочерней иерархией.

ЯВНОЕ НАЗНАЧЕНИЕ ПОКАЗАТЕЛЕЙ ГЛУБИНЫ

При назначении показателей глубины явным образом выполните следующие два шага, чтобы убедиться в том, что не заменяется существующее содержимое.

- Во-первых, убедитесь, что существующие видеоклипы не будут потомками создаваемого видеоклипа. При использовании методов `attachMovie()` и `createEmptyMovieClip()` убедитесь в том, что предполагаемый родительский видеоклип не имеет уже существующих дочерних. При использовании метода `duplicateMovieClip()` убедитесь в том, что дублируемый видеоклип не имеет уже существующих потомков.

- Присвойте каждому новому видеоклипу новое имя экземпляра и назначьте для него более высокий показатель глубины по сравнению с предыдущим видеоклипом. Например, приведенный ниже блок кода создает 10 копий видеоклипа `myClip: myClip0` на глубине 1, `myClip1` на глубине 2 и т.д.

```
for (i = 0; i < 10; i++) {
    myClip.duplicateMovieClip("myClip"+i, i+1);
}
```

Если существующие видеоклипы будут потомками создаваемого видеоклипа, вам следует знать их показатели глубины. Для организации нужного визуального порядка наложения назначьте новые показатели глубины и при необходимости замените существующие дочерние видеоклипы.

Внимание!

Не используйте имя экземпляра видеоклипа повторно, в противном случае существующий видеоклип станет неадресуемым.

Запомните следующее.

- Видеоклип, созданный с помощью метода `attachMovie()`, становится *дочерним* по отношению к видеоклипу, к которому его присоединяют. Например, приведенная ниже строка создает видеоклип `_root.myClip1.myClip2`.

```
_root.myClip1.attachMovie("clip", "myClip2", 1);
```

- Видеоклип, созданный с помощью метода `createEmptyMovieClip()`, становится *дочерним* по отношению к видеоклипу, в котором он создан. Так, приведенная ниже строка создает видеоклип `_root.myClip1.myClip2`.

```
_root.myClip1.createEmptyMovieClip("myClip2", 1);
```

- Видеоклип, созданный с помощью метода `duplicateMovieClip()`, становится *потомком* фильма, который он дублирует. Так, приведенная ниже строка создает видеоклип `_root.myClip2`.

```
_root.myClip1.duplicateMovieClip("myClip2", 1);
```

ПРИМЕНЕНИЕ УРОВНЕЙ и ГЛУБИНЫ

Загрузить внешний SWF- или JPEG-файл в *уровень* можно с помощью глобальной функции `loadMovie()` или `loadMovieNum()`. Внешние SWF-файлы можно загружать в *видео-клипы* с помощью метода `loadMovie()` класса `MovieClip` или с помощью глобальной функции `loadMovie()`.

При наличии слоев временной шкалы, назначении показателей глубины, иерархии родительских и дочерних элементов и объектно-ориентированной модели, помогающих организовать содержимое, использование уровней в качестве инструмента систематизации может показаться ненужным. Ниже перечислено несколько причин, по которым вы можете предпочесть использовать в качестве целевых объектов загрузки видеоклипы, а не уровни.

- Имя видеоклипа может нести смысловую нагрузку, что сделает программу более удобочитаемой. Уровни всегда имеют обобщенные имена: `level0`, `_level1` и т.д.
- Видеоклипы позволяют более детально регулировать визуальный порядок наложения посредством слоев, родительско-дочерней иерархии и показателей глубины. При загрузке в видеоклип элемент `_root` загруженного SWF-файла становится видеоклипом, в который он загружен. При этом ранее находившееся там содержимое заменяется, а неизменной остается показатель глубины видеоклипа, положение слоев

и размещение в иерархической структуре родительских и дочерних элементов. (По аналогии, JPEG-файл, загруженный в видеоклип, становится видеоклипом.)

- При загрузке в уровень нельзя указать глубину и использовать ее для контроля замещаемого содержимого. Элемент `_root` SWF-файла, загруженного в уровень с помощью глобальной функции `loadMovieNum()` или `loadMovie()`, принимает значение `_root` уровня, в который была выполнена загрузка, при этом все ранее находившееся там содержимое замещается.

С другой стороны, уровни имеют свои преимущества.

- Иногда загрузка в уровень дает своего рода "полное изменение", которого необходимо достичь. Например, при загрузке фильма в уровень `_level0` все уровни выгружаются, а новый фильм становится уровнем `_level0`. Если после этого загрузить фильмы в другие уровни, то фильм, расположенный в уровне `_level0`, будет задавать свойства документа, такие как частоту смены кадров, фоновый цвет и размер кадров для всех уровней. Таким образом, загрузка в уровень `_level0` обеспечивает, например, согласованность работы группы разработчиков.
- Свойства видеоклипов, такие как `_visible` и `_alpha`, можно применить к уровню, не влияя при этом на остальные. Это дает возможность легко изменять свойства видеоклипов, не затрачивая другие фрагменты фильма.
- При загрузке в уровень SWF-файла, содержащего звук, можно сделать так, что потоковый звук будет воспроизводиться в нескольких сценах.
- При загрузке SWF-файла, использующего в предложениях `ActionScript` идентификатор `_root`, лучше всего он будет работать, будучи загруженным в уровень, потому что `_root` SWF-файла принимает значение `_root` уровня. В этом случае загрузка в видеоклип может вызвать много проблем. Например, переменные `_root` SWF-файла могут быть записаны поверх существующих переменных `_root` с тем же именем.

ИСПОЛЬЗОВАНИЕ ИСХОДНЫХ ОБЪЕКТОВ для ЗАДАНИЯ СВОЙСТВ НОВЫХ ВИДЕОКЛИПОВ

Оба метода класса `MovieClip`, `attachMovie()` и `duplicateMovieClip()`, при создании видеоклипа позволяют указать *исходный объект*. Flash автоматически присваивает новому видеоклипу все локальные свойства исходного объекта. Эта возможность является преимуществом метода `duplicateMovieClip` над одноименной глобальной функцией.

В приведенном ниже примере `myClip` является видеоклипом, к которому выполняется присоединение; `linkageID` — идентификатор связи видеоклипа в библиотеке; `newInstance` — имя нового видеоклипа; `l` — глубина нового видеоклипа, а `initObj` — имя объекта.

```
myClip.attachMovie("linkageID", "newInstance", 1, initObj);
```

В результате вновь созданный видеоклип `newInstance` будет обладать всеми локальными свойствами объекта `initObj`. Видеоклип `newInstance` *не будет* обладать свойствами, которые объект `initObj` наследует у объекта-прототипа своего класса.

Исходный объект можно представить в виде функционального литерала, например, следующим образом:

```
myClip.attachMovie("linkageID", "newInstance", 1, {_x:300, y:200 } );
```

Кроме того, новый исходный объект можно создать путем использования оператора `new` с функцией-конструктором.

Совет

Знакомство с принципами объектно-ориентированного программирования состоялось в главе 11 "Знакомство с `ActionScript`" и в главе 15 "Объединение предложений в функции". Подробная информация по этой теме приведена в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

Использование оператора `new` с функцией-конструктором для создания нового исходного объекта позволяет задать для нового видеоклипа локальные свойства, обычно назначаемые для экземпляров класса, без назначения нового видеоклипа к этому классу. Как обычно, при использовании оператора `new` ключевое слово `this` в пределах функции-конструктора относится к созданному объекту. В этом случае им является исходный объект.

В приведенном ниже примере в качестве исходного объекта используется новый анонимный (безымянный) экземпляр функции-конструктора `myClass`. В результате новый видеоклип `instance` получает локальное свойство класса `instanceVar`, но не свойство прототипа класса `prototypeVar`.

```
MyClass = function() {  
    this.instanceVar = "instanceVar";  
}  
MyClass.prototype.prototypeVar = "prototypeVar";  
depth = 1;  
this.attachMovie("clip", "instance", depth, new MyClass());  
for (a in instance) trace(a); // instanceVar  
trace(typeof instance);      // movieclip
```

Видеоклип `instance` получает локальные свойства, которыми обычно обладают экземпляры класса `MyClass`, но при этом объект `instance` остается видеоклипом, а не членом класса `MyClass`.

Таким образом, исходный объект предоставляет способ частичного сопряжения двух классов в пределах определенного экземпляра. В более строгом объектно-ориентированном языке, таком как `Java`, для видеоклипа `instance` нужно было бы создать новый подкласс. Благодаря созданию нового подкласса вместо наличия членов того же класса с различными типами свойств система классов становится более управляемой.

Функцией исходного объекта можно воспользоваться, если вы хотите создать подкласс для нового видеоклипа. Этот вопрос будет рассмотрен в следующем подразделе.

ИСПОЛЬЗОВАНИЕ МЕТОДА `Object.registerClass()` для ВЫДЕЛЕНИЯ НОВОГО ВИДЕОКЛИПА В ПОДКЛАСС

Несмотря на то что `registerClass()` официально является методом класса `Object`, он используется только применительно к видеоклипам. Этот метод нельзя применять к другим типам объектов, таким как, например, кнопки.

Метод `registerClass()` является удобным способом выделения новых видеоклипов в подклассы. Эта методика, которая используется для компонентов `Macromedia`, позволяет "предварительно зарегистрировать" видеоклип как член определенного класса.

Предположим, что в библиотеке имеется символ с именем `clip`, который вы хотите использовать для создания нового видеоклипа `myClip`, принадлежащего к подклассу `myClass`. Для символа `clip` задайте идентификатор связи, например `linkageID`, а затем воспользуйтесь методом `registerClass()`:

```
Object.registerClass("linkageID", myClass);
```

Когда символ *clip* реализован, Flash вызывает функцию `myClass()` без аргументов, но с переменной `this`, указывающей на создаваемый объект нового видеоклипа, в данном случае — на `myClip`. (Реализация происходит, когда символ `clip` помещают в рабочее поле, как вручную при разработке фильма, так и программным образом во время выполнения программы, с помощью метода `attachMovie` либо метода или глобальной функции `duplicateMovieClip()`.) В результате обращения к функции `myClass()` объект `myClip` становится членом класса `myClass` без потери присущих ему методов или свойств, в том числе и графического содержимого.

ЗНАКОМСТВО С МЕТОДОМ REGISTERCLASS()

Чтобы понять, насколько важен метод `registerClass()`, вспомните, что видеоклип является специальным типом объектов Flash. Его базовые свойства (которыми он обладал в предыдущих версиях Flash) являются "защитными" в интерпретирующую программу, и их можно запросить только в процессе создания видеоклипа вручную либо с помощью методов `createEmptyMovieClip()`, `attachMovie()` или `duplicateMovieClip()`. Даже несмотря на наличие класса `MovieClip`, эти базовые свойства никак с ним не связаны. Базовыми свойствами, без которых видеоклип не может существовать, являются `_x`, `_y`, `_height`, `_width`, `_alpha`, `_rotation`, `_xscale`, `yscale` и `_name`.

С другой стороны, все методы видеоклипа (такие как `play()`, `gotoAndPlay()` и `prevFrame()`) наследуются у класса `MovieClip` и, следовательно, присущи только членам этого класса.

В результате, если вы хотите, чтобы видеоклип принадлежал также пользовательскому классу, такому как `myClass`, то столкнетесь с дилеммой. Например, ведет "в никуда" на первый взгляд кажущийся очень обнадеживающий подход, заключающийся в том, что пользовательский класс наследует свойства класса `MovieClip` (`myClass.prototype = new MovieClip()`), а для создания нового объекта в этом классе используется оператор `new` (`myClip = new MyClass`). При этом подходе объект `myClip` наследует методы и свойства, присущие классу `MovieClip`, и становится полноправным членом класса `MyClass`, наследуя свойства объекта-прототипа этого класса. Однако объект `myClip` теряет базовые свойства видеоклипа, без которых видеоклипы не могут функционировать.

С другой стороны, если функция-конструктор используется как исходный объект, для того, чтобы задать видеоклипу свойства пользовательского класса (см. предыдущий подраздел), то объект останется полноценным функционирующим видеоклипом, но не станет членом класса `myClass`. В частности, он не будет наследовать свойства прототипа класса `myClass`.

Метод `registerClass()` позволяет сделать объект `myClip` полноценным членом класса `myClass` (наследующим свойства его объекта-прототипа) и при этом сохранить базовые свойства видеоклипа объекта `myClip`. Затем можно сделать так, что пользовательский класс `myClass` будет наследовать свойства класса `MovieClip`, и в результате мы получим полнофункциональный подкласс видеоклипа, которому будут присущи все методы и свойства, предлагаемые классом `myClass`, а также все свойства, которыми должен обладать видеоклип.

СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ МЕТОДОВ REGISTERCLASS() И ATTACHMOVIE()

Легче всего использовать метод `registerClass()` вместе с методом `attachMovie()`. В приведенном ниже примере `linkageID` — идентификатор связи видеоклипа в библиотеке, а `myClass` — имя функции-конструктора.

```
Object.registerClass("linkageID", MyClass);
MyClass.prototype = new MovieClip();
_root.attachMovie("linkageID", "myClip", 1);
```

Теперь мы имеем полнофункциональный видеоклип `myClip`, являющийся членом класса `MyClass`, который, в свою очередь, наследует свойства класса `MovieClip`.

ИСПОЛЬЗОВАНИЕ МЕТОДА `REGISTERCLASS()` С ВИДЕОКЛИПОМ, СОЗДАННЫМ ВРУЧНУЮ

Чтобы использовать метод `registerClass()` с видеоклипом, созданным вручную, необходимо прибегнуть к небольшой хитрости. Дело в том, что в предложении `registerClass()` нельзя использовать имя экземпляра видеоклипа вместо его идентификатора связи.

В фильме `regclass.fla` приведено решение, согласующееся со способом реализации компонентов Macromedia. Для видеоклипа задается идентификатор связи, а код инициализации, аналогичный приведенному ниже, помещают во временную шкалу видеоклипа.

```
ttinitclip
function MyClass () {
    trace("constructor is executing");
}
MyClass.prototype = new MovieClipO;
MyClass.prototype.myFunc = function () {
    trace("myFunc is executing");
};
object.registerclass("linkageID", MyClass);
ttendinitclip
```

Задание идентификатора связи для видеоклипа, который уже находится в рабочем поле, кажется алогичным, но эта методика работает.

При исполнении SWF-файла в выходном окне вы увидите фразу `constructor is executing` (конструктор выполняется).

Если созданным вручную видеоклипом является `_root.myClip`, то функцию `myFunc()` можно выполнить следующим образом:

```
_root.myClip.myFunc();
```

В этом случае в окне вы увидите фразу `myFunc is executing` (`myFunc` выполняется).

ИСПОЛЬЗОВАНИЕ СВОЙСТВА `_PROTO` ВМЕСТО МЕТОДА `REGISTERCLASS()`

В некоторых ситуациях метод `registerClass()` неприменим. Например, не существует очевидного способа применения этого метода к видеоклипам, созданным с помощью метода `createEmptyMovieClipO`. Чтобы обойти эту проблему и сделать видеоклип членом подкласса `MovieClip`, воспользуйтесь свойством `_proto_` экземпляра видеоклипа.

Совет

Подробная информация о свойстве `_proto_` представлена в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

При использовании того же подкласса, что и в предыдущем примере (`MyClass`), файл `proto.fla` создает новый пустой видеоклип `myHighlight` и делает его членом класса `MyClass`, не влияя на присущие ему свойства видеоклипа. Ниже приведены основные строки

кода. Строка 1 создает видеоклип, строка 2 делает его членом класса `myClass`. Строка 3 исполняет `myClass`, а ключевое слово `this` относится к объекту `myHighlight`. (Если классу `myClass` присваиваются локальные свойства, то данная процедура должна быть выполнена в этом месте.) Строка 4 подтверждает, что объект `myHighlight` действительно является членом класса `myClass`.

```
1: createEmptyMovieClip("myHighlight",1);
2: myHighlight.__proto__= MyClass.prototype;
3: MyClass.apply(myHighlight); // отображается "constructor is
                               // executing"
4: myHighlight.myFunc();      // отображается "myFunc is executing"
```

Совет

Подробная информация о методе `apply()` приведена в главе 15 "Объединение предложений в функции".

КАК РЕШИТЬ ПРОБЛЕМУ ПЕРЕДАЧИ ПАРАМЕТРОВ ПРИ ИСПОЛЬЗОВАНИИ МЕТОДА `REGISTERCLASS()`

Метод `registerClass()` не позволяет передавать аргументы функции-конструктору. Для компенсации этого недостатка можно воспользоваться исходным объектом. Однако фактически исходный объект тоже не передает никаких аргументов конструктору, поэтому конструктор нужно модифицировать так, чтобы он знал, где именно в исходном объекте следует искать значения, которые обычно являются аргументами.

В приведенном ниже листинге, взятом из файла `regclassinit fla`, строка 1 регистрирует идентификатор связи `clip` в классе `myClass`. Обратите внимание на то, что идентификатор связи можно зарегистрировать **еще** до существования класса.

Функция `registerClass()` возвращает значение, которое должно быть истинным, если операция проходит успешно; в противном случае возвращаемое значение будет ложным. Однако функция `registerClass()` возвращает значение `true` даже в экстремальных случаях, когда и видеоклип, и класс не существуют. В приведенном ниже листинге возвращаемое значение не проверяется.

Строки 2–11 определяют функцию-конструктор `myClass`. Если `arg0` является неопределенным, то функция ищет значение в объекте `this.args.zero`. Аналогичным образом, если неопределенным является `arg1`, функция ищет значение в объекте `this.args.one`. Помните, что ключевое слово `this` обозначает исходный объект внутри функции-конструктора, т.е. в данном случае `this.args.zero` означает `initObj.args.zero`.

Строка 11 делает класс `MyClass` наследником класса `MovieClip`. Строка 12 создает исходный объект со свойством `args`, которое будет использоваться для передачи значений функции `myClass`. (Имя `initObj` является произвольным.) Строка 13 присоединяет фильм библиотеки, используя `initObj` в качестве исходного объекта.

На этом функциональная часть программы завершается. Строка 14 показывает, что объект `myClip` является видеоклипом. Строка 15 демонстрирует, что объект `myClip` имеет доступ к основным свойствам видеоклипа. Строка 16 показывает, что объект `myClip` имеет доступ к наследуемым свойствам класса `MovieClip`. Строки 17 и 18 демонстрируют, что с помощью функции-конструктора `myClass` по-прежнему можно создавать видеоклипы.

```
1: Object.registerClass("clip", MyClass);
2: function MyClass (arg0, arg1) {
3:     if (arg0 == undefined)
4:         this.arg0 = this.args.zero;
```



```

5:     else this.arg0 = arg0;
6:     if (arg1 == undefined)
7:         this.arg1 = this.args.one;
8:     else this.arg1 = arg1;
9:     this.localVar = "locVar";
10: }
11: MyClass.prototype = new MovieClip();
12: initObj = {args : { zero : "z" , one : "o" } };
13: _root.attachMovie("clip","myClip", 1, initObj);
14: trace(typeof _level0.myClip);           // movieclip
15: trace(typeof _level0.myClip._x);         // number
16: trace(typeof _level0.myClip._play);      // function
17: myObj = new MyClass("argument 0", "argument 1");
18: trace(typeof _level0.myObj);             • // object

```

ПРОВЕРКА ПЕРЕКРЫТИЯ ВИДЕОКЛИПОВ

Чтобы проверить, перекрывается ли какая-то часть видеоклипа с другим видеоклипом или какой-то определенной точкой, можно воспользоваться методом `hitTest()` класса `MovieClip()`. Этот метод возвращает значение `true`, если объекты совпадают; в противном случае он возвращает значение `false`. Один или оба видеоклипа либо любой дочерний видеоклип могут обладать свойством `_visible`, установленным в значение `false`, что не влияет на исполнение метода `hitTest()`.

При наличии двух видеоклипов метод `hitTest()` проверяет, накладываются ли их *ограничительные окошки*. Ограничительное окошко — это самый маленький прямоугольник, в котором содержится вся графика видеоклипа.

Определение ограничительного окошка видеоклипа

Для определения ограничительного окошка видеоклипа воспользуйтесь методом `getBounds()` класса `MovieClip`. Ниже приведен пример:

```
boundsObject = myClip.getBounds(coordinateSpace);
```

где `boundsObject` — объект с четырьмя свойствами, `xMin`, `xMax`, `yMin` и `yMax`, обозначающими соответственно левую, правую, верхнюю и нижнюю координаты объекта `myClip` относительно точки регистрации временной шкалы с именем `coordinateSpace`. Таким образом, `boundsObject.yMin` определяет верхнюю координату видеоклипа `myClip`.

Если не указывать значение координатной плоскости, то в этом качестве будет использоваться сам видеоклип (в нашем примере — `myClip`). Если не задавать значение самого видеоклипа, то будет использоваться текущая временная шкала. Таким образом, два приведенных ниже выражения являются эквивалентными.

```
getBounds();
this.getBounds(this);
```

Если необходимо получить только одно из четырех свойств, это можно сделать следующим образом:

```
topOfMyClip = myClip.getBounds().yMin;
```

Метод `MovieClip.getBounds()` можно использовать для определения перекрытия. По сравнению с `hitTest()`, метод `getBounds()` является более гибким, поскольку он позволяет проверить отдельно каждую из четырех границ видеоклипа, а также свободно подстраивать точку тестирования. Например, вместо координаты `yMin` можно использовать `yMin-10` или `yMin+10`.

Применение метода `getBounds()` продемонстрировано в фильме `getbounds.fla`, содержащемся на прилагаемом к книге компакт-диске.

Процедура проверки перекрытия двух видеоклипов имеет формат

```
myClip1.hitTest(myClip2)
```

При проверке наложения видеоклипа и точки можно использовать либо координаты ограничительного окошка видеоклипа, либо области, содержащие графику. Для применения ограничительного окошка установите булев параметр `"shapeFlag"` в значение `false`, а для использования только графики — в значение `true`. Процедура имеет такой формат:

```
target.hitTest(x, y, shapeFlag)
```

Данный подход можно использовать для того, чтобы проверить, находится ли указатель мыши над видеоклипом:

```
target.hitTest (_xmouse, _ymouse, true)
```

Эту же методику можно использовать для того, чтобы проверить, перекрывает ли видеоклип точку регистрации другого видеоклипа.

Совет

Если при использовании данной методики возникнут проблемы, для того, чтобы проверить, перекрывает ли видеоклип точку регистрации видеоклипа, созданного с помощью метода `createEmptyMovieClip()`, обратитесь к разделу "Возможные проблемы" в конце этой главы.

Как правило, метод `hitTest` О используется при наличии оператора `if`. Кроме того, проверку на наличие конфликтов следует выполнять неоднократно. Это означает, что код проверки необходимо поместить в обработчик события. Приведенный ниже блок кода проверяет в каждом кадре, не перекрывает ли какая-то часть графики объекта `myClip` точку (100, 200) временной шкалы, в котором находится данный блок кода.

```
_root.onEnterFrame = function () {  
    if (myClip.hitTest(100, 200, true)) {  
        // выполните какие-то действия  
    }  
};
```

Используемые в данной проверке значения `x` и `y` интерпретируются как точки `x` и `y` основного рабочего поля. Если вы хотите проверить, не перекрывается ли один видеоклип какой-либо точкой другого, воспользуйтесь методом `localToGlobal()` класса `MovieClip`, чтобы преобразовать координаты точки в координаты основного рабочего поля.

МЕТОДЫ ВИДЕОКЛИПОВ `LOCALTOGLOBAL()` И `GLOBALTOLOCAL()`

Началом отсчета локальных координат видеоклипа является его точка регистрации. Началом отсчета глобальных координат является верхний левый угол основного рабочего поля. Чтобы преобразовать локальные координаты в глобальные и наоборот, сначала необходимо создать объект с двумя свойствами, `x` и `y`:

```
myObj = new Object;  
myObj.x = 100;  
myObj.y = 200;
```

Затем этот объект используется в качестве параметра метода `localToGlobal()` или `globalToLocal()` класса `MovieClip`. При использовании метода `localToGlobal()` значения `x` и `y` интерпретируются как локальные координаты, которые затем преобразовываются в глобальные. При использовании метода `globalToLocal()` значения `x` и `y` интерпретируются как глобальные координаты, которые затем преобразовываются в локальные.

Данные методы не возвращают никаких значений. Вместо этого они преобразуют фактические значения `x` и `y` объекта. Например, приведенное ниже предложение преобразует `myObj.x` и `myObj.y` из локальных координат объекта `myClip` в глобальные.

```
myClip.localToGlobal(myObj);
```

УМЕНЬШЕНИЕ "ПРОПУСКОВ" ПЕРЕКРЫВАЮЩИХСЯ ОБЛАСТЕЙ ПРИ ТЕСТИРОВАНИИ С ПОМОЩЬЮ НЕВИДИМЫХ ДОЧЕРНИХ ВИДЕОКЛИПОВ

Если проверка на перекрытие выполняется только один раз в каждом кадре, то видеоклипы, которые имеют малые размеры по сравнению с их относительной скоростью, могут легко пройти сквозь друг друга без обнаружения перекрывающихся областей. Например, предположим, что есть два видеоклипа, представляющих собой квадраты размером 10 пикселей, которые движутся навстречу друг другу со скоростью 20 пикселей в кадр. Если в кадре 3 видеоклипы находятся на расстоянии 10 пикселей друг от друга, то к кадру 4 они пройдут сквозь друг друга. При этом перекрытие обнаружено не будет.

Один из способов уменьшения числа "пропусков" заключается в создании невидимого дочернего видеоклипа (свойство `_visible` которого установлено в значение `false`) внутри одного или обоих видеоклипов, вовлеченных в проверку перекрывающихся областей. Если дочерние видеоклипы будут больше родительских, то ограничивающими окошками родительских видеоклипов будут окошки их невидимых дочерних объектов. Если родительские видеоклипы движутся с постоянной скоростью относительно друг друга, то можно подобрать такой размер дочерних видеоклипов, при котором перекрывающиеся области пропущены не будут. Если скорость родительских видеоклипов относительно друг друга является переменной, то дочерние объекты можно масштабировать в сторону увеличения при возрастании относительной скорости.

СОЗДАНИЕ ИЛЛЮЗИИ ИДЕАЛЬНОГО СОВПАДЕНИЯ

Когда невидимый дочерний видеоклип больше видимой графики, программа может обнаружить перекрытие еще до наложения видимых графических элементов. Если видеоклипы двигаются достаточно быстро, то невооруженным глазом можно не заметить, что графические элементы фактически не соприкасаются. С другой стороны, обычно существует предел того, насколько большим может быть невидимый дочерний видеоклип до того, как иллюзия столкновения графических элементов друг с другом перестанет быть убедительной.

Эту проблему можно решить, переместив видеоклипы в нужное место сразу же после обнаружения перекрытия. Такую методику иллюстрирует функция `paddle()`, взятая из файла `ecoball fla`.

Функция `paddle()` подобна методу `hitTest()` в том, что она проверяет расстояния `x` и `y` между точками регистрации двух видеоклипов. Когда расстояния достаточно малы, это считается областью совпадения. При ускорении видеоклипов размеры областей совпадения увеличиваются. Этот подход задействует меньше ресурсов процессора и менее подвержен "пропускам" по сравнению с методом `hitTest()`.

Функция `paddle()` описывает отскок мяча от биты. Строка 4 функции гарантирует, что области совпадения будут обнаружены только в случае, когда экземпляр `ecoball1` движется справа налево, т.е. когда его скорость `vx` является отрицательной (`_root.ecoball1.vx < 0`).

В строке 6 функции `paddle()` регулируется размер области совпадения. Величина `hitDx` увеличивается по мере увеличения скорости биты, `paddleVx`, и скорости мяча, `ecoball1.vx`. При этом уменьшается вероятность их прохождения сквозь друг друга.

Функция `Math.max()` в строке 2 гарантирует, что скорость объекта `paddle` (биты) никогда не будет меньше "константы совпадения" `HC`. Таким образом, `paddleVx/HC` всегда будет больше или равно 1. Поэтому в строке 6 умножение на величину `paddleVx/HC` может только увеличить (и никогда не уменьшит) величину области совпадения.

```
1: function paddle() {
2:   paddleVx = Math.max(_root.paddlei._x-oldPaddlex, HO;
3:   oldPaddlex = _root.paddlei._x;
4:   if (_root.ecoballi.vx < 0) {
5:     // hitDx увеличивается, но никогда не превышает 100
6:     hitDx=Math.min(Math.abs((root.ecoballi.vx/12)*(paddleVx/HC)),100);
7:     if ( (Math.abs(_root.paddlei._x - _root.ecoballi._x) < hitDx)
&&
8:         (Math.abs(_root.paddlei._y - _root.ecoballi._y) < hitDy) ) {
9:       // бита ударяет по мячу - создается иллюзия точного совпадения
10:      _root.ecoballi._x = _root.paddlei._x;
11:      // отскок обратно
12:      _root.ecoballi.vx = -_root.ecoballi.vx;
13:    }
14:  }
15: }
```

ВЫПОЛНЕНИЕ НЕСКОЛЬКИХ ПРОВЕРОК НА НАЛИЧИЕ СОВПАДЕНИЙ В КАЖДОМ КАДРЕ

В некоторых приложениях можно поместить код проверки в обработчик события, связанный с мышью, например в `onMouseMove`. Эти обработчики событий могут активизироваться в каждом кадре несколько раз.

Внимание!

Выполнение нескольких проверок в каждом кадре поглощает много ресурсов процессора и вызывает риск замедления работы приложения до недопустимого уровня.



Еще одной возможностью является использование для движения видеоклипов и выполнения нескольких проверок в каждом кадре глобальной функции `setInterval()`. Естественно, выполнение новой проверки будет бесполезным, если со времени последней видеоклип не был перемещен. Кроме того, для обновления экрана следует использовать функцию `updateAfterEvent()`.

Совет

Глобальная функция `setInterval()` рассмотрена в следующем подразделе.

ПОВТОРНЫЙ ВЫЗОВ ФУНКЦИИ С ПОМОЩЬЮ ГЛОБАЛЬНОЙ ФУНКЦИИ `SETINTERVAL()`



Глобальная функция `setInterval()` вызывает функцию или метод неоднократно, через определенные интервалы времени в процессе воспроизведения видеоклипа. Вам необходимо задать параметр, указывающий интервал в миллисекундах. Ниже приведены два формата этой функции.

`Set Interval (функция, интервал[, arg1, arg2, ..., argn])`

Например:

```
setInterval(myFunc, 100);  
setInterval(myObj, myMethod, 100);
```

Данная функция выглядит достаточно просто. Однако в реальности, как показывают два следующих примера, взаимосвязь между указанным и фактическим интервалом исполнения является совсем не очевидной.

- Если указанный интервал меньше длины кадра (времени, требующегося на воспроизведение одного кадра), то вызов функции или метода происходит в какой-то точке по истечении интервала. Фактический интервал часто бывает в 10 раз больше указанного. Например, частота смены кадров в 1 кадр в секунду эквивалентна продолжительности кадра в 1000 миллисекунд (1 секунда). Если при частоте смены кадров 1 кадр в секунду задать интервал в 10 миллисекунд, то фактический интервал будет равен 100 миллисекундам, что в 10 раз превышает указанный интервал.
- Кроме того, интервал может варьироваться в зависимости от того, насколько интенсивной работы процессора требует функция или метод, какие еще процессы происходят параллельно с данным и насколько мощным является компьютер.
- Если указать интервал, больший продолжительности кадра, то функция будет выполняться при переходе к следующему кадру. Например, если при частоте 1 кадр в секунду указать интервал 1500 миллисекунд (1,5 кадра), то функция будет выполняться через кадр.

ОПРЕДЕЛЕНИЕ ФАКТИЧЕСКОГО ИНТЕРВАЛА

Приведенную ниже функцию можно использовать для определения фактического интервала, образующегося на основе любого заданного. Просто измените значение `interval` в первой строке.

```
interval = 100;  
clearVal = setInterval(measureInterval, interval);  
function measureInterval () {  
    if (restart == undefined) restart = 0;  
    else restart = start;  
    start = getTimer();  
    var elapsed = start - restart;  
    trace("interval: " + elapsed);  
    updateAfterEvent();  
}
```

Обратите внимание на функцию `updateAfterEvent()` в последней строке, использующуюся для обновления экрана.

Внимание!

Функция `setInterval` O может ухудшить скорость воспроизведения фильма. Чем больше интервалов выполняется одновременно, тем меньше производительность при воспроизведении фильма.

ОСТАНОВКА и "СБРОС" ПОВТОРЯЮЩЕГОСЯ ВЫЗОВА ФУНКЦИИ

Функция `setInterval()` возвращает *идентификатор интервала*, который можно передать в метод `clearInterval()`, чтобы остановить исполнение функции. Например, в приведенном в предыдущем подразделе коде идентификатор интервала `clearVal` указан во второй строке. Таким образом, следующая строка отменит выполнение функции `measureInterval()`:

```
clearInterval(clearVal);
```

ПЕРЕТАСКИВАНИЕ ВИДЕОКЛИПОВ

При разработке графических пользовательских интерфейсов и игр очень полезной будет возможность подобрать графику на экране, переместить ее в другое место и отпустить там, где необходимо.



Во Flash 5 создать поведение перетаскивания было очень просто. Во Flash MX это сделать еще проще благодаря появлению видеоклипов кнопок — видеоклипов с обработчиками событий, присущих кнопкам, таких как `onPress` и `onRelease`.

Совет

Подробная информация о видеоклипах кнопок и обработчиках событий содержится в главе 16 "Взаимодействие, события и установление последовательности".

ПРЕДЛОЖЕНИЯ `STARTDRAG()` И `STOPDRAG()`

Во Flash 5 поведение перетаскивания реализовывалось путем помещения внутри видеоклипа невидимой кнопки и присоединения к ней двух обработчиков событий: `on (press)`, содержащего предложение `startDrag()`, и `on (release)`, содержащего предложение `stopDrag()`. Такая же базовая стратегия применяется и во Flash MX, за исключением того, что теперь обработчики событий `onPress` и `onRelease` можно присоединять непосредственно к видеоклипу.

В обоих случаях результат с точки зрения пользователя будет одинаковым: при нажатии кнопки мыши видеоклип, указанный в предложении `startDrag()` (внутри обработчика событий `onPress`), "прикрепляется" к указателю мыши и перемещается вместе с ним. Когда пользователь отпускает кнопку мыши, в обработчике событий `onRelease` выполняется предложение `stopDrag()` и любой перетаскиваемый в данный момент видеоклип перестает быть таковым, т.е. видеоклип больше не привязан к указателю мыши и при дальнейшем передвижении мыши вместе с ней не перемещается.

Обратите внимание на то, что метод `stopDrag()` применяется не только к видеоклипу, указанному в одноименном предложении. Он отключает процедуру перетаскивания для любого видеоклипа, который можно перетаскивать в текущий момент. Таким образом, он является точным эквивалентом глобальной функции `stopDrag()`. Имя видеоклипа подобно комментарию, напоминающему о том, какой видеоклип в данный момент должен быть доступным для перетаскивания.

Метод `stopDrag()` не принимает никаких параметров. Например, ниже приведена часть кода, соответствующая блоку `stopDrag()` типовой реализации процедуры перетаскивания.

```
myClip.onRelease = function () {  
    myClip.stopDrag();  
};
```

Ниже приведен блок кода `startDrag()` типовой реализации процедуры перетаскивания.

```
myClip.onPress = function () {  
    myClip.startDrag();  
};
```

Дополнительные параметры метода `startDrag()` рассматриваются в следующем подразделе.

ПАРАМЕТРЫ МЕТОДА `STARTDRAG()`

Метод `startDrag()` имеет ряд вспомогательных параметров, предназначенных для выполнения двух следующих задач.

- Привязка точки регистрации видеоклипа к указателю мыши. Это происходит, если вспомогательный параметр `lock` установлен в значение `true`. Если он установлен в значение `false`, то видеоклип и указатель мыши остаются в том пространственном отношении, в котором они пребывали, когда пользователь первоначально нажал кнопку мыши. Например, в приведенном ниже предложении точка регистрации видеоклипа `myClip` привязывается к указателю мыши.

```
myClip.startDrag(true);
```

- Если аргумент `lock` не указан, он трактуется как установленный в значение `false`.
- Ограничение области, в пределах которой можно перетаскивать видеоклип. Эта область задается четырьмя параметрами: `left`, `top`, `right` и `bottom`. Например, приведенное ниже предложение ограничивает область перетаскивания видеоклипа `myClip` прямоугольником, начинающимся в левом верхнем углу рабочего поля и простирающимся на 100 пикселей вправо и 200 пикселей вниз.

```
myClip.startDrag(true, 0, 0, 100, 200]]
```

Интересный пример реализации перетаскивания приведен в файле `spacelisten fla` от Flash-разработчика Энди Холла (Andy Hall), содержащемся на прилагаемом к книге компакт-диске. В строке 1 создается пустой видеоклип, а в строках 2-8 с помощью методов рисования ему ставится в соответствие графический символ квадрата. Видеоклипу ставится в соответствие функциональность кнопки путем присоединения обработчика событий `onRelease`, а возможность завершения процедуры перетаскивания видеоклипа реализуется за счет помещения в этот обработчик событий предложения `stopDrag()` (строки 9–11).

Для плавной реализации перетаскивания теперь следовало бы определить обработчик события `onPress` и поместить в него предложение `startDrag()`, как показано в предыдущем подразделе. Однако вместо этого в файле `spacelisten fla` отключается отображение курсора в виде руки (строка 12) и "скрывается" предложение `startDrag()`, находящееся внутри обработчика события `onKeyDown` (строки 14–21), который предназначен для того, чтобы "наблюдать" за нажатием пользователя клавиши <Пробел>. И только когда пользователь нажмет эту клавишу, будет отображен курсор в виде руки (строка 16) и определен обработчик события `onPress`, содержащий предложение `startDrag()`.

В данном коде также содержится обработчик события `onKeyDown`, который, когда пользователь отпустит клавишу, повторно отключает отображение курсора в виде руки (строка 24), а также обработчик события `onPress`, устанавливая его в значение `null` (строка 25).

В результате перетаскивание можно выполнить только в том случае, если нажата клавиша <Пробел>.

```
1: _root.createEmptyMovieClip("a",1);
2: a.moveTo(0,0);
3: a.lineStyle(2,0x000000);
4: a.beginFill(0x000000,30);
5: a.lineTo(0,100);
6: a.lineTo(100,100);
7: a.lineTo(100,0);
8: a.endFill();
9: a.onRelease = a.onReleaseOutside = function() {
10:     this.stopDrag();
11: }
12: a.useHandCursor = false;
13: spaceListen = new Object();
14: spaceListen.onKeyDown = function () {
15:     if (Key.getCode() == Key.SPACE) {
```

```

11: }
12: a.useHandCursor = false;
13: spaceListen = new Object();
14: spaceListen.onKeyDown = function () {
15:     if (Key.getCode() == Key.SPACE) {
16:         a.useHandCursor = true;
17:         a.onPress = function () {
18:             this.startDrag();
19:         }
20:     }
21: }
22: spaceListen.onKeyUp = function () {
23:     if (Key.getCode() == Key.SPACE) {
24:         a.useHandCursor = false;
25:         a.onPress = null;
26:     }
27: }
28: Key.addListener(spaceListen);

```

СВОЙСТВО ВИДЕОКЛИПА _DROPTARGET

Когда пользователь перетаскивает какой-либо объект, очень важно, *где* он его отпускает. Перетаскивание файла в корзину **инициирует** совершенно другие операции в программе по сравнению с перетаскиванием файла в папку. Предназначенное только для чтения свойство видеоклипа `_droptarget` позволяет легко проверить, где пользователь отпустил или должен отпустить перетаскиваемый видеоклип.

Если точка регистрации перетаскиваемого видеоклипа располагается поверх любой части другого видеоклипа (независимо от того, была ли отпущена кнопка мыши), то свойство `_droptarget` содержит путь к этому видеоклипу. Если перетаскиваемый видеоклип не расположен над другим видеоклипом, то свойство `_droptarget` будет установлено в значение `undefined`.

Основы свойства `_droptarget` заложены еще во Flash 4. Возвращаемый им путь к видеоклипу указывается в виде строки с использованием принятого во Flash 4 "представления с использованием символов косой черты", как показано в следующих примерах:

```

/myClip
/myClip/child

```

ДИНАМИЧЕСКИЕ МАСКИ

Можно вручную создать слой маски, который будет отображать нижние слои, располагающиеся под графикой слоя маски. Если слои маски не содержат графику, то содержимое расположенных под ним слоев будет скрыто.

Совет

Маски рассматривались в главе 3 "Рисование во Flash".



Во Flash 5 маску можно было задать только с помощью графического пользовательского интерфейса, но не **ActionScript**. Кроме того, посредством **ActionScript** невозможно было управлять видеоклипом, находящимся в слое маски. Если вы хотели анимировать маску, необходимо было создать промежуточные отображения формы или движения. С **ActionScript** можно было работать в видеоклипах, содержащихся в *маскированных* слоях, и в видеоклипах, содержащих внутренние маски. Все эти функции поддерживаются и во Flash MX.

Новым во Flash MX является метод `setMask()` класса `MovieClip`, позволяющий программным образом назначить один видеоклип маской другого. Например, в приведенном ниже предложении `myMaskedClip` является именем экземпляра видеоклипа, подлежащего маскированию, а `myMask` — именем экземпляра видеоклипа, являющегося маской.

```
myMaskedClip.setMask(myMask);
```

Маска и маскируемый объект образуют единую "моногоамную" пару: можно иметь только одну маску на каждый маскируемый объект и только один маскируемый объект на каждую маску. При задании новой маски для видеоклипа все предыдущие маски в этом качестве функционировать не будут. Как правило, это ограничение не строгое, поскольку и маска, и маскируемые видеоклипы могут содержать несколько кадров, несколько слоев и несколько дочерних видеоклипов, работающих по сценарию.

Если вы хотите отключить маскирование определенного видеоклипа без назначения новой маски, то в качестве параметра видеоклипа-маски задайте значение `null`:

```
myMaskedClip.setMask(null); // myMaskedClip больше не является маскированным
```

Вот что нужно помнить при динамическом маскировании.

- Маски всегда невидимы, *пока они являются масками*. Если видеоклип-маска перестает использоваться в этом качестве, он становится видимым, а это не всегда нужно пользователю. Следовательно, перед отключением маску следует сделать невидимой, как показано ниже:

```
myMaskedClip.setMask (myMask);  
myMask.visible = false;  
myMaskedClip.setMask (null);
```

- Для создания масок можно использовать прикладной программный интерфейс рисования. Однако, так же, как не дают никакого эффекта штрихи (линии) в масках, созданных вручную, являются совершенно бесполезными и линии, созданные в масках с помощью прикладного программного интерфейса рисования. Масками могут служить только те части видеоклипа, которые имеют заливку.

Совет

Прикладной программный интерфейс рисования рассматривается в главе 18 "Процедура рисования с помощью ActionScript".

- Системные шрифты в маскированных видеоклипах отображаются, но не маскируются. Это означает, что они будут видны, даже будучи расположенными под маской! Чтобы обойти эту проблему, необходимо внедрить шрифты следующим образом:

```
myTextField.embedFonts = true;
```

- Для компонентов пользовательского интерфейса Flash свойство `embedFonts` можно задать глобально:

```
globalStyleFormat.embedFonts = true;
```

- Задание видеоклипа для самой маски не играет никакой роли, как в этом примере:

```
mc.setMask(mc); // ничего не происходит
```

Видеокадры и аудио: метод `attachAudio()`

Недокументированный метод `attachAudio()` класса `MovieClip` позволяет присоединять к видеоклипу потоковое аудио. По своей функциональности он аналогичен методу `attachVideo()` класса `video`. (Класс `MovieClip` имеет также недокументированный метод `attachVideo()`, но я не смог получить о нем никакой информации.)

С помощью Flash MX Flash-плеер позволяет сделать локальное потоковое видео и аудио доступным для локального просмотра и прослушивания. Эту возможность можно использовать, например, если вы хотите посмотреть и послушать себя при подготовке публичного выступления. При присоединении локального потокового видео и/или аудио Flash-плеер не публикует и не записывает видео и аудио; он просто позволяет посмотреть и послушать то, что происходит в текущий момент.

Приведенный ниже пример демонстрирует присоединение потокового аудио из локального микрофона.

```
createEmptyMovieClip ("aud", 1);  
inputMic = Microphone.get(); // захват потокового аудио  
inputMic.setGain(2); // задание громкости  
aud.attachAudio(inputMic); // присоединение потокового аудио к видеоклипу
```

Возможность присоединения потокового аудио и видео становится более интересной при работе с приложением *Flash Communication Server*, которое позволяет Flash-плееру публиковать видео и аудио для удаленной аудитории по сети. Таким образом, к примеру, могут реализоваться видеоконференции. Кроме того, Flash-плеер может записывать видео и аудио для последующего воспроизведения.

Совет

Подробная информация о функции `Microphone.get()` и о классе `Video` представлена в главе 20 "Использование встроенных объектов фильмов".

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему я не могу получить доступ к данным в загруженном SWF-файле?

Вы должны уметь присоединить фильм из библиотеки загруженного SWF-файла и получить доступ к переменным, видеоклипам и функциям, определенным в SWF-файле.

Если вы не можете этого сделать, то, вероятно, некорректно ссылаетесь на соответствующие элементы. Для просмотра путей и идентификаторов, связанных с теми или иными элементами, выполните следующие действия.

- Выполните команду **Control⇨Debug Movie** (Управление⇨Отладка фильма). Для перечисления переменных в открывшемся окне Debugger (Отладчик) выполните команду **Debug⇨List Variables** (Отладка⇨Перечислить переменные) либо нажмите клавиши <Cmd+Option+V> (Macintosh) или <Ctrl+Alt+V> (Windows).
- Для перечисления объектов в окне Debugger выполните команду **Debug⇨List Objects** (Отладка⇨Перечислить объекты) либо нажмите клавиши <Cmd+L> (Macintosh) или <Ctrl+L> (Windows).

Если вы обращаетесь к элементам правильно, то, вероятно, пытаетесь получить к ним доступ до того, как SWF-файл был полностью загружен. Чтобы убедиться в том, что файл загружен полностью, выполните следующие действия.

- Установите соответствующую задержку. Например, код, предоставляющий доступ к элементам, можно поместить через несколько кадров после кода, загружающего SWF-файл. Однако это предполагает необходимость предусмотреть время загрузки.
- Используйте событие `data` функции `loadMovie` О в комбинации либо с методами `getBytesLoaded()` и `getBytesTotal()` класса `MovieClip`, либо со свойствами видеоклипов `_framesloaded` и `_totalframes`, либо с исключенной функцией `ifFrameLoaded()`.

Я использую метод `hitTest()`, чтобы проверить, накладывается ли видеоклип на точку регистрации фильма, созданного с помощью метода `createEmptyMovieClip()`. Почему проверка не выполняется?

Когда видеоклипы создаются вручную, то вполне естественно создавать графику вокруг точки регистрации и затем позиционировать видеоклип в рабочем поле. При этом вы привыкаете, что точка регистрации располагается в центре графического изображения.

При использовании метода `createEmptyMovieClip()` для создания графики используется прикладной программный интерфейс рисования, и графика может быть создана и размещена без изменения точки регистрации. Например, с помощью функции `makeTriangle()`, приведенной в файле `dynamicButtonMovie.fla` главы 16 "Взаимодействие, события и установление последовательности", можно создать и разместить треугольник, написав всего одну строку кода, без изменения точки регистрации. Если изменять точку регистрации, то для достижения того же самого визуального эффекта понадобятся уже три строки кода. Например, в приведенном ниже листинге результат будет выглядеть одинаково как при исполнении строк 12–14, так и при исполнении только строки 15.

```

1: function makeTriangle(x1, y1, x2, y2, x3, y3, zdepth) {
2:   this.createEmptyMovieClip ("triangle", zdepth);
3:   with (triangle){
4:     beginFill (0x0000FF, 50);
5:     moveTo (x1, y1);
6:     lineTo (x2, y2);
7:     lineTo (x3, y3);
8:     lineTo (x1, y1);
9:     endFill();
10:  }
11: }
12: /* makeTriangle(0, 0, 60, 130, -50, 1);
13: triangle._x = 200;
14: triangle._y = 200; */
15: makeTriangle(200, 200, 260, 330, 250, 1);

```

При использовании более эффективного синтаксиса легко забыть о том, что точка регистрации не находится около графики. Тогда при выполнении каких-либо операций, для которых важна точка регистрации, можно получить удивительные результаты. Например, приведенная ниже проверка сообщит о наложении видеоклипа `myClip` на точку регистрации треугольника. Однако в это время видеоклип `myClip` не будет находиться рядом с графическим изображением треугольника:

```

makeTriangle(200, 200, 260, 330, 250, 1);
myClip.hitTest(triangle._x, triangle._y);

```

FLASH ЗА РАБОТОЙ: ИГРА "СЧАСТЛИВЫЙ СЛУЧАЙ"

В содержащемся на компакт-диске файле `scratch.fla` от разработчика Питера Холла (Peter Hall) для создания игры "счастливый случай" используются прикладной программный интерфейс рисования и динамическое маскирование.

В файле `scratch.fla` за четырьмя закрытыми окнами случайным образом скрыты некоторые символы: яблоко, колокольчик, лимон и звезда. Игрок стирает покрытие и открывает окна в надежде найти совпадение. Стирая покрытие, игрок фактически создает маску с помощью прикладного программного интерфейса рисования. Когда игрок заполняет маску, под

ней появляется символ, поскольку части маски, содержащие графику, являются частями, отображающими расположенное под ними содержимое.

Видеоклип, содержащий символы, называется `symbols`. Именем экземпляра видеоклипа-маски является `wax` (покрытие). Каждое стирание создает дочерний видеоклип внутри `wax` с именами `wax.1`, `wax.2` и т.д.

Приведенная ниже строка задает видеоклип `wax` в качестве маски видеоклипа `symbols`.
`symbols.setMask(wax);`

Следующая строка создает видеоклип `wax`:

```
createEmptyMovieClip("wax", 1);
```

Методы `startScratch()` и `endScratch()` определены в прототипе класса `MovieClip`, поэтому доступ к ним имеют все видеоклипы. Эти методы вызывают в обработчиках событий `wax.onMouseDown` и `wax.onMouseUp`. Таким образом, обработчик событий `wax.onMouseDown` (нажатие кнопки мыши) начинает процесс стирания, а `wax.onMouseUp` (отпускание кнопки мыши) его останавливает.

Таким образом, основу программы `scratch.fla` составляют следующие элементы.

- Два метода прототипа класса `MovieClip`: `startScratch()` и `endScratch()`. Функция `doScratch()` является вспомогательной для метода `startScratch()`.
- Два обработчика событий видеоклипа `wax`: `wax.onMouseDown` и `wax.onMouseUp`.

Имеется также функция `startAgain()`, позволяющая начать игру сначала.

Ниже приведена функциональная схема программы.

- `MovieClip.prototype.startScratch = function(width)`

Создает пустой видеоклип (такой как `wax.1`, `wax.2`, `wax.3`) и рисует один "стирающий штрих" с помощью функции `doScratch()`.

- `doScratch = function()`

Использует прикладной программный интерфейс рисования для рисования в текущем видеоклипе маски (таком как `wax.1`). Рисование выполняется заливкой, поскольку линии в качестве масок не используются.

- `MovieClip.prototype.endScratch = function()`

Удаляет свойства в видеоклипе маски (таком как `wax.1`). Свойства были заданы в функции `startScratch()`. К удаляемым свойствам относится и обработчик события `onEnterFrame`, созданный в функции `startScratch()`.

- `wax.onMouseDown = function() {this.startScratch(6)};`

Начинает новую операцию стирания при каждом нажатии кнопки мыши.

- `wax.onMouseUp = functionO {this [this.i] .endScratch()};`

Завершает операцию стирания при отпускании кнопки мыши.

- `startAgain = functionO`

Вытирает все рисунки из маски (`wax`).

На заметку

По причинам, которые я не смог определить, файл `scratch.fla` не работает в версии плеера Flash 6 для Macintosh.

ПРОЦЕДУРА РИСОВАНИЯ с ПОМОЩЬЮ ACTIONSCRIPT

В ЭТОЙ ГЛАВЕ...

Знакомство с методиками рисования во Flash MX	395
Использование объектов для отслеживания операций рисования	398
Рисование линий и кривых	399
Оптимизация функций рисования кривых	405
Использование сплошной заливки	406
Работа с градиентной заливкой	407
Возможные проблемы	414
Flash за работой: рисование круга, состоящего из восьми сегментов	415

ЗНАКОМСТВО с МЕТОДИКАМИ РИСОВАНИЯ во FLASH MX



Программное управление процедурами рисования является новой функцией Macromedia Flash MX, реализуемой посредством методов рисования класса `MovieClip`. Эти методы, позволяющие рисовать с помощью `ActionScript`, имеют очень простой формат. Они доставят вам огромное удовольствие. (Попробуйте поработать с содержащимися на компакт-диске файлами `play.pla`, `3Dcube.fla` от разработчика Кита Питерса (Keith Peters), `test5.fla`, а также `api_flower.fla` и `api_cube.fla` от разработчика Милли Маруани (Millie Maruani).) С помощью методов рисования можно выполнять такие задачи, которые ранее требовали больших усилий.

К примеру, во Flash 5 можно было создать программу для рисования, но во Flash MX сделать это гораздо проще. Так, в содержащемся на компакт-диске файле `mouseDraw.fla` основная функция рисования реализована всего лишь пятью строками кода!

С другой стороны, не всегда легко добиться того, чтобы с помощью методов рисования точно выполнялись необходимые операции. В частности, чтобы с высокой точностью управлять нарисованной кривой, потребуется либо знание тригонометрии, либо использование "метода проб и ошибок".

В целом с помощью методов рисования можно выполнить следующее:

- нарисовать линии и кривые;
- заполнить замкнутые области.

На заметку

Все методы рисования называются *прикладным программным интерфейсом рисования*.

Сохранение рисунков

Во Flash не существует средств создания программным образом графических файлов (таких как GIF, PNG или BMP). Однако связанные с рисунком данные можно легко сохранить с помощью *локального объекта коллективного использования*. После этого данные можно перенести обратно во Flash-приложение и перерисовать графическое изображение, что продемонстрировано в содержащемся на компакт-диске файле *Drawing.fla* от разработчика Питера Холла. Этот файл, реализованный менее чем в 75 строках кода, представляет собой простую программу для рисования, которая может сохранять и извлекать данные рисунков.

Совет

Подробная информация об объектах коллективного использования представлена в главе 26 "Локальная связь".

В табл. 18.1 перечислены восемь методов прикладного программного интерфейса рисования, их форматы и предназначение.

ТАБЛИЦА 18.1. МЕТОДЫ ПРИКЛАДНОГО ПРОГРАММНОГО ИНТЕРФЕЙСА РИСОВАНИЯ

Метод	ПРИМЕРЫ	ОПИСАНИЕ
beginFill	<code>myClip.beginFill();</code>	Если есть разомкнутая траектория, замыкает ее с помощью линии и применяет заливку к замкнутой области. Заливка последующих замкнутых траекторий не выполняется
	<code>myClip.beginFill(0x00FF00);</code>	Если есть разомкнутая траектория, замыкает ее с помощью линии и применяет заливку к замкнутой области. Заливка последующих замкнутых траекторий выполняется сплошным зеленым цветом
	<code>myClip.beginFill(0xFF0000, 80);</code>	Если есть разомкнутая траектория, замыкает ее с помощью линии и применяет заливку к замкнутой области. Заливка последующих замкнутых траекторий выполняется красным цветом с прозрачностью 80%
beginGradientFill	<code>myClip.beginGradientFill()</code>	Если есть разомкнутая траектория, замыкает ее с помощью линии и применяет заливку к замкнутой области. Заливка последующих замкнутых траекторий не выполняется
	<code>myClip.beginGradientFill(undefined,</code>	Если есть разомкнутая траектория, замыкает ее с помощью линии и применяет заливку к

МЕТОД	ПРИМЕРЫ	ОПИСАНИЕ
	<pre>colorArray, alphaArray, ratioArray, transformationMatrix); myClip.beginGradientFill ("linear", colorArray, alphaArray, ratioArray, transformationMatrix);</pre>	<p>замкнутой области. Заливка последующих замкнутых траекторий не выполняется</p> <p>Если есть разомкнутая траектория, замыкает ее с помощью линии и применяет заливку к замкнутой области. Выполняет линейную градиентную заливку последующих замкнутых траекторий, переходя от цвета к цвету, указанных в массиве (color-Array), с уровнями прозрачности, приведенными в массиве (alphaArray), коэффициентами, указанными в массиве (ratioArray), и преобразуемыми в соответствии с матрицей преобразования transformationMatrix</p>
clear	<pre>myClip.clear();</pre>	Удаляет из клипа все созданные программным образом рисунки и отменяет текущий стиль линии и текущую заливку
curveTo	<pre>myClip.curveTo (controlX, controlY, anchorX, anchorY);</pre>	Рисует кривую, используя текущее положение пера, координаты анкерных (anchorX , anchorY) и контрольных точек (controlX , controlY)
endFill	<pre>myClip.endFill();</pre>	Завершает и защищает от последующих изменений заливку, созданную с момента предыдущего исполнения предложения beginFill() или beginGradientFill() . Не выполняет заливку замкнутых областей до тех пор, пока не встретится другое предложение beginFill() ИЛИ beginGradientFill()
lineStyle	<pre>myClip.lineStyle(); myClip.lineStyle(6); myClip.lineStyle(6, 0xFF0000); myClip.lineStyle(6, 0xFF0000, 50);</pre>	<p>Прекращает рисование линий</p> <p>Рисует сплошные черные линии толщиной 6 пикселей</p> <p>Рисует сплошные красные линии толщиной 6 пикселей</p> <p>Рисует сплошные красные линии толщиной 6 пикселей и прозрачностью 50%</p>
lineTo	<pre>myClip.lineTo (100, 50)</pre>	Рисует линию текущего стиля. Начинает рисование с текущего положения пера и заканчивает в точке, расположенной на расстоянии 100 пикселей вправо и 50 пикселей вниз относительно точки регистрации клипа myClip
moveTo	<pre>myClip.moveTo (200, 250)</pre>	Устанавливает текущее положение пера в точке, расположенной на расстоянии 200 пикселей вправо и 250 пикселей вниз относительно точки регистрации клипа myClip

Часть прикладного программного интерфейса рисования, касающаяся рисования линий, в основном имитирует инструмент Реп (Перо). Однако (если знать, как это сделать) можно дублировать все операции, которые выполняются с помощью обычных инструментов рисования Flash: рисовать круги, прямоугольники, треугольники, прямые или волнистые линии, выполнять сплошную и градиентную заливку и т.п.

Все функции рисования являются методами класса **MovieClip**. Это имеет смысл, поскольку во Flash видеоклипы являются контейнерами графики. Все операции прикладного программного интерфейса рисования относятся к одному видеоклипу. Например, если вы указываете стиль линии, он применяется только к графике, программным образом созданной внутри одного клипа.

ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ для ОТСЛЕЖИВАНИЯ ОПЕРАЦИЙ РИСОВАНИЯ

Для того чтобы сгруппировать создаваемые графические изображения и присвоить им имена, несущие смысловую нагрузку, можно использовать объекты. Систематизированные имена гораздо более понятны, чем просто числа.

Предположим, к примеру, что вы рисуете окружность. Необходимо указать ее центральную точку (координаты *x* и *y*) и радиус.

Для группирования и присвоения имен данным можно использовать классы **Point** и **Circle**. Как всегда во Flash, классы реализуются в виде функций, а новые члены класса создаются с помощью оператора **new**.

Совет

Знакомство с классами и оператором **new** состоялось в главе 15 "Объединение предложений в функции".

Для создания центральной точки воспользуйтесь классом **Point**:

```
function Point (x , y) {  
    this.x = x;  
    this.y = y;  
};  
myCenter = new Point(100, 100);
```

Ниже показано, как координаты *x* и *y* этой точки назначаются параметрами метода **moveTo()**;

```
moveTo(myCenter.x, myCenter.y);
```

Эта строка кода гораздо понятнее, чем строка

```
moveTo (100, 100) ;
```

С помощью класса **Circle** центральная точка создается как свойство окружности. Для определения **окружности** необходимо задать еще одно свойство — радиус:

```
function Circle (centerX, centerY, radius) {  
    this.center = new Point(centerX, centerY);  
    this.radius = radius;  
};
```

РИСОВАНИЕ линий и КРИВЫХ

"Рабочими лошадками" прикладного программного интерфейса рисования является метод `lineTo()`, посредством которого рисуются прямые линии, и метод `curveTo()`, с помощью которого рисуются кривые. К обоим методам применимы следующие положения.

- Графические изображения, создаваемые с помощью прикладного программного интерфейса рисования, располагаются под графическими изображениями, созданными в том же видеоклипе вручную.
- Методы рисования предполагают наличие начальной точки (текущее положение рисования или текущее перо).
- Если какие-то параметры пропущены, метод не выполняется и текущее положение рисования не изменяется.
- Точка назначается путем задания координат *x* и *y*. Эта точка расположена на расстоянии *x* и *y* относительно точки регистрации видеоклипа, в котором выполняется рисование. В основной временной шкале точкой регистрации является левый верхний угол, в других случаях ею по умолчанию является середина видеоклипа.

РИСОВАНИЕ линий

Чтобы нарисовать линию, необходимо выполнить две операции:

- указать стиль линии;
- сообщить Flash координаты начальной и конечной точки.

Flash рисует линию, начиная с указанной начальной точки и до конечной точки. Затем конечная точка становится текущим положением пера, которое используется для рисования другой линии без обращения к методу `moveTo()`.

УКАЗАНИЕ Стиля линии

Перед тем как рисовать линию, необходимо указать ее стиль. Для этого используется метод `lineStyle()`. Стиль будет поддерживаться до тех пор, пока вы не укажете новый. Стили линии можно изменять в пределах траектории, после завершения одной линии и перед началом следующей.

На заметку

Непрерывный ряд линий или кривых, образованных "без отрыва пера от бумаги", называется *траекторией*.

Метод `lineStyle()` имеет три опциональных параметра, определяющих толщину, цвет и прозрачность линии.

- Толщина измеряется в пикселях. Значение 0 означает волосную линию.
- Цвет указывается в виде числа; наиболее удобочитаемым является шестнадцатеричное представление.
- Прозрачность указывается в виде числа в диапазоне от 0 (невидимый) до 100 (сплошной). Если этот параметр опущен или указано число больше 100, Flash использует значение 100. Если указано отрицательное число, Flash использует значение 0.

Совет

Подробная информация об использовании объекта `Color` и о представлении цветов в виде шестнадцатеричных значений приведена в главе 20 "Использование встроенных объектов фильмов".

Цвета, назначаемые для видеоклипа с помощью объекта **Color**, замещают цвета, назначенные с помощью прикладного программного интерфейса рисования.

Несмотря на то что все три параметра не являются обязательными, для наличия следующего необходимо наличие предыдущего. Нельзя указать цвет, если не указана толщина, и нельзя указать прозрачность, если не указан цвет.

Метод `lineStyle()` имеет такой формат:

```
myClip.lineStyle(толщина, цвет, прозрачность);
```

Если опустить все параметры, то Flash прекратит рисовать линии в видеоклипе.

Примеры метода `lineStyle()` см. выше, в табл. 18.1.

НАЗНАЧЕНИЕ НАЧАЛЬНОЙ И КОНЕЧНОЙ ТОЧЕК

Если только что была нарисована линия, Flash предполагает, что ее конечная точка будет начальной для следующей линии. Flash никогда "не отрывает пера от бумаги", если не приведено никаких специальных распоряжений сделать это.

Если линия еще не нарисована или если вы не хотите, чтобы конечная точка последней линии была начальной для следующей, необходимо сообщить Flash, где следует начать рисовать линию. Для этого используется метод `moveTo()`, задающий текущее положение пера. Эта функция одновременно выполняет команды "начать здесь" и "оторвать перо от бумаги и перенести его сюда".

Совет

Пример использования метода `moveTo()` при изменении заливки приведен ниже, в разделе "Использование сплошной заливки".

Пример применения метода `moveTo()` см. выше, в табл. 18.1.

При вызове метода рисования без обращения к методу `moveTo()` начальной точкой по умолчанию будет точка регистрации (0, 0).

Координаты конечной точки задаются с помощью метода `lineTo()`. Пример его применения см. в табл. 18.1.

НЕПРЕРЫВНОЕ РИСОВАНИЕ с помощью мыши

Основные функции рисования легко реализуются с помощью прикладного программного интерфейса рисования. Рассмотрим, к примеру, файл `mouseDraw fla`. Написав несколько строк кода, мы заставляем мышь непрерывно рисовать черную линию толщиной 6 пикселей. Ниже приведен такой код.

```
lineStyle(6);
moveTo(_xmouse, _ymouse);
onMouseMove = function() {
    lineTo(_xmouse, _ymouse);
    updateAfterEvent();
}
```

Наличие предложения `updateAfterEvent()` не является абсолютной необходимостью, но оно делает линию более плавной.

РИСОВАНИЕ КВАДРАТА

Методы рисования линий проиллюстрируем на примере создания квадрата. Ниже приведен код.

```
1: function Point (x, y) {
2:   this.x = X;
3:   this.y = y;
4: };
```

```

5: function Square (centerX, centerY, side) {
6:   this.center = new Point(centerX, centerY);
7:   this.side = side;
8:   top = this.center.y + (side/2);
9:   this.bottom = this.center.y - (side/2);
10:  this.left = this.center.x - (side/2);
11:  this.right = this.center.x + (side/2);
12: };
13: createEmptyMovieClip("mySquareClip",1);
14: mySquareClip.mySquare = new Square(300,300,100);
15: with (mySquareClip) {
16:   with (mySquare) {
17:     lineStyle(6);
18:     moveTo(left, top);
19:     lineTo(left, bottom);
20:     lineTo(right, bottom);
21:     lineTo(right, top);
22:     lineTo(left, top);
23:   }
24: }

```

Теперь построчно рассмотрим, какие операции выполняет программа.

- Создаются класс `Point` (строки 1–4) и класс `Square` (строки 5–12). Класс `Square` принимает три параметра. Первые два указывают координаты `x` и `y` центра квадрата, третий указывает длину стороны квадрата. Исходя из этих значений, функция `Square` вычисляет значения `top`, `bottom`, `left` и `right`, каждое из которых составляет половину длины стороны, начиная от центра квадрата.
- Создается видеоклип, в котором выполняется рисование (строка 13).
- Создается член класса `Square` с именем `mySquare` (строка 14).
- Указывается стиль линии (строка 17). Для задания толщины линии используется обычное число. Для сохранения значений толщины, цвета и прозрачности линии можно создать свойство класса `Square`, подобно тому как для сохранения значений `x` и `y` центра квадрата используется свойство `center`. Например, можно определить класс `Lstyle` следующим образом:

```

function Lstyle (lineThickness, lineColor, lineTransparency) {
    this.lineThickness = lineThickness;
    this.lineColor = lineColor;
    this.lineTransparency = lineTransparency;
};

```

Затем к классу `Square` в качестве параметров можно добавить свойства `lineThickness`, `lineColor` и `lineTransparency` и создать член класса `Lstyle` внутри класса `Square`, аналогично примеру со свойством `center`, являющимся членом класса `Point`. Например, приведенная ниже строка внутри функции `Square` будет создавать свойство `line` в каждом объекте, образованном классом `Square`.

```
this.line = new Lstyle(lineThickness, lineColor, lineTransparency)
```

Тогда предложение `new` в строке 14 будет выглядеть следующим образом:

```
mySquareClip.mySquare = new Square(300,300,100,6,null,null);
```

А строка 17 будет такой:

```
lineStyle(line.lineThickness);
```

Однако существующее предложение `lineStyle (6)`; короче и легче для восприятия.

- Назначается начальная точка (строка 18).
- Рисуется квадрат (строки 17-22). Чтобы избежать повторения имени видеоклипа (`mySquareClip`) и имени объекта квадрата (`mySquare`), программа использует вложенные операторы `with`. Когда необходимо описать переменную или выполнить метод, Flash сначала выполнит поиск свойства или метода с этим именем в объекте `mySquare`, а затем в объекте `mySquareClip`.

Совет

Подробная информация об операторе `with` приведена в главе 14 "Работа с данными: использование предложений".

Обратите внимание на то, насколько удобочитаемой является часть кода, выполняющая рисование квадрата. Вы можете отчетливо представить себе, что рисование начинается с левой верхней точки, потом рисуется линия к нижней левой точке, затем — линия к правой нижней точке и так до тех пор, пока квадрат не будет завершен. В этом и заключается преимущество реализации функций рисования в рамках объектно-ориентированного программирования.

РИСОВАНИЕ КРИВЫХ

Кривые рисуют с помощью метода `curveTo()`. Вначале бывает тяжело понять, каким образом ведет себя эта функция. Даже после того, как вы это поймете, создавать сложные контуры и формы с математической точностью будет непросто. Тем не менее, когда вы осознаете, как работает метод `curveTo()`, и немного поэкспериментируете, можно будет добиться восторжительных результатов.

К счастью, основные подготовительные операции за вас уже сделаны. Рассмотрим содержащийся на компакт-диске файл `drawMethods fla` от разработчика Рика Юинга (Ric Ewing). В нем содержатся функции, создающие как основные контуры, например дуги (до полных окружностей), овалы, квадраты, прямоугольники и правильные многоугольники с любым количеством сторон, так и более сложные контуры, например звезды, "вспышки" и круги с вырезанными сегментами. Эти функции можно модифицировать для выполнения самых сложных, невероятных и даже полезных задач.

Глядя на код, вы обнаружите, что все кривые фигуры требуют использования тригонометрических функций. Эта глава ограничивается лишь рассмотрением двух примеров, в которых требуется применение тригонометрии. Если вы хотите лучше разобраться с тригонометрией, то в некоторых примерах главы 19 "Использование встроенных базовых объектов" этот предмет рассматривается более подробно.

Совет

Подробная информация о тригонометрических функциях приведена в главе 19 "Использование встроенных базовых объектов".

ЗНАКОМСТВО с МЕТОДОМ `CURVETO()`

Если вы еще этого не сделали, потратьте немного времени и поработайте с инструментом Реп (Перо). В целом метод `curveTo()` дублирует функциональные возможности этого инструмента.

Совет

Инструмент Реп описан в главе 2 "Интерфейс Flash", а приемы работы с ним рассматриваются в главе 3 "Рисование во Flash".

При работе и с инструментом Реп, и с методом `curveTo()` используются кривые Безье, которые определяют кривые на основе *анкерных* и *контрольных точек*. Две анкерные точки

определяют концы каждой кривой, а контрольные точки (на **концах маркеров**) действуют подобно магнитам, вытягивая кривую в направлении контрольной точки.

На заметку

Если вы хотите подробнее ознакомиться с кривыми Безье, обратитесь к узлу <http://www.macromedia.com> и найдите ссылку, ведущую к информации об инструменте Pen во Flash.

На рис. 18.1 изображены два типа кривых Безье: второго и третьего порядка. Кривые Безье второго порядка имеют одну контрольную точку, а третьего порядка — две контрольные точки. При редактировании с помощью инструмента **Pen** используются кривые Безье третьего порядка, но затем они преобразовываются в кривые второго порядка и сохраняются в SWF-файле, поскольку таким образом занимают меньше места. В методе `curveTo()` с самого начала используются кривые Безье второго порядка.

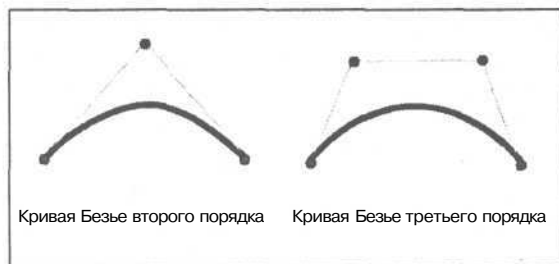


Рис. 18.1. Кривые Безье второго и третьего порядка

Поскольку в инструменте Реп для редактирования используются кривые Безье третьего порядка, то знакомство с этим инструментом, будучи достаточно неплохой стартовой площадкой, все же будет недостаточным для использования кривых Безье второго порядка в методе `curveTo()`. Попробуйте поработать с содержащимся на компакт-диске файлом `curveTo fla` от разработчика Энди Холла (Andy Holl). Он даст наглядное представление о том, как в кривых Безье второго порядка две анкерные точки и контрольная точка используются для определения кривой.

АППРОКСИМАЦИЯ КРИВЫХ ПОСРЕДСТВОМ ТРИАНГУЛЯЦИИ

Чтобы аппроксимировать любую кривую, задайте анкерные точки, а затем выберите контрольную точку, образующую вместе с ними прямоугольный треугольник. Для контрольной точки задайте координату *x* одной анкерной точки и координату *y* другой. Направление кривой будет определяться тем, какая анкерная точка используется для задания той или иной координаты. Если такая аппроксимированная кривая недостаточно приближается к желаемому результату, немного сдвиньте контрольную точку и посмотрите, что у вас получилось. Выполняйте эту операцию до тех пор, пока не достигнете нужного результата.

Методику триангуляции можно использовать, к примеру, для создания окружности, состоящей из четырех сегментов.

С помощью кривых Безье практически невозможно нарисовать идеальную окружность. Однако с разделением на все большее количество сегментов, можно аппроксимировать ее с высокой степенью точности. Например, **Rash** аппроксимирует окружности с помощью восьми сегментов.

С помощью метода `curveTo()` и триангуляции можете нарисовать аппроксимированную окружность, состоящую из четырех сегментов. При этом используются простые математические операции сложения и вычитания. Такая окружность изображена на рис. 18.2. (Эта фигура взята из файла `fourSegmentCircle fla`, содержащегося на прилагаемом к книге компакт-диске.)

Чтобы увидеть, как выполняется триангуляция, рассмотрим окружность, вписанную в квадрат (рис. 18.3). Окружность касается квадрата в четырех точках. Примем эти точки за местоположения анкерных точек, являющихся конечными для четырехдуг, образующих окружность.

Теперь все, что нужно сделать, — это выяснить, где следует указать контрольную точку для каждой дуги. Каждый угол квадрата вместе с двумя анкерными точками образует прямоугольный треугольник, так что углы квадрата можно использовать для триангуляции.

Начните с задания четырех свойств (*top*, *bottom*, *left* и *right*) класса *Circle*, определенного ранее в этой главе. Эти свойства облегчат указание верхней, нижней, левой и крайней правой точек воображаемого квадрата, описанного вокруг окружности. Эти точки можно легко вычислить, поскольку верхняя точка круга лежит на расстоянии одного радиуса вверх, крайняя правая точка — на расстоянии одного радиуса вправо от центра и т.д.

```
function Circle (centerX, centerY, radius)
{
    this.center = new Point(centerX, centerY);
    this.radius = radius;
    this.top = centerY - radius;
    this.bottom = centerY + radius;
    this.left = centerX - radius;
    this.right = centerX + radius;
};
```

При использовании класса *Circle* рисование аппроксимации круга аналогично рисованию квадрата. Разница заключается в том, что вместо прямых рисуются кривые линии. Для каждой кривой необходимо указать вторую анкерную точку (первой всегда является текущее положение пера) и контрольную точку.

Координаты *x* и *y* контрольной точки являются первыми двумя параметрами метода *CurveTo()*. В приведенном ниже листинге можно увидеть, что они являются углами воображаемого квадрата, описанного вокруг окружности.

```
1: _root.createEmptyMovieClip( "circleClip", 1 );
2: circleClip.fourSegmentCircle = new Circle(400,175,150);
3: with ( _root.circleClip ) {
4:     with (fourSegmentCircle) {
5:         lineStyle( 6 );
6:         moveTo( center.x, bottom );
7:         curveTo( right, bottom, right, center.y );
8:         curveTo( right, top, center.x, top );
9:         curveTo( left, top, left, center.y );
10:        curveTo( left, bottom, center.x, bottom );
11:    }
12:}
```

Координаты *x* и *y* второй анкерной точки являются вторыми двумя параметрами метода *curveTo()*.

Рассмотрим первую дугу, являющуюся нижним правым квадрантом окружности.

Первая дуга начинается в точке, в которой нижняя часть окружности касается квадрата: (*center.x*, *bottom*). Метод *moveTo()* в строке 6 сообщает Flash о том, что начинать следует отсюда.

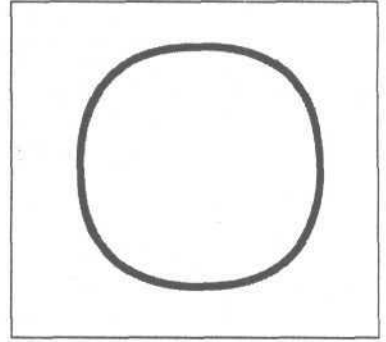


Рис. 18.2. Состоящая из четырех сегментов окружность, созданная с помощью метода *curveTo()* и триангуляции

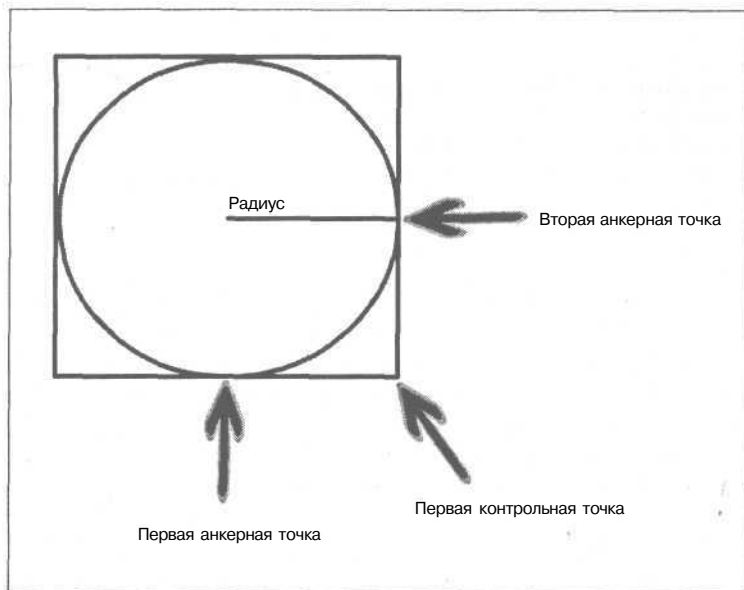


Рис. 18.3. При рисовании дуг с помощью триангуляции начните с представления окружности, вписанной в квадрат

Вторая анкерная точка первой дуги находится в точке, в которой правая часть круга касается квадрата: (`right, center.y`). Это указывает метод `curveTo ()` в строке 7.

Контрольной точкой первой дуги является правый нижний угол воображаемого квадрата: (`right, bottom`). На это также указывает метод `curveTo ()` в строке 7.

Остальные три дуги следуют той же самой стратегии триангуляции.

При желании после расчета точек триангуляции можно переместить их так, чтобы добиться наилучшей формы. Например, в содержащемся на компакт-диске файле `4SegCircle.fla` от разработчика Рика Юинга (Ric Ewing) для создания более идеальной формы окружности, состоящей из четырех сегментов, используется простой поправочный множитель.

ОПТИМИЗАЦИЯ ФУНКЦИЙ РИСОВАНИЯ КРИВЫХ

Если вы собираетесь рисовать несколько аналогичных кривых, подумайте об оптимизации функций рисования кривых. Существуют две схемы решения этой задачи.

- Сокращение методов общего назначения для устранения всего, что не требуется для определенной кривой, которую вы рисуете.
- Устранение расчетов путем замены констант.

Например, рассмотрим метод `DrawArc ()` в файле `drawMethods.fla`. Если вы рисуете только полные окружности, а не дуги, то для `arc` можно заменить константу 360. Можно также заменить 0 для `startAngle`, поскольку не имеет значения, в каком месте вы начинаете рисовать окружность. И наконец, если вы рисуете окружности, а не овалы, можно устранить переменную `yRadius` там, где это возможно, заменив ее на `radius`.

Программа будет работать быстрее, если заранее рассчитать математические функции. Например, в приведенной ниже функции рисования кривых все математические функции предварительно вычислены. Поскольку код оптимизирован, `radius` заменен на `r`.


```
function fastCircle(r, x, y) {
    this.moveTo(x+r, y);
    this.curveTo(0.9991*r+x, 0.4151*r+y, 0.7071*r+x, 0.7071*r+y);
    this.curveTo(0.4142*r+x, r+y, x, r+y);
    this.curveTo(-0.4151*r+x, 0.9991*r+y, -0.7071*r+x, 0.7071*r+y);
    this.curveTo(-r+x, 0.4142*r+y, -r+x, y);
    this.curveTo(-0.9991*r+x, -0.4151*r+y, -0.7071*r+x, -0.7071*r+y);
    this.curveTo(-0.4142*r+x, -r+y, x, -r+y);
    this.curveTo(0.4151*r+x, -0.9991*r+y, 0.7071*r+x, -0.7071*r+y);
    this.curveTo(r+x, -0.4142*r+y, r+x, y);
}
lineStyle(6);
fastCircle(50, 100, 100);
```

Когда Flash компилирует SWF-файлы, автоматически заменяются константы для математических функций, параметрами которых являются константы. Таким образом, если вы предпочитаете вместо выражения `Math.cos(45*Math.PI/180)` использовать константу `0.7071`, то знайте, что в SWF результат будет точно таким же.

Совет

Если программа выполняет много операций рисования в одном видеоклипе и вы обнаружили, что она работает очень медленно, обратитесь к разделу "Возможные проблемы" в конце этой главы.

ИСПОЛЬЗОВАНИЕ сплошной ЗАЛИВКИ

Совет

С понятием заливки мы познакомились в главе 3 "Рисование во Flash".

Замкнутые контуры можно заполнять либо сплошной, либо градиентной заливкой точно так же, как и при работе с обычными инструментами рисования. Для применения сплошной заливки используются методы `beginFill()` и `endFill()` класса `MovieClip`. Примеры использования этих методов см. в табл. 18.1.

Предложение `beginFill()` будет действовать применительно ко всем элементам, которые вы рисуете в определенном видеоклипе, до тех пор, пока не будет выполнено предложение `endFill()`, `beginFill()` или `beginGradientFill()`.

Поскольку метод `beginFill()` или `beginGradientFill()` замыкает любой текущий незамкнутый контур, то предложение `endFill()` часто оказывается ненужным. Во многих случаях результаты будут одинаковыми как при использовании метода `endFill()`, так и без него. Например, если во время применения заливки вы нарисуете отдельную кривую, то Flash автоматически замкнет ее и применит заливку к замкнутому контуру, независимо от наличия или отсутствия предложения `endFill()`.

Однако в некоторых случаях наличие предложения `endFill()` вызывает существенные отличия. Например, при создании двух перекрывающихся контуров без применения метода `endFill()` между методами `curveTo()` или `lineTo()`, использующимися для создания этих контуров, в области наложения заливка будет отменена и будет виден фон, как в двух накладывающихся треугольниках, изображенных на рис. 18.4, слева. Если же предложение `endFill()` помещено между двумя методами рисования (и методом `beginFill()`, обеспечивающим заливку второго контура), то оба контура будут полностью заполнены и второй контур будет расположен поверх первого, как показано на рис. 18.4, справа (этот рисунок взят из файла `filltest fla`, содержащегося на прилагаемом к книге компакт-диске). Для фигуры, расположенной справа, предложение `endFill()` применено после создания одного треугольника и перед началом создания следующего. Оба контура являются полностью за-

полненными. На рисунке слева метод `endFill()` не применяется. В области наложения заливка отменена и виден фон.

Графическое изображение, представленное на рис. 18.4, слева, было создано посредством приведенного ниже кода, без применения метода `endFill()`, прерывающего методы

```
lineTo(), создающие два треугольника.  
this.beginFill(0xFF0000);  
this.moveTo (one.x, one.y);  
this.lineTo(three.x, three.y);  
this.lineTo(two.x, two.y);  
this.lineTo (one.x, one.y);  
this.lineTo (four.x, four.y);  
this.lineTo(two.x, two.y);  
this.endFill();
```

Используемые здесь точки являются членами класса `Point`, определенного ранее в этой главе. Точка 1 на рис. 18.4 имеет имя `one`, точка 2 — `two` и т.д.

Графическое изображение, представленное на рис. 18.4, справа, было создано посредством приведенного ниже кода, с использованием методов `endFill()`, `beginFill()` и `moveTo()`, прерывающих методы `lineTo()`, создающие два треугольника.

```
this.beginFill(0xFF0000);  
this.moveTo (five.x, five.y);  
this.lineTo(seven.x, seven.y);  
this.lineTo(six.x, six.y);  
this.endFill();  
this.beginFill(0xFF0000);  
this.moveTo(five.x, five.y);  
this.lineTo (eight.x, eight.y);  
this.lineTo (six.x, six.y);  
this.endFill();
```

РАБОТА С ГРАДИЕНТНОЙ ЗАЛИВКОЙ

Градиентная заливка, представляющая собой полосы цветов, переходящих друг в друга, до сих пор является самой сложной функцией прикладного программного интерфейса рисования. Если вы до сих пор этого не сделали, начните знакомство с градиентной заливкой с процедур ее создания вручную. Тогда вы лучше поймете, чего следует добиваться при работе с программным эквивалентом этой операции.

Для реализации градиентной заливки используются методы `beginGradientFill()` и `endFill()` класса `MovieClip`. Примеры использования этих методов см. в табл. 18.1.

Метод `beginGradientFill()` будет действовать применительно ко всем элементам, которые вы рисуете в определенном видеоклипе, до тех пор, пока не будет выполнен метод `endFill()`, `clear()` либо метод `beginFill()` или `beginGradientFill()`.

Градиентная заливка может быть *линейной* (параллельные полосы) или *радиальной* (концентрические круги). Пример линейной градиентной заливки приведен на рис. 18.5, а радиальной градиентной заливки — на рис. 18.6.

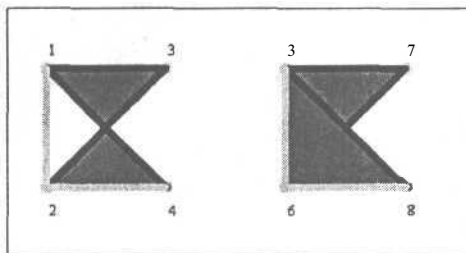


Рис 18.4 Два способа создания накладывающихся треугольников с заливкой

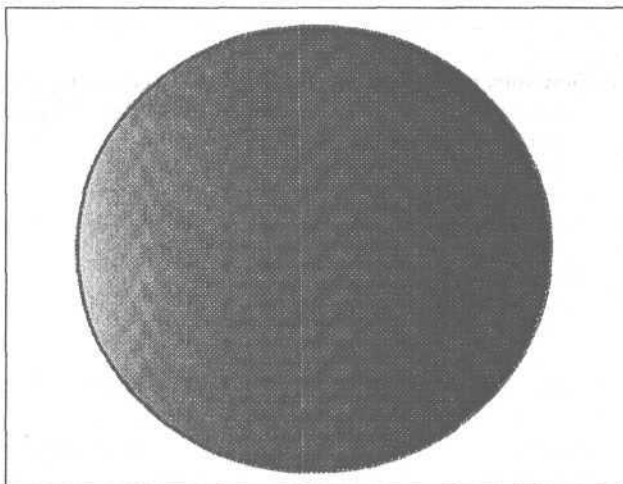


Рис. 18.5. Линейная градиентная заливка

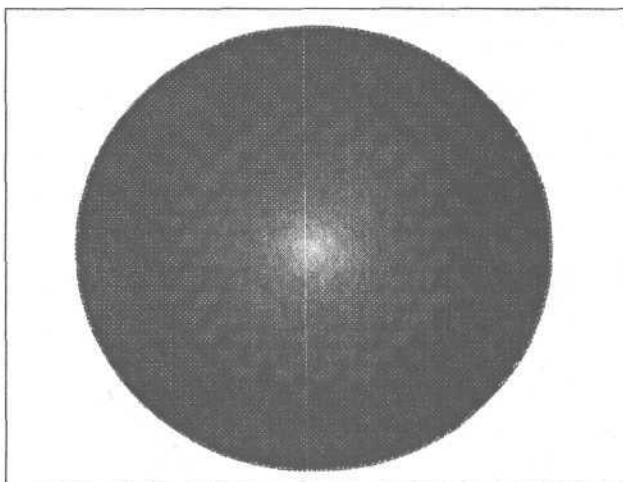


Рис. 18.6. Радиальная градиентная заливка

РАБОТА С ТРЕМЯ ГРАДИЕНТНЫМИ МАССИВАМИ

Три свойства градиентной заливки хранятся в трех различных массивах. Этими свойствами являются цвета, показатели прозрачности и коэффициенты.

Первым указывается массив *цветов*, которые будут использоваться в заливке. Каждое значение этого массива представляет собой число от 0 до 0xFFFFFF.

Вторым указывается массив *показателей прозрачности*. Каждое значение этого массива представляет собой число от 0 до 100 и применяется к соответствующему элементу массива. Таким образом, элемент `alphaArray[0]` задает прозрачность цвета в массиве `colorArray[0]`.

Третьим указывается массив *коэффициентов*. Каждое число этого массива задает точку, в которой Flash начинает заливку определенным несмешанным цветом. При движении в сто-

рону от этой точки Flash выполняет заливку цветом со все большим и большим процентом добавления соседнего цвета.

Задание этих чисел является программным эквивалентом перемещения маркеров полосы определения градиентной заливки на панели Color Mixer (Цветовой микшер) с целью определения ширины каждой цветовой полосы (для линейной заливки) или кольца (для радиальной заливки).

Каждое значение массива является числом от 0 до 255. Чтобы понять, как функционируют эти числа, представьте себе линейку с 255 метками, простирающуюся вдоль полосы определения градиентной заливки. Каждое число в массиве коэффициентов соответствует одной из этих меток, сообщая Flash о необходимости установления маркера соответствующего цвета в этом месте. Если задать число больше 255, то функция перестанет работать.

Flash считывает значения слева направо, в результате чего цвета, указанные в массиве раньше, будут предшествовать цветам, указанным позже. Например, если поместить на одной виртуальной "метке" несколько виртуальных "маркеров", то Flash отобразит только цвет, соответствующий первому маркеру. (Может немного проявиться смесь скрытых цветов.)

Flash всегда применяет заливку ко всей области, для которой она была назначена. Эта операция выполняется путем задания последнего цвета массива для любой необъявленной области. Таким образом, при задании нулевого коэффициента для всех цветов вся область будет залита сплошным цветом, являющимся последним в массиве.

Число элементов во всех массивах должно быть одинаковым, иначе функция работать не будет. На практике число элементов в массивах должно быть не меньше 2 и не больше 8. Фактически число элементов в массивах может быть любым, лишь бы оно было одинаковым во всех массивах, но для создания градиента требуются минимум два цвета. Если бы в каждом массиве было по одному элементу, получилась бы сплошная, а не градиентная заливка. Кроме того, цвета после восьмого не отображаются. Таким образом, Flash отобразит цвета, соответствующие элементам массива, начиная с `colorArray [0]` и до `colorArray [7]`, а цвет, соответствующий элементу `colorArray [8]`, отображен не будет.

МАТРИЦА ПРЕОБРАЗОВАНИЯ

Матрица преобразования позволяет выполнять некоторые стандартные преобразования градиентной заливки: трансляцию (указание центральной точки заливки), масштабирование (изменение размера смешанной части заливки) и вращение. Вращение изменяет ориентацию всей заливки. Например, поворот на 90 градусов изменяет заданные по умолчанию вертикальные полосы линейной заливки на горизонтальные.

Матрица представляет собой объект, который можно указать с помощью литерала объекта либо создать с помощью пользовательского класса и оператора `new`.

Совет

Подробная информация о литералах объектов приведена в главе 12 "Управление переменными, данными и типами данных".

Матрицу определяют путем создания объекта, свойства которого становятся элементами матрицы. Этот объект используется в качестве последнего параметра, переданного методу `beginGradientFill()`.

Создаваемый объект матрицы может содержать следующие свойства.

- Девять свойств, определяющих матрицу 3x3, как в обычной математике. Для реализации методов трансляции, вращения и масштабирования обычного объекта матрицы предназначен файл с расширением `.AS` (файл "включения"). Это делает традиционные операции с матрицей интуитивно понятными, но для каждого преобразования (трансляции, вращения, масштабирования) требуется отдельное обращение к функции.
- Шесть свойств, представляющих параметры трансляции, вращения и масштабирования. При каждом обращении вы можете выполнять одну, две или три операции. Этот формат более краток и интуитивно понятен.

Если вы хотите выполнить только трансляцию, вращение и масштабирование, то оба формата выполняют эту задачу совершенно одинаково. Матрица 3x3 более эффективна при управлении порядком операций масштабирования, трансляции и вращения, поскольку ее можно преобразовывать несколько раз, а вызывать метод `beginGradientFill()` — только один раз.

Одни и те же операции, выполненные в разном порядке, дадут различные результаты. Выполнение их в порядке **масштабирование—трансляция—вращение** не является аналогичным последовательности **трансляция—вращение—масштабирование**.

При одновременном выполнении нескольких операций в кратком формате порядок всегда будет таким: масштабирование-вращение-трансляция. Операции можно выполнять и по одной, меняя таким образом их порядок, но при этом каждый раз придется вызывать метод `beginGradientFill()`.

Традиционный формат матрицы используется в случае, если вы не хотите ограничиться только операциями трансляции, вращения и масштабирования. К примеру, можно создать свои собственные функции матриц.

КРАТКИЙ ФОРМАТ МАТРИЦЫ ПРЕОБРАЗОВАНИЯ

Приведенное ниже предложение создает объект матрицы с именем `m` (имя является обязательным), используя краткий формат.

```
m = { matrixType:"box", X:100, y:100, w:200, h:200, r:(45/180)*Math.PI };
```

Первым свойством всегда является строка `"box"`. Свойства `x` и `y` используются для выполнения трансляции, `w` и `h` — для масштабирования, а `r` — для реализации вращения.

Трансляция означает перемещение градиента. По умолчанию Flash размещает центр градиента в верхнем левом углу клипа, в котором расположена заливка. При трансляции каждая точка градиента перемещается по горизонтали на величину `x` и по вертикали на величину `y`.

Ниже приведено несколько примеров трансляции.

- Для линейной градиентной заливки, которая не была повернута и таким образом представляет собой ряд вертикальных цветовых полос, смещение по вертикали не влияет на внешний вид градиента.
- Смещение неразвернутой линейной градиентной заливки по горизонтали аналогично перемещению цветовых маркеров полосы определения градиентной заливки на панели Color Mixer (Цветовой микшер) на указанную величину.
- Если линейную градиентную заливку повернуть на 90 градусов, мы получим ряд горизонтальных цветовых полос, для которых смещение по горизонтали не имеет никакого значения. Смещение по вертикали аналогично перемещению цветовых маркеров.

Об операции масштабирования стоит сказать только то, что масштабируется *смешанная* часть заливки. При увеличении значений `w` и `h` все большая часть заливки становится смешанной. При уменьшении этих значений мы все больше приближаемся к чистому насыщенному цвету.

Имейте в виду, что величина **вращения** указывается в радианах. В приведенном выше примере поворот на 45 градусов преобразован в радианы путем деления на 180 и умножения на π .

Совет

Подробная информация о радианах представлена в главе 19 "Использование встроенных базовых объектов".

ИСПОЛЬЗОВАНИЕ ТРАДИЦИОННОЙ МАТРИЦЫ ПРЕОБРАЗОВАНИЯ 3x3

Традиционная матрица преобразования 3x3 является основой всех операций с матрицами во Flash.

Матрица преобразования 3x3 часто используется для выполнения двух- или трехмерных преобразований. Матрица имеет такой формат:

a	b	c
d	e	f
g	h	i

В табл. 18.2 представлены девять элементов матрицы преобразования 3x3 и описан каждый из них.

ТАБЛИЦА 18.2. ЭЛЕМЕНТЫ МАТРИЦЫ ПРЕОБРАЗОВАНИЯ 3x3

ЭЛЕМЕНТ	ОПИСАНИЕ
a	Задаёт свойство <code>xscale</code> градиентной заливки
b	Наклон и масштабирование x (используется при вращении)
c	Всегда равен 0 при двухмерных преобразованиях
d	Наклон и масштабирование y (используется при вращении)
e	Задаёт свойство <code>yscale</code> градиентной заливки
f	Всегда равен 0 при двухмерных преобразованиях
g	Задаёт координату x центра градиентной заливки
h	Задаёт координату y центра градиентной заливки
i	Всегда равен 1 при двухмерных преобразованиях

При выполнении двухмерных преобразований элементы c и f всегда равны 0, а элемент i всегда равен 1. Эти значения можно изменять только при выполнении трехмерных преобразований, однако с градиентной заливкой Flash такие преобразования не выполняются.

Объект обычной матрицы 3x3 можно создать следующим образом:

```
matrix = {a: 200, b:0, c:0, d:0, e:200, f:0, g:0, h:0, i:1};
```

Используя файл включения, начать создание матрицы преобразования можно так:

```
ttinclude "TransformMatrix.as"
_root.myMatrix = new TransformMatrix();
```

Продолжая предыдущий пример с использованием файла включения, операции масштабирования, вращения и трансляция выполняются следующим образом:

```
_root.myMatrix.scale(300, 300);
_root.myMatrix.rotate(45);
_root.myMatrix.translate(300,300);
```

Вот и все, что нужно знать для успешного применения объекта традиционной матрицы с методом `beginGradientFill()`.

В приведенном ниже листинге сведено все, что вы узнали о градиентной заливке и матрице преобразования 3x3.

```
1: ttinclude "TransformMatrix.as"
2:
3: _root.myMatrix = new TransformMatrix();
4: _root.myMatrix.rotate(45);
5: _root.myMatrix.scale(350, 200);
6: _root.myMatrix.translate(300,300);
7: RED = 0xFF0000;
```

```

8: YELLOW = 0xFFFF00;
9: GREEN = 0x00FF00;
10: _root.createEmptyMovieClip("myGradientClip", 8);
11: with (_root.myGradientClip) {
12:     colors = [RED, YELLOW, GREEN];
13:     alphas = [100, 100, 90];
14:     ratios = [0, 127, 255];
15:     beginGradientFill ("linear", colors, alphas, ratios,
_root.myMatrix);
16:     lineStyle(10, 0x0000FF);
17:     moveTo(100, 100);
18:     lineTo(400, 100);
19:     lineTo(400, 400);
20:     lineTo(100, 400);
21:     lineTo(100, 100);
22:     endFill();
23: }

```

На заметку

Очень важно оставить пустую строку между предложением включения и следующей строкой. Отделение предложения включения от остального кода пустой строкой считается признаком хорошего тона.

В строке 1 содержится файл включения Macromedia. Строка 3 содержит матрицу. Строки 4–6 преобразуют матрицу. В строках 7–9 создаются константы, которые делают цветовые массивы более удобочитаемыми. Строка 10 создает новый видеоклип, в котором выполняется рисование. Строки 12–14 создают массивы, а строка 15 — градиентную заливку. Остальная часть программы предназначена для рисования контуров и завершения заливки.

Совет

Получили неожиданные результаты заливки? Обратитесь к разделу "Возможные проблемы" в конце этой главы.

ПОНИМАНИЕ ОПЕРАЦИЙ с МАТРИЦОЙ 3x3

В данном подразделе операции преобразования матрицы 3x3 будут рассмотрены детально. Вы получите более точное представление о том, что означают операции трансляции, вращения и масштабирования применительно к градиентной заливке. Кроме того, немного познакомившись с файлом включения Macromedia, вы получите представление о тех действиях, которые нужно выполнить для создания собственных функций матрицы. К ним относятся некоторые математические операции, в том числе и тригонометрические.

Матрица 3x3 используется для преобразования точки с координатами (x, y) в точку с координатами (x2, y2) посредством следующих уравнений (написанных на языке ActionScript):

$$\begin{aligned}
 x_2 &= a \cdot x + d \cdot y + g; \\
 y_2 &= b \cdot x + e \cdot y + h;
 \end{aligned}$$

Таким образом Flash преобразует каждую отдельную точку в градиент.

В следующих подразделах кратко описано, как Flash использует приведенные выше уравнения для выполнения операций трансляции, масштабирования и вращения. Примеры реализации этих функций в ActionScript приведены в файле матрицы преобразования .AS от компании Macromedia.

МАСШТАБИРОВАНИЕ

Для выполнения масштабирования с помощью матрицы 3x3 параметр a задают равным значению масштабного коэффициента координаты x, а параметр e — равным значению мас-

штабного коэффициента координаты y . Параметры d , g , b и h устанавливаются равными нулю. В результате каждая координата x и y умножается на соответствующий масштабный коэффициент. Кроме этого больше ничего не происходит.

Обратите внимание на то, что масштабирование относится к смешанной части заливки. При увеличении параметров a и e все большая часть заливки становится смешанной, а при их уменьшении цвет становится все более чистым и насыщенным.

ТРАНСЛЯЦИЯ

Для выполнения трансляции с помощью матрицы 3×3 параметры a и e задают равными 1, т.е. $a \cdot x$ равно x , а $e \cdot y$ равно y . Затем параметры g и h задают равными коэффициентам трансляции координат x и y . Каждая координата x становится равной $x+g$, а координата y — $y+h$. В результате каждая точка градиентной заливки смещается на соответствующую величину трансляции.

ВРАЩЕНИЕ

Выполнение вращения (поворота) требует познаний в тригонометрии.

- Параметр a устанавливается равным косинусу угла поворота.
- Параметр b устанавливается равным синусу угла поворота.
- Параметр d устанавливается равным величине, противоположной по знаку синусу угла поворота.
- Параметр e устанавливается равным косинусу угла поворота.
- Параметры a и b устанавливаются равными нулю.

Подставим эти значения в два уравнения, приведенных выше, и получим следующее выражение:

```
x2 = Math.cos(r)*x - Math.sin(r)*y;  
y2 = Math.sin(r)*x + Math.cos(r)*y;
```

Совет

Чтобы получить наглядное представление о том, какую роль играют уравнения матрицы преобразования и как они соотносятся с тригонометрическими функциями, обратитесь к файлу `trigdemo fla`, который описывается в главе 19 "Использование встроенных базовых объектов". В этом файле показано, как подобно часовой стрелке, но в обратном направлении, вращается зеленая линия.

Каждая точка градиентной заливки поворачивается подобно стрелке часов. Flash всегда трактует положительное вращение как направление по часовой стрелке, хотя традиционной положительным считается вращение против часовой стрелки.

ОБЪЕДИНЕНИЕ ОПЕРАЦИЙ МАТРИЦЫ

Если вы хотите скомбинировать операции матрицы, например, создать одну матрицу, которая выполняет и трансляцию, и масштабирование, примените процедуру, именуемую *матричным умножением*, или *сочленением*. В файле `.AS` матрицы преобразования от Macromedia содержится функция сочленения. Фактически функции масштабирования, вращения и трансляции матрицы 3×3 представляют собой замаскированные функции сочленения. Если вы посмотрите на них в файле `.AS`, то увидите, что каждая из них создает новую матрицу (инициализируемую для каждой из операций так, как описано в предыдущих подразделах), а затем сочленяет ее с матрицей, использующейся при вызове функции.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему моя программа рисования стала работать с прерываниями и замедлилась?

Если ваша программа выполняет много операций рисования в одном видеоклипе, она может замедлиться и начать работать рывками. Если распределить то же количество рисунков по нескольким видеоклипам, вы обнаружите, что программа будет работать лучше. Эта методика используется, к примеру, в содержащемся на компакт-диске файле `scratch.fl` от разработчика Питера Холла (Peter Hall) (описанного в главе 17 "Демонстрация мощи видеоклипов").

Что делать, если при выполнении заливки я получаю не те результаты, на которые рассчитывал?

При вставке предложений `endFill` и `beginFill` неплохо было бы использовать и метод `moveTo()`, чтобы перейти к следующему контуру, даже если этот метод не изменяет текущее положение пера и на первый взгляд кажется бесполезным. Если этого не сделать, то иногда можно получить неожиданные результаты.

Еще одна возможная причина получения нежелательных результатов может заключаться в применении слишком малых коэффициентов масштабирования. Масштабирование "почти один к одному" может привести к аномальным результатам, и чем ближе к 1 коэффициент масштабирования, тем хуже будет "вести себя" заливка, особенно при выполнении операции вращения.

Естественно, нет никаких оснований для того, чтобы выполнять масштабирование с коэффициентом 1, поскольку при этом нет никакого градиента. По этой причине такое поведение официально называется FOL (Fact of Life — факт из жизни), и оно не является дефектом программы.

В приведенном ниже коде, взятом из содержащегося на компакт-диске файла `matrixbug.fl`, показано, каких операций следует избегать. Мы получаем цветовой переход "красный/зеленый/красный", тогда как ожидаем перехода "красный/желтый/зеленый". (Нужные результаты продемонстрированы в содержащемся на компакт-диске файле `Matrix.fl`.)

```
// w:1, h:1 == непредсказуемые результаты !!
_root.m = { matrixType:"box", x:100, y:100, w:1, h:1,
r: (0/180)*Math.PI};
RED = 0xFF0000;
YELLOW = 0xFFFF00;
GREEN = 0x00FF00;
_root.createEmptyMovieClip("myGradient", 8);
with (_root.myGradient) {
    colors = [RED, YELLOW, GREEN];
    alphas = [100, 100, 90];
    ratios = [0, 12, 255];
    beginGradientFill ( "linear", colors, alphas, ratios, _root.m);
    lineStyle(10, 0x0000FF);
    moveTo(0,0);
    lineTo(300,0);
    lineTo(300,300);
    lineTo(0,300);
    lineTo(0,0);
    endFill();
}
```

FLASH ЗА РАБОТОЙ: РИСОВАНИЕ КРУГА, СОСТОЯЩЕГО ИЗ ВОСЬМИ СЕГМЕНТОВ

Концептуально рисование круга, состоящего из восьми сегментов, не слишком отличается от аналогичной процедуры для круга, состоящего из четырех сегментов. Исключение составляет всего один шаг. В данном случае круг нужно вписать в восьмиугольник, как показано на рис. 18.7, и выяснить, где находятся "углы" восьмиугольника, которые будут использоваться в качестве контрольных точек. (Данный рисунок основан на коде из содержащегося на компакт-диске файла `eightSegmentCircle fla.`)

Определение координат углов требует применения тригонометрии. В целом, когда необходимо нарисовать кривые, вписанные в правильные многоугольники, легче всего это сделать с помощью тригонометрических функций. Альтернативным способом является экспериментирование с контуром до тех пор, пока не будет найдена форма, близкая к желаемой (как в файле `4SegCircle fla.`).

Рассмотрим, как можно определить координаты на примере одной контрольной точки для круга, состоящего из восьми сегментов. Взгляните на контрольную точку с правой стороны круга (см. рис. 18.7), помеченную как (cx, cy) . Она является одной из вершин треугольника внутри круга. (Отметки на рисунке соответствуют переменным метода `drawArc()` файла `drawMethods fla.`, содержащегося на компакт-диске.)

ОПРЕДЕЛЕНИЕ КООРДИНАТ x КОНТРОЛЬНОЙ ТОЧКИ

Координату x точки (cx, cy) легко определить прямо из рисунка, т.е. для ее определения тригонометрические функции применять не нужно. Эта точка располагается прямо над точкой (x, y) , находящейся на расстоянии одного радиуса от центра круга. Все, что нужно сделать, — это прибавить радиус к значению координаты x центра круга ax . Например, если ax равно 400, а радиус равен 150, то cx равно 550.

ОПРЕДЕЛЕНИЕ КООРДИНАТЫ y КОНТРОЛЬНОЙ ТОЧКИ

Чтобы выяснить, насколько далеко контрольная точка располагается от центра круга по вертикали (т.е. координата y точки cy , изображенной на рис. 18.7), необходимо применить

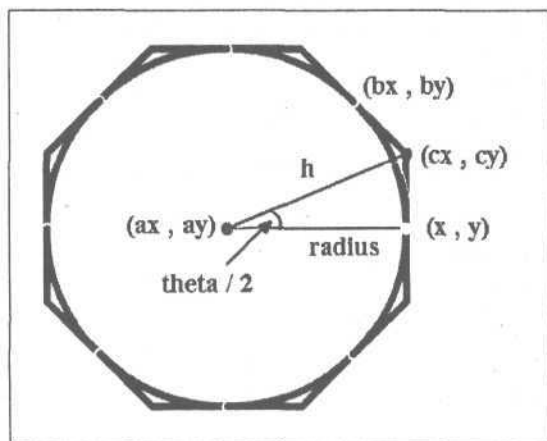


Рис. 18.7. Чтобы понять процедуру создания круга, состоящего из восьми сегментов, сначала представьте себе, что вы вписываете круг в восьмиугольник

тригонометрические функции. Расчет проходит в три этапа. Исходя из рис. 18.7, эти этапы заключаются в следующем:

- определение величины угла $\theta / 2$;
- определение длины h — гипотенузы треугольника;
- использование этих двух чисел для определения cy .

ОПРЕДЕЛЕНИЕ ВЕЛИЧИНЫ УГЛА $\theta / 2$

При делении круга на восемь частей каждая часть равна 45 градусам ($360/8$). Именем переменной, назначенной для угла в методе `drawArc()`, является θ (греческая буква, традиционно используемая для обозначения углов). Угол, который на рис. 18.7 обозначен как $\theta / 2$, является половиной из частей, на которые разделен круг. Следовательно, $\theta / 2$ равно 22,5 градуса.

ОПРЕДЕЛЕНИЕ длины h — ГИПОТЕНУЗЫ ТРЕУГОЛЬНИКА

Вот и наступил черед тригонометрии. По определению косинус угла $\theta / 2$ на рис. 18.7 равен величине `radius`, деленной на сторону h треугольника (его гипотенузу). На языке ActionScript это выражение будет таким: `Math.cos(theta / 2)` равно `radius / h`.

Умножим обе части равенства на h и разделим на `Math.cos(theta / 2)`. В результате h равно `radius / Math.cos(theta / 2)`.

ОПРЕДЕЛЕНИЕ cy

По определению синус $\theta / 2$ равен cy , деленному на h . На языке ActionScript `Math.sin(theta / 2)` равно `cy / h`.

Умножим обе части равенства на h . В результате получим, что cy равно `Math.sin(theta / 2) * h`.

Мы знаем, что h равно `radius / Math.cos(theta / 2)`. Следовательно, cy равно `Math.sin(theta / 2) * radius / Math.cos(theta / 2)`.

В конце метода `drawArc()` в содержащемся на компакт-диске файле `drawingMethods fla` вы обнаружите выражение, похожее на то, что использовалось для расчета cy . Значение sx рассчитывается по похожей формуле (в данном примере его можно получить по аналогии).

В методе `drawArc()` используется несколько других переменных. Например, этот метод наряду с кругами работает и с овалами, следовательно, он имеет переменные `radius` и `yRadius`, о которых при работе с кругами беспокоиться не нужно.

РАСШИРЕННЫЕ СЦЕНАРИИ ACTIONSCRIPT

В ЭТОЙ ЧАСТИ...

Глава 19. Использование встроенных базовых объектов

Глава 20. Использование встроенных объектов фильмов

Глава 21. Упаковка данных и функций с помощью пользовательских объектов

Глава 22. Компоненты

Глава 23. Использование обучающих взаимодействий

Глава 24. Коллективное использование **ActionScript**

Глава 25. Тестирование, отладка и устранение проблем

ИСПОЛЬЗОВАНИЕ ВСТРОЕННЫХ БАЗОВЫХ ОБЪЕКТОВ

В ЭТОЙ ГЛАВЕ...

Знакомство со встроенными объектами	419
Базовые объекты	420
Возможные проблемы	464
Flash за работой: удивительные массивы	465

ЗНАКОМСТВО со ВСТРОЕННЫМИ ОБЪЕКТАМИ

Встроенные объекты являются самым распространенным средством отображения функциональных возможностей Flash для программиста, работающего на языке **ActionScript**. Встроенные объекты, являющиеся "рабочими лошадками" ActionScript, используются для выполнения основных рутинных задач, таких как управление графикой, **цветами**, звуками, нажатием клавиш и другими элементами Flash-фильма.

Во Flash MX встроенные объекты стали всеобъемлющими. Теперь в программе есть объектно-ориентированная основа для текстовых полей и функций форматирования текста, кнопок, рабочего поля, специальных добавлений к пользовательскому интерфейсу Flash (**CustomActions**), переменных, хранящихся в файлах на локальном диске в качестве "объектов коллективного использования", видео, переменных, загружаемых с серверов (**LoadVars**), локальных каналов взаимосвязи с другими SWF-файлами (**LocalConnection**) и соединений с серверами связи и серверами приложений (**NetConnection/NetStream**).

Встроенные объекты подразделяются на две основные категории: *функции-конструкторы* (или классы) и *глобальные объекты*. Основное отличие между ними заключается в том, что с

помощью функций-конструкторов можно создавать новые экземпляры, тогда как с помощью глобальных объектов этого делать нельзя.

Глобальный объект также называют *одноэлементным множеством*, поскольку он является экземпляром класса, разрешающего только один экземпляр. Например, можно иметь только одно рабочее поле, одну клавиатуру, одну мышь и одно выделенное в данный момент текстовое поле. Одноэлементные множества являются экземплярами, в которых функции-конструкторы не функционируют.

С помощью глобальных объектов нельзя выполнять три важные операции.

- Создавать новые экземпляры этих объектов.
- Выполнять их функции-конструкторы.
- Изменять значения существующих свойств, в том числе и методов.

Совет

Подробная информация о функциях-конструкторах и их роли в создании объектов приведена в главе 15 "Объединение предложений в функции".

Как правило, для создания новых экземпляров функция-конструктор выполняется вместе с ключевым словом `new`. Однако существуют четыре "неконструктивных" (моя терминология) функции-конструктора: `Button`, `Function`, `MovieClip` и `Video`, которые не могут создавать пригодные к употреблению новые экземпляры, но имеют другие области применения. Что касается этих классов, то их экземпляры создаются другими способами.

Так, объект `Video` создается на панели `Library` (Библиотека) путем выбора из меню ее параметров опции `New Video` (Новое видео). (Для открытия меню параметров щелкните на кнопке, расположенной в правом верхнем углу панели.)

Совет

Видеоклипы создаются как вручную, так и программным образом с помощью методик, описанных в главе 17 "Демонстрация мощи видеоклипов".

Кнопки создаются одним из двух способов: либо вручную (по той же методике, что и видеоклипы), либо с помощью метода `attachMovie ()`.

Совет

Функции создаются вручную с помощью методик, описанных в главе 15 "Объединение предложений в функции".

БАЗОВЫЕ ОБЪЕКТЫ

Восемь встроенных объектов Flash MX определены стандартом ECMA-262 и внедрены в JavaScript. (Однако иногда Flash MX немного отличается от стандарта и от JavaScript.) В данной главе рассматриваются эти восемь объектов, которые в средствах разработки Flash называются *базовыми*.

С помощью базовых объектов реализуются основные функции управления данными, используемые для выполнения самых разных задач программирования, в том числе и для создания Web-анимации. Однако базовые объекты не имеют никаких особенностей, являющихся специфическими для Web-анимации. Восемь базовых объектов перечислены в табл. 19.1.

КЛАСС ARRAY

Массив представляет собой упорядоченную пронумерованную совокупность элементов данных. Массив можно использовать в качестве "картотеки" данных. Каждый элемент массива подобен ящику картотеки, на котором проставлен *индекс* (порядковый номер). Индексы начинаются с нуля и отсчитываются как положительные целые числа. Весь массив также должен содержаться в элементе данных, чтобы к нему можно было обращаться как к единому объекту.

ТАБЛИЦА 19.1. БАЗОВЫЕ ОБЪЕКТЫ

ОБЪЕКТ	ПРЕДСТАВЛЯЕТ	ОДНОЭЛЕМЕНТНОЕ МНОЖЕСТВО или КОНСТРУКТОР
Array (Массив)	Упорядоченный список	Конструктор
Boolean (Булева величина)	Значение типа true/false или другая двойственность	Конструктор
Date (Дата)	Календарная дата/время	Конструктор
Function (Функция)	Функция	Конструктор(*)
Math (Математический)	Математические функции и константы	Одноэлементное множество
Number (Число)	Числа	Конструктор
Object (Объект)	Неупорядоченная совокупность	Конструктор
String (Строка)	Буквенно-цифровые данные	Конструктор

“Неконструктивный” конструктор: с помощью ключевого слова new нельзя создавать пригодные к употреблению новые экземпляры.

Совет

Примеры операций с массивами приведены в конце этой главы, в разделе “Flash за работой: удивительные массивы”.

В табл. 19.2 сведены три типа данных, требующихся для работы с массивами: элементы, индексы и контейнер данных всего массива.

ТАБЛИЦА 19.2. Типы ДАННЫХ, НЕОБХОДИМЫЕ для РАБОТЫ с МАССИВАМИ

ТИП ДАННЫХ	АНАЛОГИЯ С КАРТОТЕКОЙ	ДОПУСТИМЫЕ ЗНАЧЕНИЯ
Элементы массива	Каждый элемент подобен со- держимому одного ящика кар- тотеки	Любое допустимое выраже- ние
Индекс каждого элемента	Числовая надпись на каждом ящике картотеки	Любое выражение, преобра- зующееся в целое число, на- чиная с 0 и до Number.MAX_VALUE вклю- чительно
Контейнер данных для всего массива	Надпись на всей картотеке	Переменная, элемент массива или свойство объекта, относя- щиеся к массиву

Элемент массива может содержать любой тип данных, существующий в **ActionScript**: число, строку, булеву величину, функцию, объект, массив, т.е. любой простой или сложный элемент дан-ных. Для определения элемента массива может использоваться любое допустимое выражение.

На заметку

Выражение — это фраза ActionScript, преобразующаяся в элемент данных.

В одном массиве могут содержаться данные разных типов.
Индексом может быть любое неотрицательное целое число или любое выражение, преоб-разующееся в неотрицательное целое число. Максимальным разрешенным индексом являет-

ся значение `Number.MAX_VALUE`. При указании значения, превышающего его, в качестве индекса Flash все равно будет использовать `Number.MAX_VALUE`.

На заметку

`Number.MAX_VALUE` — это самое большое число, которое можно представить в ActionScript (`1,79769313486231e+308`).

Контейнером данных, позволяющим обращаться ко всему массиву как к единому объекту, может быть переменная, элемент массива или свойство объекта.

СОЗДАНИЕ МАССИВОВ

Массивы можно создавать двумя способами: с помощью литерала массива или функции-конструктора `Array()` вместе с ключевым словом `new`.

При создании массива посредством литерала массива используется такой формат:

```
контейнер = [выражение1, выражение2, выражение3];
```

где *контейнер* — это переменная, элемент массива или свойство объекта, относящиеся ко всему объекту, а выражения внутри квадратных скобок определяют элементы массива. Приведем несколько примеров:

```
myArray = ["John", "Mary", "Peter", "Pat"]; // контейнером является переменная
```

```
students = new Object();  
students.scores = [98, 95, 87, 65]; // контейнером является свойство объекта
```

```
myMixedArray = ["John", 98, "Mary", 95]; // массив данных разных типов
```

С помощью функции-конструктора `Array()` массив можно создать тремя способами.

- Вызвать конструктор без аргументов:

```
myArray = new Array();
```

При таком подходе создается *пустой* массив, т.е. не содержащий элементов.

- Вызвать функцию-конструктор `Array()` с одним числовым аргументом:

```
myArray = new Array(8);
```

Единственное, что здесь происходит, — это установка в указанное значение свойства `length` (длина) массива. При таком подходе создается массив с указанным числом элементов, каждый из которых имеет значение `undefined`. Однако при обращении к любому из элементов пустого массива также будет возвращено значение `undefined`, поэтому свойство `length` массива является единственным способом отличить пустой массив (без элементов) от массива с пустыми (неопределенными) элементами.

- Вызвать функцию-конструктор `Array()` с аргументами, являющимися значениями элементов массива. Первый аргумент становится значением первого элемента массива, второй аргумент — значением второго элемента массива и т.д. Например:

```
myArray = new Array(8, 10, 12);  
myArray = new Array("John", "Matt", "Mary", myVar, myArray, myObj);
```

Если функция-конструктор `Array()` видит только один числовой аргумент, она предполагает, что массив создается вторым способом, и назначает свойство `length`. Если конструктор видит один нечисловой аргумент или несколько аргументов, то аргументы используются в качестве значений элементов массива (способ 3).

ОБРАЩЕНИЕ К ЭЛЕМЕНТАМ МАССИВА

Для обращения к элементам массива используются индексы. Формат обращения следующий:

контейнер[индекс]

Таким образом, первым элементом массива `myArray` будет `myArray[0]`, вторым элементом — `myArray[1]` и т.д. Этот формат используется как для считывания, так и для записи элементов массива. Ниже приведено несколько примеров считывания.

```
MyVar = myArray[2]; // считывание элемента в переменную
myObj.myProp = myArray[6]; // считывание элемента в свойство объекта
```

Далее следует несколько примеров записи:

```
myArray[2] = myVar; // запись переменной в элемент
myArray[6] = myObj.myProp; // запись свойства объекта в элемент
```

А вот пример считывания из одного массива записи в другой:

```
myArray[3] = myOtherArray[0];
```

Тот факт, что к элементам массива можно обращаться с помощью чисел, является основным показателем мощи массивов. Наиболее распространенной методикой является использование индексной переменной и свойства массива `length` для задания цикла `for`. Этот цикл выполняет поиск всех элементов в массиве, тестирует тем или иным способом каждый из них и на основе полученных результатов предпринимает какое-либо действие. В приведенном ниже примере выполняются поиск в массиве имен и тестирование каждого имени для того, чтобы проверить, равно ли оно заданному значению, а затем — замена на другое в случае выполнения условия.

```
NameArray = new Array("John Jones", "Mary Peterson", "Michael Smith");
for (i = 0; i < nameArray.length ; i++) { // задание
    if (nameArray[i] == "Mary Peterson") { // тестирование
        nameArray[i] = "Mary Laliberte"; // действие на основе
        //тестирования
    }
}
```

Обратите внимание на то, что необходимо использовать выражение `i < nameArray.length`, а не `i <= nameArray.length`, поскольку последний индекс массива *на единицу меньше* значения свойства `length`.

Если определенную операцию массива необходимо выполнить несколько раз, ее можно сократить до функции. Например, показанную в предыдущем примере операцию поиска и замены можно преобразовать в такую функцию:

```
function searchAndReplace(inThisArray, findThis, replaceWithThis) {
    for (var i = 0; i < inThisArray.length ; i++) { // задание
        if (inThisArray[i] == findThis) { // тестирование
            inThisArray[i] = replaceWithThis; // действие на основе
            // тестирования
        }
    }
}
```

Как вы заметили, в выражение `var i = 0`; добавлено ключевое слово `var`. Это делает `i` локальной переменной функции `searchAndReplace()` и предотвращает ее взаимодействие с другими случаями использования `i` в качестве переменной в иных местах программы.

Совет

Подробная информация о ключевом слове `var` приведена в главе 15 "Объединение предложений в функции".

ДОБАВЛЕНИЕ ЭЛЕМЕНТОВ МАССИВА

Добавить новые элементы в массив можно путем присвоения им значений. В приведенном ниже примере создается пустой массив, а затем в него добавляется элемент с индексом 4.

```
MyArray = new Array();  
myArray[4] = «testing»;
```

Этот код также задает свойство `myArray.length` равным 5.

Приведенный пример иллюстрирует создание *разреженного* массива, в котором элементы, имеющие значения, "разбросаны" среди элементов, которым значения не присвоены. Во Flash выделяется память только для элементов с назначенными значениями, таким образом, использование разреженных массивов в ней не запрещается.

В предыдущем подразделе было показано, что для добавления элемента в массив и для замены существующего элемента с определенным индексом применяется один и тот же синтаксис.

УДАЛЕНИЕ ЭЛЕМЕНТОВ МАССИВА

При удалении элементов, находящихся в середине массива, программист стремится выполнить следующие задачи:

- изменить значение элемента на `undefined`;
- перераспределить память, ранее использовавшуюся для хранения значения в элементе.

За выполнение обеих задач отвечает оператор `delete`. Например:

```
delete myArray[6];
```

При удалении *последнего* элемента массива программист попытается выполнить обе предыдущие задачи и, вероятно, уменьшить на единицу значение свойства `length` данного массива. Все три задачи можно выполнить с помощью метода `pop()` класса `Array`:

```
myArray.pop(); // удаляется последний элемент, уменьшается длина
```

Метод `pop()` также возвращает значение удаленного элемента.

Совет

Подробная информация о методе `pop()` приводится далее, в подразделе "Методы `pop()` и `push()`".

Если вы хотите удалить последний элемент вручную и уменьшить значение свойства `length`, то это можно также выполнить следующим образом:

```
myArray = new Array("John", "Matt", "Joe");  
delete myArray[2]; // удаление элемента "Joe", перераспределение памяти  
trace(myArray[2]); // undefined  
myArray.length = 2; // длина была равна 3, ее уменьшили на 1
```

Если просто задать для элемента значение `undefined`, перераспределение памяти *не произойдет*. Например, приведенный ниже код будет вызывать постепенное увеличение объема задействованной памяти компьютера. Если убрать комментарий, содержащийся в строке 4, то объем использованной памяти увеличиваться не будет. (В обоих случаях Flash-плеер отобразит предупреждение, в котором будет говориться о том, что сценарий замедлил производительность плеера, и спросит, хотите ли вы прервать этот сценарий. Это обычная практика при использовании слишком длинных циклов `for`, подобных приведенному ниже.)

```
1: for (i=0 ; i < Number.MAX_VALUE ; i++) {  
2:     myArray[i] = new Array(1000000000);  
3:     myArray[i] = undefined;  
4:     delete myArray[i];  
5: }
```

ИМЕНОВАННЫЕ ЭЛЕМЕНТЫ МАССИВА: АССОЦИАТИВНЫЕ МАССИВЫ

Фактически массив является слабо замаскированным объектом. Результатом является тот факт, что массивы, как и объекты, могут иметь именованные свойства. Массив с именованными свойствами является *ассоциативным*.

Назначить или извлечь именованные свойства массива можно двумя способами: с помощью строки в квадратных скобках или оператора "точка", используемого в качестве идентификатора. Два приведенных ниже примера демонстрируют процедуру определения свойства с именем «Mary» массива `myPhones`.

```
myPhones ["Mary"] = "555-555-1212";  
myPhones.Mary = "555-555-1212";
```

Аналогичным образом, используя ТОТ же синтаксис, свойство можно извлечь:

```
trace( myPhones ["Mary"] ); // 555-555-1212  
trace( myPhones.Mary ); // 555-555-1212
```

Эти два подхода для достижения оптимальных результатов можно комбинировать. Например, свойство, заданное в виде строки, можно извлечь с помощью оператора "точка". При обоих подходах имя свойства должно быть разрешенным идентификатором или выражением, которое дает в результате выполнения разрешенный идентификатор. В первом случае идентификатор указывается в виде строки, а во втором — просто как идентификатор.

Удалить именованные свойства можно с помощью оператора `delete`. Например, любое из приведенных ниже предложений удаляет свойство «Mary» и перераспределяет связанную с ним память.

```
Delete myPhones ["Mary"];  
delete myPhones.Mary;
```

Свойство `length` массива не отражает никаких именованных свойств. Массив, в который добавлено 100 именованных свойств, но не имеющих индексов, все равно будет иметь значение `length`, равное 0. Кроме того, к именованным свойствам массива не применяются методы массивов, такие как `pop()` и `push()`, речь о которых пойдет в следующем подразделе.

Доступ к именованным свойствам массива осуществляется точно так же, как и к свойствам любого объекта, — с помощью цикла `for-in`.

Совет

Цикл `for-in` рассматривался в главе 14 "Работа с данными: использование предложений".

Для объекта можно задать нумерованные свойства, как показано в следующем примере:

```
myObj = new Object();  
myObj[0] = "hello world";
```

Однако назначение нумерованных свойств не делает объект `myObj` массивом. Так, объект `myObj` все равно не имеет свойства `length` и не работает с методами класса `Array`.

МЕТОДЫ МАССИВОВ

Класс `Array` имеет 12 документированных методов: `join()`, `pop()`, `push()`, `reverse()`, `sort()`, `sortOn()`, `concat()`, `slice()`, `splice()`, `shift()`, `unshift()` и `toString()` (табл. 19.3). Кроме того, класс `Array` наследует документированные и недокументированные методы класса `Object`.

ТАБЛИЦА 19.3. МЕТОДЫ И СВОЙСТВА МАССИВОВ

Имя	МЕТОД /СВОЙ- СТВО	СИНТАКСИС	ОПИСАНИЕ
concat	Метод	<code>myArray.concat(значение0, значение1, значениеN)</code>	Соединяет аргументы и возвращает их как новый массив
join	Метод	<code>myArray.join([разделитель])</code>	Объединяет все элементы массива в строку
pop	Метод	<code>myArray.pop()</code>	Удаляет последний элемент массива и возвращает его значение
push	Метод	<code>myArray.push(значение0, значение1, ... значениеN)</code>	Добавляет один или несколько элементов в конец массива и возвращает новую длину массива
reverse	Метод	<code>myArray.reverse()</code>	Меняет направление массива на противоположное
shift	Метод	<code>myArray.shift()</code>	Удаляет первый элемент массива и возвращает его значение
slice	Метод	<code>myArray.slice(начальный_индекс, конечный_индекс)</code>	Извлекает участок массива и возвращает его как новый массив
sort	Метод	<code>myArray.sort([функция_сравнения])</code>	Сортирует массив
sortOn	Метод	<code>myArray.sortOn(имя_поля)</code>	Сортирует массив в алфавитном порядке на основе поля в массиве
splice	Метод	<code>myArray.splice(начальный_индекс, число_удаляемых_элементов, значение0, значение1, ... значениеN)</code>	Добавляет и/или удаляет элементы массива
toString	Метод	<code>myArray.toString()</code>	Возвращает строковое значение, представляющее собой элементы объекта Array
unshift	Метод	<code>myArray.unshift(значение1, значение2, ... значениеN)</code>	Добавляет один или несколько элементов в начало массива и возвращает новую длину массива
length	Свой- ство	<code>myArray.length</code>	Возвращает число элементов массива с учетом пустых или неопределенных элементов

Совет

Подробная информация о методах класса Object будет приведена ниже, в подразделе "Класс Object".

МЕТОДЫ JOIN() И toString()

Метод `join()` преобразует все элементы массива в строки и соединяет их. Практически точно так же ведет себя и метод `Array.toString()`. Однако с помощью метода `join()` можно указать символ или символы *разделителя*, используемые для разделения строк, тогда как с помощью метода `toString()` этого сделать нельзя.

Метод `join()` имеет следующий формат:

```
myArray.join(разделитель);
```

Если при использовании метода `join()` разделитель не указан, то в его качестве Flash автоматически использует символ запятой и тогда метод `join()` в точности повторяет метод `toString()`. Например:

```
myArray = new Array("John", "Matt", "Joe");  
names = myArray.join(); // names = "John,Matt,Joe"
```

Я заключил выходную информацию комментария в кавычки для того, чтобы подчеркнуть, что итоговое значение является строкой. Однако на практике итоговое значение в кавычки не заключается.

Задание символа разделителя дает возможность форматировать выходные данные. Например, если необходимо, чтобы после запятой следовал символ пробела, то в качестве разделителя можно указать два символа: запятую и пробел:

```
myArray = new Array("John", "Matt", "Joe");  
names = myArray.join(", "); // names = "John, Matt, Joe"
```

Любые вложенные массивы (массивы внутри массива) преобразовываются в строки с помощью метода `toString()`. Следовательно, при этих преобразованиях в качестве разделителя используется запятая, а не символ, указанный пользователем. Например:

```
myArray = new Array("John", "Matt", ["Peter", "Mary", "Sue"], "Joe");  
names = myArray.join("-"); // names = "John-Matt-Peter,Mary,Sue-Joe"
```

Метод `join()` не изменяет массива, к которому он применяется.

МЕТОДЫ `POP()` И `PUSH()`

Методы `pop()` и `push()` трактуют массив как *стек*, а точнее, стек, в котором добавление и удаление элементов осуществляется по принципу LIFO ("последним пришел — первым обслужен"). Представьте себе, что вы печете блинчики, по очереди снимаете со сковороды и кладете друг на друга на тарелку. Каждый раз, когда повар кладет один или несколько блинчиков в стопку, он выполняет метод `push()`. Когда вы съедаете верхний блинчик, выполняются метод `pop()`. В данном случае верхний из блинчиков "стека" является концом массива.

Оба этих метода модифицируют *существующий* массив.

Метод `push()` добавляет один или несколько элементов в конец массива и возвращает новую длину массива, как в приведенном ниже примере.

```
MyArray = new Array ("John", "Matt", "Joe") ;  
newLength = myArray.push ("Peter", "Mary", "Sue") ;  
trace(myArray); // John,Matt,Joe,Peter,Mary,Sue  
trace(newLength); // 6
```

При вызове без аргументов метод `push()` добавляет к массиву пустой элемент (значением которого является `undefined`). Эта операция эквивалентна приращению свойства `length` массива (`myArray.length++`).

Для определения значения, добавляемого в массив, можно использовать любое выражение. Выражение выполняется до того, как значение присваивается элементу, т.е. если `x` равно 3, а `y` — 4, то два следующих предложения приведут к совершенно одинаковому результату:

```
myArray.push ( x + y ) ;  
myArray.push ( 7 ) ;
```

Метод `pop()` удаляет последний элемент массива и возвращает значение удаленного элемента. Кроме того, он на единицу уменьшает свойство `length` данного массива.

МЕТОД REVERSE()

Метод `reverse()` обращает порядок элементов в массиве на противоположный. Этот метод выполняет обращение *на месте*, т.е. модифицирует существующий массив. Кроме того, он возвращает обращенный массив, как показано в приведенном ниже примере.

```
MyArray = new Array (3, 2, 1);
returnArray = myArray.reverse(); // myArray теперь такой: [1,2,3]
trace (returnArray); // 1,2,3
```

МЕТОДЫ SORT() И SORTON()

Метод `sort()` *на месте* сортирует элементы существующего массива. Он возвращает отсортированный массив. При вызове без аргументов метод `sort()` сортирует элементы массива в алфавитном порядке, при необходимости временно преобразовывая их в строки. Любые неопределенные элементы отсортировываются в конец массива.

Чтобы выполнить сортировку по какому-то другому принципу, в качестве аргумента метода `sort()` необходимо указать *функцию сравнения*. Она имеет два аргумента, представляющие собой любые два элемента массива. Это сродни вопросу о том, какой из двух заданных элементов массива должен быть указан первым. Ответить на этот вопрос нужно с помощью функции сравнения.

Функция должна вернуть отрицательное значение, если сначала в отсортированном массиве должен быть указан первый элемент. Если же сначала должен быть указан второй элемент, то функция сравнения должна вернуть положительное число. Если функция сравнения возвращает 0, это говорит о том, что порядок указания элементов не имеет значения.

В приведенном ниже примере функция сравнения `byVotes()` сортирует массив объектов `candidates` (кандидаты). Первыми будут указаны объекты, обладающие большим значением свойства `votes` (голоса). Функция сравнения возвращает положительное число, если второй объект получил больше голосов, чем первый. Фактически функция говорит: "Сначала следует указать объект с большим количеством голосов". Того же результата можно достичь путем возврата отрицательного значения в случае, если первый объект получил больше голосов, чем второй.

```
Abbey = {name: "Abbey", votes: 10};
Bob = {name: "Bob", votes: 2};
Charles = {name: "Charles", votes: 30};
Dave = {name: "Dave", votes: 100};
candidates = [Abbey, Charles, Bob, Dave];
function byVotes (a, b) {
    return b.votes - a.votes;
}
candidates.sort (byVotes);
for (i = 0; i < candidates.length; i++) {
    trace (candidates[i].name);
}
/* результат
Dave
Charles
Abbey
Bob
*/
```



Метод `sortOn()` реализует более краткое представление сортировки объектов в массиве на основе значения свойства каждого объекта. Однако в предыдущем примере этот метод работать не будет, поскольку он выполняет сортировку только в алфавитном порядке. Если же вы хотите выполнить сортировку в алфавитном порядке по имени кандидатов, то можете воспользоваться методом `sorton()`:

```
candidates.sortOn("name"); // возвращает [Abbey, Bob, Charles, Dave]
```

МЕТОД CONCAT()

Метод `concat()` добавляет в массив один или несколько элементов. Он, скорее, возвращает новый массив, чем модифицирует существующий. Пользователь должен указать значения, которые будут добавлены в качестве аргументов к методу `concat()`, как в следующем примере:

```
myArray = new Array (3, 2, 1);
myArray.concat(4,5,6); // возвращает [3,2,1,4,5,6]
```

Если один из аргументов является массивом, он будет "развернут" и затем добавлен. Следовательно, указанный ниже код приведет к такому же результату, что и предыдущий.

```
MyArray = new Array (3, 2, 1);
myArray.concat( 4, [5,6] ); // возвращает [3,2,1,4,5,6]
```

Однако вложенные массивы не "развертываются":

```
myArray.concat( [ 4, [5,6] ] ); // возвращает [3,2,1,4,[5,6] ]
```

МЕТОД SLICE()

Метод `slice()` возвращает новый массив, являющийся "срезом" (субмассивом) исходного. Он не меняет исходного массива. Метод `slice()` принимает два аргумента. Первый аргумент определяет индекс первого возвращаемого элемента, а второй — индекс, на единицу больший, чем индекс последнего возвращаемого элемента. Если указать только один аргумент, то срез будет содержать все элементы массива, начиная с элемента, соответствующего индексу, указанному в виде аргумента. Отрицательный аргумент определяет индекс относительно конца массива, начиная с -1 для последнего аргумента массива. Ниже приведено несколько примеров.

```
MyArray = [1,2,3,4,5,6];
myArray.slice(1,2); // 2
myArray.slice(1,-1); // 2,3,4,5
myArray.slice(0,3); // 1,2,3
```

МЕТОД SPLICE()

Метод `splice()` можно использовать для добавления или удаления элементов из массива. Он может в одной операции и добавлять, и удалять элементы. Данный метод, скорее, модифицирует существующий массив, чем возвращает новый. Метод `splice()` имеет такой формат:

```
myArray.splice(начальный_индекс, число_удаляемых_элементов, значение0,
значение1, ... значениеN)
```

Все аргументы, кроме первого, являются опциональными. При вызове метода `splice()` только с аргументом *начальный_индекс* будут удалены все элементы массива, начиная с элемента, имеющего этот индекс, и до конца массива. Метод возвращает удаленные элементы. Например:

```
myArray = [1,2,3,4,5,6];
trace(myArray.splice(1)); // возвращает: 2,3,4,5,6 - удаленные элементы
trace(myArray); // 1 - все, что осталось
```


Второй аргумент сообщает методу `splice()`, сколько элементов подлежит удалению. Оставшиеся элементы массива смещаются назад (в направлении к меньшим номерам индексов), заполняя образовавшиеся ниши. Например:

```
myArray = [1,2,3,4,5,6];  
trace(myArray.splice(1,2)); // 2,3 - удалены только два элемента  
trace(myArray); // 1,4,5,6 - оставшиеся четыре элемента массива
```

Остальные аргументы являются значениями элементов, подлежащих вставке в массив, начиная с элемента, следующего за элементом с начальным индексом. Существующие элементы массива сдвигаются вперед так, чтобы в массив были вставлены новые элементы. Например:

```
myArray = [1,2,3,4,5,6];  
trace(myArray.splice(1,2, "flowers","teakettles")); // 2,3  
trace(myArray); // 1,flowers,teakettles,4,5,6
```

МЕТОДЫ `UNSHIFT()` И `SHIFT()`

Методы `unshift()` и `shift()` похожи на методы `push()` и `pop()`, за исключением того, что они вставляют и удаляют элементы в *начале* массива, а не в его конце.

Метод `unshift()` вставляет один или несколько элементов в начало массива и возвращает новую длину массива. Существующие элементы массива сдвигаются вперед. Например:

```
myArray = [1,2,3,4,5,6];  
trace(myArray.unshift("flowers", "teakettles")); // 8 - новая длина  
trace(myArray); // flowers,teakettles,1,2,3,4,5,6 - новый массив
```

Метод `shift()` удаляет первый элемент массива. Существующие элементы массива сдвигаются назад, заполняя образовавшуюся нишу. Метод возвращает удаленный элемент. Например:

```
myArray = [1,2,3,4,5,6];  
trace(myArray.shift()); // 1 - возвращает удаленный элемент  
trace(myArray); // 2,3,4,5,6 - все, что осталось
```

СВОЙСТВО `LENGTH`

Свойство `length` является единственным свойством класса `Array`. Оно всегда на единицу больше индекса последнего элемента массива. Свойство `length` может не соответствовать числу элементов, содержащихся в массиве, поскольку массивы могут иметь неопределенные элементы.

Чаще всего свойство `length` используется в циклах, в которых необходимо пройти все элементы массива.

Совет

Использование свойства `length` в цикле было описано выше, в подразделе "Методы `sort()` и `sortOn()`".

ИНТЕРФЕЙСНЫЕ КЛАССЫ: `BOOLEAN`, `NUMBER` И `STRING`

Классы `Boolean`, `Number` и `String` являются "интерфейсными" классами соответствующих элементарных типов данных. Экземпляр интерфейсного класса содержит элементарную величину данных в недоступном внутреннем свойстве. Разница между элементарной величиной данных и экземпляром класса заключается в том, что экземпляр может иметь свойства и методы; и те и другие являются локальными и наследуемыми. В трех базовых интерфейсных классах все встроенные методы и свойства, за исключением одного, являются наследуемыми из объекта-прототипа класса. Единственным исключением является свойство `length` класса `string`: очевидно, что каждая строка должна иметь свое собственное свойство `length`.

В табл. 19.4 представлены методы и свойства интерфейсных классов.

ТАБЛИЦА 19.4. МЕТОДЫ И СВОЙСТВА ИНТЕРФЕЙСНЫХ КЛАССОВ: BOOLEAN, NUMBER, STRING

Имя	Метод/ свойство	ФОРМАТ	ОПИСАНИЕ
<code>Boolean.toString</code>	Метод	<code>myBoolean.toString()</code>	Возвращает строковое представление (<code>true</code>) или (<code>false</code>) объекта <code>myBoolean</code>
<code>Boolean.valueOf</code>	Метод	<code>meBoolean.valueOf()</code>	Возвращает значение элементарного типа объекта <code>myBoolean</code>
<code>Number.toString</code>	Метод	<code>myNumber.toString</code>	Возвращает строковое представление объекта <code>myNumber</code>
<code>Number.valueOf</code>	Метод	<code>myNumber.valueOf()</code>	Возвращает элементарное значение объекта <code>myNumber</code>
<code>Number.MAX_VALUE</code>	Свойство	<code>myNumber.MAX_VALUE</code>	Константа, максимальное число, которое можно представить в ActionScript: 1.79769313486231e+308
<code>Number.MIN_VALUE</code>	Свойство	<code>myNumber.MIN_VALUE</code>	Константа, наименьшее положительное число, которое можно представить в ActionScript: 5e-324
<code>Number.NaN</code>	Свойство	<code>myNumber.NaN</code>	"Не число"; константа, представляющая значение, не являющееся числом, но которое используется там, где ожидается число
<code>Number.NEGATIVE_INFINITY</code>	Свойство	<code>myNumber.NEGATIVE_INFINITY</code>	Константа, представляющая собой значение, меньшее, чем <code>Number.MAX_VALUE</code>
<code>Number.POSITIVE_INFINITY</code>	СВОЙСТВО	<code>myNumber.POSITIVE_INFINITY</code>	Константа, представляющая собой значение, большее, чем <code>Number.MAX_VALUE</code> . (То же самое, что и глобальная константа <code>Infinity</code> .)
<code>String.charAt</code>	Метод	<code>myString.charAt (индекс)</code>	Возвращает символ в определенном месте строки <code>myString</code>
<code>String.charCodeAt At</code>	Метод	<code>myString.charCodeAt (индекс)</code>	Возвращает кодировку Unicode символа в объекте <code>myString [индекс]</code> в виде шестнадцатеричного целого числа в диапазоне от 0 до 65,535

Имя	Метод/ свойство	ФОРМАТ	ОПИСАНИЕ
<code>String.concat</code>	Метод	<code>myString.concat (значение1, ... значениеN)</code>	Возвращает новую строку, образованную в результате слияния (объединения) строки <code>myString</code> и строк, указанных в аргументах (<i>значение 1, ... значениеN</i>)
<code>String.fromCharCode</code>	Метод	<code>String.fromCharCode (код_символа1, код_символа2, ... код_символаN)</code>	Возвращает новую строку, составленную из символов, коды которых указаны в качестве аргументов
<code>String.indexOf</code>	Метод	<code>myString.indexOf (подстрока, [начальный_индекс])</code>	Выполняет поиск строки, начиная с начального индекса, и возвращает индекс первого экземпляра подстроки. Возвращает -1, если подстрока не найдена
<code>String.lastIndexOf</code>	Метод	<code>myString.lastIndexOf (подстрока, [начальный_индекс])</code>	Выполняет поиск строки, начиная с начального индекса, и возвращает индекс последнего экземпляра подстроки. Возвращает -1, если подстрока не найдена
<code>String.slice</code>	Метод	<code>myString.slice (начальный_индекс, [конечный_индекс])</code>	Возвращает подстроку исходной строки, начиная с символа, расположение которого в строке <code>myString</code> соответствует начальному индексу, и до (но не включая) символа, расположение которого в строке <code>myString</code> соответствует конечному индексу. (Индексом первого символа строки является 0.)
<code>String.split</code>	Метод	<code>myString.split (["разделитель"], [limit])</code>	Разделяет строку на подстроки, разбивая ее в местах расположения <i>разделителя</i> , и возвращает подстроки в виде массива. Если в качестве разделителя используется пустая строка (" "), то каждый символ строки будет элементом возвращенного массива. Если разделитель не определен, то первым и единственным элементом возвращенного массива будет целая строка. Опциональный аргумент limit определяет максимальное число подстрок, которые могут быть возвращены

Имя	Метод/ свойство	ФОРМАТ	ОПИСАНИЕ
<code>String.substr</code>	Метод	<code>myString.substr</code> (<i>начальный_индекс</i> , <i>[длина]</i>)	Возвращает подстроку, которая начинается с начального индекса и включает в себя число символов, указанных в аргументе <i>длина</i> , или остальные символы строки (если аргумент <i>длина</i> отсутствует). Если начальный индекс отрицательный, то начальное положение определяется от конца строки, при этом -1 обозначает последний символ, -2 — предпоследний символ и т.д.
<code>String.substring</code>	Метод	<code>myString.substring</code> (<i>начальный_индекс</i> , <i>конечный_индекс</i>)	Возвращает подстроку, начиная с начального индекса и заканчивая символом перед конечным индексом. Если аргумент <i>конечный_индекс</i> опущен, то возвращаемая подстрока будет длиться до конца строки. Если начальный индекс равен конечному, то метод возвращает пустую строку. Если начальный индекс больше конечного, то интерпретирующая программа Flash должна поменять их местами перед выполнением функции; на практике иногда возвращается пустая строка
<code>String.toLowerCase</code>	Метод	<code>myString.toLowerCase()</code>	Возвращает копию строки <code>myString</code> , все символы которой являются строчными. Исходное значение не изменяется
<code>String.toUpperCase</code>	Метод	<code>myString.toUpperCase()</code>	Возвращает копию строки <code>myString</code> , все символы которой являются прописными. Исходное значение не изменяется
<code>String.valueOf</code>	Метод	<code>myString.valueOf()</code>	Возвращает элементарное значение указанного объекта <code>String</code>
<code>String.length</code>	Свойство	<code>myString.length</code>	Длина строки

КЛАСС BOOLEAN

Класс Boolean (булевы величины) — самый простой в ActionScript. Он имеет только два свойства, `_proto_` и `constructor`, которыми обладает каждый создаваемый объект. (Мы не будем тратить время и упоминать эти свойства при рассмотрении объектов других классов.)

Совет

Со свойством `_proto_` мы познакомились в главе 15 "Объединение предложений в функции", а свойство `constructor` будет рассмотрено далее в этой главе, в подразделе "Класс Object". Подробная информация об обоих свойствах будет представлена в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

Класс Boolean имеет всего два метода, которые также присущи всем объектам.

- Метод `toString()` возвращает в виде строки булево значение (`true` или `false`).
- Метод `valueOf()` возвращает элементарную булеву величину, позволяющую сравнить значения, содержащиеся в объектах.



Для сравнения булевых объектов в версии Flash 5 метод `valueOf()` был не нужен, поскольку сравнение самих объектов было эквивалентно сравнению их значений. В версии Flash MX эта методика больше не работает.

Совет

Подробная информация о сравнении объектов представлена в приведенной ниже врезке.

Сравнение объектов



Во Flash MX два объекта нельзя оценивать как равные только потому, что они имеют одинаковые свойства с одинаковыми значениями. По умолчанию, два обращения к объектам считаются равными только в случае, если они относятся к одному и тому же объекту.

В этом смысле версия Flash 5 вела себя непоследовательно, а Flash MX более строго следует стандарту ECMA-262, выполняя сравнение объектов по ссылке, а не по значению. В приведенном ниже примере во Flash MX выполняется сравнение по ссылке, тогда как во Flash 5 оно выполняется по значению. В итоге мы получим разные результаты сравнения:

```
a = new Number(1);  
b = new Number(1);  
trace( a == b ); // ложно во Flash MX, истинно во Flash 5
```

Для сравнения содержимого объектов классов Boolean, Number и string во Flash MX необходимо использовать метод `valueOf()`:

```
trace( a.valueOf() == b.valueOf() ); // истинно
```

При работе с другими объектами для получения значения при сравнении может понадобиться другой метод. Например, для сравнения цветов, хранящихся в двух экземплярах класса Color, необходимо использовать метод `getRGB()`.

Совет

Подробно о свойстве `valueOf()` и сравнениях рассказывается в главе 13 "Использование операторов".

Совет

Подробная информация об отличиях между способами сравнения "по ссылке" и "по значению" приведена в главе 15 "Объединение предложений в функции".

Необходимость в использовании метода `toString()` возникает достаточно редко, поскольку обычно, если это нужно, в **ActionScript** выполняется автоматическое преобразование данных в строку. Этот метод можно применить для принудительного использования строк или для того, чтобы быть абсолютно уверенным в том, что вы имеете дело со строками. Однако ту же самую задачу выполняет и глобальная функция преобразования `string()`, которая к тому же позволяет использовать один и тот же формат как для объектов, так и для элементарных данных.

Совет

О методах `toString()` и `valueOf()` рассказывается ниже, в подразделе "Класс `Object`".

Совет

Подробная информация о глобальных функциях преобразования `string()`, `Number()` и `Boolean()` приведена в главе 12 "Управление переменными, данными и типами данных".

КЛАСС NUMBER

Класс `Number` имеет методы `toString()` и `valueOf()`, которые соответственно возвращают строковый эквивалент числа и элементарную числовую величину.

Кроме того, класс `Number` имеет пять констант: `NaN` ("не число"), `Number.MAX_VALUE`, `Number.MIN_VALUE`, `Number.POSITIVE_INFINITY` и `Number.NEGATIVE_INFINITY`.

Совет

Константы класса `Number` рассматривались в главе 12 "Управление переменными, данными и типами данных".

КЛАСС STRING

Класс `String` имеет метод `valueOf()`, который возвращает строковый примитив. Каждый член класса `string` имеет локальное свойство `length` (длина), предназначенное только для чтения. Кроме того, класс `String` имеет множество методов управления строкой, которые являются наиболее распространенными в **ActionScript**. Наряду с этим они больше всего задействуют ресурсы процессора, что побуждает программистов писать альтернативные оптимизированные методы. В целом строковые методы во **Flash MX** работают быстрее, чем во **Flash 5**.

Строковые методы можно разделить на четыре категории: поиск символа или последовательности символов, извлечение части или частей строки, преобразование в строчные или прописные символы и генерация символов из их кодов и наоборот.



Метод `String.split()`, разбивающий строки на подстроки, во **Flash MX** был приведен в соответствие стандарту **ECMA-262** путем добавления опционального второго аргумента *limit*, указывающего максимальное число подстрок. Кроме того, когда первый аргумент, *разделитель*, является пустой строкой, метод `String.split()` будет возвращать массив, каждый элемент которого будет строкой. Во **Flash 5** в такой же ситуации возвращался массив, в котором содержался только один элемент, представляющий собой целую строку.

Приведенный ниже код демонстрирует отличие применения данного метода во **Flash MX** и во **Flash 5**.

```
MyString = "Joe";  
i = myString.split("");  
trace (i); // J,o,e во Flash MX, Joe во Flash 5  
trace (i.length); // 3 во Flash MX, 1 во Flash 5
```

Метод `String.substring()` возвращает подстроку, начиная с символа, соответствующего начальному индексу и заканчивая символом, находящимся перед конечным индексом. В плеере

Rash 6 при выполнении этого метода обнаружился сбой. Так, если начальный индекс больше конечного, то интерпретирующая программа Flash перед выполнением функции должна поменять их местами. Однако в реальности иногда возвращается пустая строка. Это может происходить и постоянно, например, в случае, когда начальный индекс равен длине строки. В приведенном ниже примере subs должно быть равно `two12`, а фактически возвращается пустая строка.

```
Str = "onetwo12"
subs = str.substring(8,3);
```

Этот дефект можно компенсировать, вставив в программу приведенный ниже код в точку, предшествующей методу `String.substring()`.

```
String.prototype.substring = function (from, to) {
    var temp = from;
    if (to == undefined) to = this.length;
    if (from == to) return "";
    if (from > to) {
        from = to;
        to = temp - from;
    }
    else to -= from;
    return this.substr (from, to);
};
```

Описанный выше дефект в последующих версиях Rash-плеера может быть исправлен.

КЛАСС DATE

Класс Date позволяет определить текущее время и дату и хранить полученные значения в объектах. Все значения времени и дат в **ActionScript** хранятся в виде одного числа: количества миллисекунд до или после полуночи 1 января 1970 года универсального синхронизированного времени (среднего времени по Гринвичу). Это число возвращается при выполнении метода `Date.valueOf()`.

Методы класса Date можно использовать для получения и установки даты/времени в виде одного объекта либо для получения и установки года, **месяца**, даты, дня, часа, минуты, секунды и миллисекунды независимо, в виде локального или универсального синхронизированного времени (UTC). Разницу (в минутах) между универсальным синхронизированным и локальным временем можно получить с помощью метода `getTimezoneOffset()`.

Класс Date имеет также метод `toString()`, который возвращает полную информацию о дате в удобном для считывания формате, например `Sun Jun 16 18:04:58 GMT -0600 2003`. Ту же самую задачу можно выполнить с помощью глобальной функции `Date()`. Таким образом, выражение `myDate.toString()` даст тот же самый результат, что и `Date(myDate)`. Ни то ни другое выражение в связи с автоматическим преобразованием типов данных не применяются слишком часто.

Методы класса Date представлены в табл. 19.5.

ТАБЛИЦА 19.5. МЕТОДЫ КЛАССА DATE

Имя	ФОРМАТ	ОПИСАНИЕ
<code>Date.getDate</code>	<code>myDate.getDate()</code>	Возвращает день месяца от 1 до 31 в соответствии с локальным временем
<code>Date.getDay</code>	<code>myDate.getDay()</code>	Возвращает день недели от 0 (воскресенье) до 6 (суббота) в соответствии с локальным временем

Имя	ФОРМАТ	ОПИСАНИЕ
<code>Date.getFullYear</code>	<code>myDate.getFullYear()</code>	Возвращает год, представленный в виде четырех цифр, в соответствии с локальным временем
<code>Date.getHours</code>	<code>myDate.getHours()</code>	Возвращает час от 0 (полночь) до 23 в соответствии с локальным временем
<code>Date.getMilliseconds</code>	<code>myDate.getMilliseconds()</code>	Возвращает миллисекунды от 0 до 999 в соответствии с локальным временем
<code>Date.getMinutes</code>	<code>myDate.getMinutes()</code>	Возвращает минуты от 0 до 59 в соответствии с локальным временем
<code>Date.getMonth</code>	<code>myDate.getMonth()</code>	Возвращает месяц от 0 (январь) до 11 (декабрь) в соответствии с локальным временем
<code>Date.getSeconds</code>	<code>myDate.getSeconds()</code>	Возвращает секунды от 0 до 59 в соответствии с локальным временем
<code>Date.getTime</code>	<code>myDate.getTime()</code>	Возвращает число миллисекунд, прошедших после 1 января 1970 года универсального времени
<code>Date.getTimezoneOffset</code>	<code>myDate.getTimezoneOffset()</code>	Возвращает разницу в минутах между локальным временем, установленным на компьютере, и универсальным временем
<code>Date.getUTCDate</code>	<code>myDate.getUTCDate()</code>	Возвращает день месяца от 1 до 31 в соответствии с универсальным временем
<code>Date.getUTCDay</code>	<code>myDate.getUTCDay()</code>	Возвращает день недели от 0 (воскресенье) до 6 (суббота) в соответствии с универсальным временем
<code>Date.getUTCFullYear</code>	<code>myDate.getUTCFullYear()</code>	Возвращает год, представленный в виде четырех цифр в соответствии с универсальным временем
<code>Date.getUTCHours</code>	<code>myDate.getUTCHours()</code>	Возвращает час от 0 (полночь) до 23 в соответствии с универсальным временем
<code>Date.getUTCMilliseconds</code>	<code>myDate.getUTCMilliseconds()</code>	Возвращает миллисекунды от 0 до 999 в соответствии с универсальным временем
<code>Date.getUTCMinutes</code>	<code>myDate.getUTCMinutes()</code>	Возвращает минуты от 0 до 59 в соответствии с универсальным временем
<code>Date.getUTCMonth</code>	<code>myDate.getUTCMonth()</code>	Возвращает месяц от 0 (январь) до 11 (декабрь) в соответствии с универсальным временем

Имя	ФОРМАТ	ОПИСАНИЕ
Date.getUTCSeconds	myDate.getUTCSeconds()	Возвращает секунды от 0 до 59 в соответствии с универсальным временем
Date.getYear	myDate.getYear()	Возвращает целое число, которое, будучи добавленным к 1900, даст полное значение года в соответствии с локальным временем. Например, 0 означает 1900 год
Date.setDate	myDate.setDate(<i>дата</i>)	Устанавливает день месяца в значение да та от 0 до 31 в соответствии с локальным временем. Возвращает новое время в миллисекундах
Date.setFullYear	myDate.setFullYear(<i>год</i> [, <i>месяц</i> [, <i>дата</i>]])	Устанавливает с помощью четырех цифр <i>год</i> и (опционально) <i>месяц</i> , от 0 (январь) до 11 (декабрь), и <i>дату</i> — от 1 до 31 (все параметры устанавливаются в соответствии с локальным временем). Возвращает новое время в миллисекундах
Date.setHours	myDate.setHours(<i>час</i> [, <i>минута</i> [, <i>секунда</i> [, <i>миллисекунда</i>]]])	Устанавливает час от 0 (полночь) до 23 и (опционально) <i>минуты</i> — от 0 до 59, <i>секунды</i> — от 0 до 59 и <i>миллисекунды</i> — от 0 до 999 (все параметры устанавливаются в соответствии с локальным временем). Возвращает новое время в миллисекундах
Date.setMilliseconds	myDate.setMilliseconds(<i>миллисекунды</i>)	Устанавливает <i>миллисекунды</i> от 0 до 999 в соответствии с локальным временем. Возвращает новое время в миллисекундах
Date.setMinutes	myDate.setMinutes(<i>минуты</i> [, <i>секунды</i> [, <i>миллисекунды</i>]])	Устанавливает <i>минуты</i> от 0 до 59, а также (опционально) <i>секунды</i> — от 0 до 59 и <i>миллисекунды</i> — от 0 до 999 (все параметры устанавливаются в соответствии с локальным временем). Возвращает новое время в миллисекундах
Date.setMonth	myDate.setMonth(<i>месяц</i> [, <i>дата</i>])	Устанавливает <i>месяц</i> от 0 (январь) до 11 (декабрь), а также (опционально) <i>дату</i> — от 1 до 31 (оба параметра устанавливаются в соответствии с локальным временем). Возвращает новое время в миллисекундах

Имя	ФОРМАТ	ОПИСАНИЕ
Date.setSeconds	myDate.setSeconds (секунды [, миллисекунды])	Устанавливает секунды от 0 до 59 и (опционально) <i>миллисекунды</i> — от 0 до 999 (оба параметра устанавливаются в соответствии с локальным временем). Возвращает новое время в миллисекундах
Date.setTime	myDate.setTime (миллисекунды)	Устанавливает время в миллисекундах, прошедшее после полудни 1 января 1970 года. Возвращает то же значение, которое было задано
Date.setUTCDate	myDate.setUTCDate(дата)	Устанавливает <i>дату</i> в соответствии с универсальным временем — от 1 до 31. Возвращает новое время в миллисекундах
Date.setUTCFullYear	myDate.setUTCFullYear (год [, месяц [, дата]])	Устанавливает РОД, состоящий из четырех цифр, и (опционально) <i>месяц</i> — от 0 (январь) до 11 (<i>декабрь</i>), а также <i>дату</i> — от 1 до 31 (все параметры устанавливаются в соответствии с универсальным временем)
Date.setUTCHours	myDate.setUTCHours (час [, минуты [, секунды [, миллисекунды]]])	Устанавливает час от 0 (полночь) до 23 и (опционально) <i>минуты</i> — от 0 до 59, <i>секунды</i> — от 0 до 59 и <i>миллисекунды</i> — от 0 до 999 (все параметры устанавливаются в соответствии с универсальным временем). Возвращает новое время в миллисекундах
Date.setUTCMilliseconds	myDate.setUTCMilliseconds(миллисекунды)	Устанавливает <i>миллисекунды</i> от 0 до 999 в соответствии с универсальным временем. Возвращает новое время в миллисекундах
Date.setUTCMinutes	myDate.setUTCMinutes (минуты [, секунды [, миллисекунды]])	Устанавливает <i>минуты</i> от 0 до 59, а также (опционально) <i>секунды</i> — от 0 до 59 и <i>миллисекунды</i> — от 0 до 999 (все параметры устанавливаются в соответствии с универсальным временем). Возвращает новое время в миллисекундах

Имя	ФОРМАТ	ОПИСАНИЕ
<code>Date.setUTCMonth</code>	<code>myDate.setUTCMonth</code> (<i>месяц</i> [, <i>дата</i>])	Устанавливает месяц от 0 (январь) до 11 (декабрь), а также (опционально) <i>дату</i> — от 1 до 31 (оба параметра устанавливаются в соответствии с универсальным временем). Возвращает новое время в миллисекундах
<code>Date.setUTCSeconds</code>	<code>myDate.setUTCSeconds</code> (<i>секунды</i> [, <i>миллисекунды</i>])	Устанавливает секунда от 0 до 59 и (опционально) <i>миллисекунды</i> — от 0 до 999 (оба параметра устанавливаются в соответствии с универсальным временем). Возвращает новое время в миллисекундах
<code>Date.setYear</code>	<code>myDate.setYear(год)</code>	Определяет значение, которое будет извлечено при выполнении метода <code>Date.getYear()</code> . Аргумент <i>год</i> является положительным целым числом. Если <i>год</i> представлен в виде одно- или двухзначного числа, то при выполнении метода <code>Date.getYear()</code> будет извлечено это число. Если <i>год</i> представлен в виде трех- или четырехзначного числа, то при выполнении метода <code>Date.getYear()</code> будет извлечена разность между этим числом и 1900
<code>Date.toString</code>	<code>myDate.toString()</code>	Возвращает строку, представляющую дату и время в следующем формате: Sat May 4 12:42:19 GMT-0700 2003
<code>Date.UTC</code>	<code>Date.UTC</code> (<i>год</i> , <i>месяц</i> [, <i>дата</i> [, <i>час</i> [, <i>минуты</i> [, <i>секунды</i> [, <i>миллисекунды</i>]]]]])	Возвращает число миллисекунд, прошедших с полуночи 1 января 1970 года универсального времени, а также дату/время, определяемые аргументами <i>месяц</i> , <i>дата</i> , <i>час</i> , <i>минуты</i> , <i>секунды</i> и <i>миллисекунды</i>

Конструктор `Date` можно использовать тремя способами.

- `myDate = new Date();` — устанавливает `myDate` в значение текущей даты и времени.
- `myDate = new Date(1000000);` — устанавливает `myDate` в значение даты и времени в один миллион миллисекунд после полуночи 1 января 1970 года.
- `myDate = new Date(2003, 1, 2, 3, 4, 5, 6);` — устанавливает `myDate` в значение 2 января 2003 года, 3:04 (утра) плюс 5 секунд и 6 миллисекунд, т.е. формат будет таким: `new Date (год, месяц, дата, час, минуты, секунды, миллисекунды);`

В последнем формате все параметры, за исключением *года* и *месяца*, являются опциональными, и если они не **предоставляются**, то устанавливаются в значение 0. Часы устанавливаются от 0 до 23, где 0 соответствует полуночи, а 12 — полудню. Для 1900-1999 гг. значения можно устанавливать от 0 до 99, а для всех последующих годов обязательно нужно указывать четыре цифры.

КЛАСС FUNCTION



Класс Function во Flash MX является новым. Он вносит функции в систему классов и открывает новые возможности, например, добавление пользовательских функций к объекту `Function.prototype` или создание подклассов функций. Поскольку в **ActionScript** классы реализуются в виде функций, то, кроме того, объект `Function.prototype` является логическим местом размещения методов, которые могут выполняться в любом классе. В качестве примера можно привести метод `extends()`, который устанавливает наследование между дочерним и родительским классами.

Совет

Пример создания метода `extends()` в объекте `Function.prototype` приведен в главе 21 "Упаковка данных и функций с помощью пользовательских объектов".

Класс Function имеет всего два встроенных метода: `apply()` и `call()`.

Класс Function является "неконструктивным" конструктором: пригодные к употреблению новые экземпляры с помощью ключевого слова `new` создать нельзя.

Совет

Методы `apply()` и `call()`, а также синтаксис создания функций описаны в главе 15 "Объединение предложений в функции".

КЛАСС OBJECT

Класс Object в ActionScript является основой всех остальных классов. Все другие классы наследуют класс Object и, следовательно, имеют общие с ним свойства и методы.

Одним из скрытых достоинств Flash MX является возможность увеличения свойств класса Object с 3 до 10. Чтобы "отобразить" и увидеть эти свойства, воспользуйтесь недокументированной функцией `ASSetPropFlags()` класса Object:

```
ASSetPropFlags(Object.prototype, null, 8, 1);  
for (a in Object.prototype) trace (a);
```

Данный цикл `for-in` позволяет извлечь 10 свойств: `constructor`, `isPrototypeOf()`, `hasOwnProperty()`, `toLocaleString()`, `toString()`, `valueOf()`, `addProperty()`, `unwatch()` и `watch()`. Каждый объект, созданный с помощью оператора `new`, автоматически наследует эти свойства, причем 9 последних из них являются методами. Свойства класса Object представлены в табл. 19.6.

Первые пять свойств, представленные в этой таблице, являются недокументированными. Они определены в стандарте ECMA-262, хотя в нем используется не свойство `isPrototypeOf()`, а `propertyIsEnumerable()`. Тем не менее необходимо иметь в виду, что поведение любой недокументированной функции в последующих версиях Flash может быть изменено. Теоретически они даже могут вообще не поддерживаться.

ТАБЛИЦА 19.6. МЕТОДЫ И СВОЙСТВА КЛАССА OBJECT

ИМЯ	МЕТОД/ СВОЙСТВО	ФОРМАТ	ОПИСАНИЕ
<code>addProperty</code>	Метод	<code>myObject.addProperty("myProp", getMyProp, setMyProp)</code>	Создает свойство получения/установки с именем <code>myProp</code>
<code>constructor</code>	Свойство	<code>myObject.constructor</code>	Указывает функцию-конструктор объекта <code>myObject</code>
<code>hasOwnProperty</code>	Метод	<code>myObject.hasOwnProperty("myProp")</code>	Возвращает значение <code>true</code> , если объект <code>myObject</code> имеет локальное свойство с именем <code>myProp</code> ; в противном случае возвращает значение <code>false</code>
<code>isPrototypeOf</code>	Метод	<code>isPropertyEnumerable(свойство)</code>	Возвращает значение <code>true</code> , если аргумент <i>свойство</i> является локальным свойством, перечисляемым в цикле <code>for-in</code>
<code>isPrototypeOf</code>	Метод	<code>protoObject.isPrototypeOf(экземпляр_объекта)</code>	Возвращает значение <code>true</code> , если аргумент <i>экземпляр_объекта</i> использует <code>protoObject</code> в качестве объекта-прототипа, где он получает коллективные свойства; в противном случае возвращает значение <code>false</code>
<code>registerClass</code>	Метод	<code>Object.registerClass</code>	Предварительно регистрирует видеоклип членом класса (см. главу 17 "Демонстрация мощи видеоклипов")
<code>toLocaleString</code>	Метод	<code>myObject.toLocaleString()</code>	Возвращает строку, соответствующую определенной стране или региону; по умолчанию возвращает строку <code>[object Object]</code>
<code>toString</code>	Метод	<code>myObject.toString()</code>	Возвращает строку <code>[object Object]</code>
<code>unwatch</code>	Метод	См. табл. 19.8	Удаляет регистрацию, созданную методом <code>watch()</code>
<code>valueOf</code>	Метод	<code>myObject.valueOf()</code>	В классе <code>object</code> по умолчанию возвращает сам объект. В дочерних классах, таких как <code>Boolean</code> , <code>Date</code> , <code>Number</code> и <code>String</code> , могут быть определены более полезные методы <code>valueOf()</code>
<code>watch</code>	Метод	См. табл. 19.8	Регистрирует функцию обратного вызова, которую следует активизировать при изменении указанного свойства объекта

Во Flash 5 класс `Object` имеет только три свойства: `toString()`, `valueOf()` и `constructor`. Если во Flash 5 их необходимо перечислить с помощью цикла `for-in`, то формат отображения скрытых свойств будет таким:

```
ASSetPropFlags(Object.prototype, null, 2);
```

Из 10 свойств класса `Object` 7 определены стандартом `ECMA-262` и имеются также в `JavaScript`. (И в `ECMA-262`, и в `JavaScript` определено свойство `propertyIsEnumerable()`, а не `isPropertyEnumerable()`.) Три свойства присущи только Flash: `addProperty()`, `unwatch()` и `watch()`.

СВОЙСТВО CONSTRUCTOR

Свойство `constructor` предназначено для того, чтобы указывать функцию-конструктор данного объекта. Следовательно, свойство `constructor` можно использовать для определения класса объекта:

```
raySound = new Sound();
if ((typeof mySound == "object") && (mySound.constructor == Sound))
    // затем сделайте что-нибудь с объектом Sound
```

Каждый экземпляр класса обладает собственным свойством `constructor`. Объект-прототип `prototype` класса (как и любой другой объект) также обладает свойством `constructor`.

Совет

Подробная информация о свойстве `constructor` приведена в главе 21 "Упаковка данных и функций с помощью пользовательских объектов". В этой же главе упоминается недокументированное свойство `_constructor_` (с символами подчеркивания справа и слева).

МЕТОД toString()

Метод `toString()` предназначен для возвращения строки, каким-либо образом представляющей объект. Интерпретирующая программа `ActionScript` активизирует этот метод, когда объект необходимо преобразовать в строку. Некоторые встроенные объекты обладают характерным для их класса методом `toString()`. Такие объекты перечислены в табл. 19.7.

ТАБЛИЦА 19.7. ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ МЕТОДОВ `toString()` ВСТРОЕННЫХ ОБЪЕКТОВ

КЛАСС	ЗНАЧЕНИЕ, ВОЗВРАЩАЕМОЕ МЕТОДОМ <code>toString()</code>
Array	Разделенный запятыми перечень элементов массива, преобразованных в строки
Boolean	Значение <code>true</code> или <code>false</code> в виде строки
Date	Полная дата и время в формате <code>Sun Jun 16 18:04:58 GMT-0600 2002(*)</code>
Button, MovieClip, TextField	Абсолютный путь к экземпляру
Number	Число в виде строки
Object	Строка <code>[object Object]</code>
Объект XML-узла или XML-документ	Исходный XML-код

* GMT означает среднее время по Гринвичу (то же самое, что и универсальное синхронизированное время).

Классы, не обладающие характерными только для них методами `toString()`, используют метод, который они унаследовали от класса `Object` (возвращающий строку `[object Object]`). Чтобы получить возвращаемое значение, несущее большую смысловую нагрузку, для таких классов можно задать специальный метод `toString()`, как в приведенном ниже примере.

```
Sound.prototype.toString = function () {return "Sound";};
```

При создании пользовательского класса для него можно создать и метод `toString()`, возвращающий данные, которые можно использовать либо для отладки, либо просто в программе.

МЕТОД `toLocaleString()`

Метод `toLocaleString()` является альтернативой методу `toString()` — его версией для определенной реализации. Предназначением данного метода является *обеспечение локализованной* версии строки, представляющей объект. Локализацией называется адаптация к различным странам и языкам. Метод `toLocaleString()` прототипа класса `Object` возвращает то же самое, что и метод `toString()`. Однако методы `toLocaleString()` можно определять для встроенных классов. Метод `toLocaleString()` в `ActionScript` не документирован.

МЕТОД `isPropertyEnumerable()`

Метод `isPropertyEnumerable()` принимает один аргумент, являющийся свойством объекта, и возвращает значение `true`, если это свойство перечисляется в цикле `for-in`. Если свойство не существует или не перечислено в цикле `for-in`, метод `isPropertyEnumerable()` возвращает значение `false`.

Метод `isPropertyEnumerable()` не учитывает свойства в цепочке прототипа. Это может привести к результатам, которые не будут интуитивно понятными. Например, в приведенном ниже коде перечисляются свойства нового экземпляра класса `TextField`, а затем для каждого свойства вызывается метод `isPropertyEnumerable()`. Можно посчитать, что поскольку свойства получены с помощью цикла `for-in`, то с помощью этого цикла они также должны быть перечислены. На первый взгляд, вы правы, но метод `isPropertyEnumerable()` все равно возвращает значение `false`, поскольку он не учитывает свойства в цепочке прототипа, а все свойства нового экземпляра класса `TextField` находятся в коллективном использовании с объектом `TextField.prototype`. При вызове метода `isPropertyEnumerable()` для каждого свойства непосредственно в объекте `TextField.prototype` метод `isPropertyEnumerable()` возвращает значение `true`.

```
myTextField = new TextField();
for (a in myTextField) {
    trace(a);
    trace(myTextField.isPropertyEnumerable(a));
    trace(TextField.prototype.isPropertyEnumerable(a));
}
/* Результат
condenseWhite
false
true
```

```
restrict  
false  
true  
textHeight  
false  
true  
и т.д.  
*/
```

В ActionScript метод `isPropertyEnumerable()` недокументирован. Он эквивалентен свойству `propertyIsEnumerable()` стандарта ECMA-262 и JavaScript.

МЕТОД ISPROTOOF()

Метод `isProtoOf()` принимает один аргумент, являющийся объектом. Формат метода следующий:

```
protoObject.isPrototypeOf(instanceObject);
```

Если экземпляр объекта *instanceObject* использует *protoObject* в качестве прототипа объекта, у которого он получает коллективные свойства, то метод `isProtoOf()` возвращает значение `true`. В противном случае возвращается значение `false`.

Метод `isProtoOf()` зависит от свойства `_proto_` экземпляра объекта для поиска прототипа экземпляра объекта. Например, в приведенном ниже коде используется текстовое поле из предыдущего примера. Можно заметить, что если вы повторно укажете свойство `_proto_` экземпляра объекта так, что оно больше не будет указывать на прототип класса, то метод `isProtoOf()` корректно работать не будет:

```
trace(TextField.prototype.isPrototypeOf(myTextField)); // true  
myTextField.__proto__= null;  
trace(TextField.prototype.isPrototypeOf(myTextField)); // false
```

МЕТОД VALUEOF()

По аналогии с тем, что метод `toString()` представляет объект в виде строки, метод `valueOf()` представляет объект как другой элементарный тип данных. По умолчанию метод `valueOf()` класса `Object` возвращает сам объект. Например:

```
trace(myTextField.valueOf() == myTextField); // true  
trace(typeof myTextField.valueOf() == typeof myTextField); // true  
trace(typeof myTextField.valueOf()); // объект
```

Однако в других классах метод `valueOf()` может быть более полезным. Так, класс `Date` имеет метод `valueOf()`, который возвращает число миллисекунд между временем объекта `Date` и полночью 1 января 1970 года. Если дата предшествует этому значению, возвращаемое число будет отрицательным. (Этот метод эквивалентен методу `Date.getTime()`.)

```
myDate = new Date();  
trace (myDate.valueOf()); // 1024275924859  
trace (myDate); // Sun Jun 16 18:05:24 GMT-0700 2002
```

Для классов `Boolean`, `Number` и `String` определен метод `valueOf()`, который возвращает примитивный элемент данных, что позволяет выполнять сравнение элементарных значений.

Совет

Подробная информация о сравнении объектов приведена выше, во врезке "Сравнение объектов".

МЕТОД HASOWNPROPERTY()

Недокументированный метод `hasOwnProperty()` принимает один строковый аргумент, указывающий имя возможного локального свойства. Метод возвращает значение `true`, если объект имеет локальное свойство с этим именем. В противном случае он возвращает значение `false`. Метод имеет такой формат:

```
myTextField.hasOwnProperty("myProp")
```

В приведенном ниже примере создается глобальная функция `findValue()`, в которой метод `hasOwnProperty()` используется для определения, имеет ли объект или любой из прототипов, которые он наследует, метод `valueOf()`. Метод `findValueOf()` возвращает число, указывающее, сколько уровней следует пройти вверх по цепочке прототипов для того, чтобы найти метод `valueOf()`. Если возвращается значение 0, это значит, что метод `valueOf()` имеет сам объект. Если возвращается значение 1, это означает, что метод `valueOf()` имеет родительский объект, и т.д. **Строка 7** проверяет, был ли найден `valueOf()` в классе `Object`. Если это так, то метод `findValue()` возвращает значение -1 (отрицательное число).

Если цепочка прототипов пройдена, а метод `valueOf()` не найден, то результат в строке 5 будет неопределенным, цикл `while` завершается (строка 4), функция прерывается и возвращает значение `undefined` (строка 15). Это может произойти, к примеру, если цепочка прототипов была разбита прежде, чем достигнут класс `Object`.

```
1: _global.findValueOf = function (obj) {
2:   var i = 0;
3:   var result = false;
4:   while (result !== undefined) {
5:     result = obj.hasOwnProperty("valueOf");
6:     if (result === true) {
7:       if (obj === Object.prototype) return (-1);
8:       else return i;
9:     }
10:    else {
11:      obj = obj.__proto__; // один шаг вверх по цепочке
                           //прототипов
12:      i++;
13:    }
14:  }
15:  return;
16: }
17:
18: createEmptyMovieClip("myClip",1);
19: trace (findValueOf(myClip)); // -1, valueOf в Object.prototype
20: myDate = new Date();
21: trace (findValueOf(myDate)); // 1, valueOf в родительском классе
```

МЕТОД ADDPROPERTY()

Метод `addProperty()` позволяет использовать функцию получения/задания вместо непосредственного изменения свойств объектов. Сначала кратко опишем преимущества функций получения/задания, а затем рассмотрим, как используется метод `addProperty()`.

Пусть создается класс `MyClass`, для экземпляров которого задано свойство `myProp`:

```
function MyClass () {
  this.myProp = "myVal";
};
```

Теперь при создании экземпляра он будет иметь свойство `myProp`:

```
myObj = new MyClass 0;  
trace(myObj.myProp); // myVal
```

Если вы хотите изменить значение `myObj.myProp`, это можно сделать непосредственно, как показано ниже.

```
myObj.myProp = "myNewVal";  
trace(myObj.myProp); // myNewVal
```

Однако лучшим способом изменения значения `myObj.myProp` считается создание функции задания:

```
MyClass.prototype.setMyProp = function (newVal) {this.myProp =  
newVal;};  
myObj.setMyProp("myNewVal");  
trace(myObj.myProp); // myNewVal
```

Преимущество заключается в том, что если вы решите проверить достоверность данных или предпринять какие-то другие действия на основе представленного значения, это можно сделать в функции задания, больше ничего не изменяя в программе. Если же где-либо использовались прямые ссылки, придется все их изменить (или заменить на функцию задания).

Функции получения дают аналогичные преимущества при извлечении данных:

```
MyClass.prototype.getMyProp = function () {return this.myProp;};  
trace(myObj.getMyProp()); // myNewVal
```

Функции получения/задания могут оказаться особенно важными при коллективной работе, поскольку они обеспечивают простое, согласованное взаимодействие всех пользователей с объектом, одновременно позволяя вносить внутренние корректировки.

Большинство встроенных свойств, например `_x` и `_y` для видеоклипов, являются функциями получения/задания.

Однако при работе с функциями получения/задания возникают две проблемы.

- Первая проблема касается удобочитаемости. Например, строка

```
myObj.myProp = "myNewVal";
```

- более удобочитаема, чем строка

```
myObj.setMyProp("myNewVal");
```

- Вторая проблема заключается в том, что только лишь создание функций получения/задания не побуждает к их применению. При возникновении проблем все равно необходимо выяснить, не вызваны ли они наличием прямой ссылки, позволяющей обойти функции получения/задания.

Обе проблемы можно решить с помощью метода `addProperty()`. Он позволяет добавить свойство и установить его соответствие с функциями получения/задания с помощью всего лишь одного шага. После этого прямая ссылка фактически будет вызывать функции получения/задания. Таким образом, мы получим более удобочитаемый синтаксис и будем застрахованы от обхода функций получения/задания.

Метод `addProperty()` имеет такой формат:

```
myObj.addProperty("myProp", getMyProp, setMyProp )
```

Вместо имен функций получения и задания можно использовать функциональные литералы, как продемонстрировано в содержащемся на компакт-диске файле `addpropertyliteral fla`.

Если указаны и функция получения, и функция задания, то обе они должны существовать. Если вы хотите создать свойство, предназначенное только для чтения, то на место функции задания поместите `null`. В приведенном ниже примере `x` и `y` изменяют с помощью других функций, а предназначенное только для чтения свойство `diff` используется для отслеживания разницы между `x` и `y`.

```
myObj = { x : 2 , y : 3 };
function getDiff () {
    return (this.x - this.y);
}
rayObj.addProperty("diff", getDiff, null);
var difference = myObj.diff; // -1
```

Если теперь попытаться задать свойство `myObj.diff`, то обнаружится, что код выполняться не будет:

```
myObj.diff = 20;
trace(myObj.diff); // -1
```

Метод `addProperty()` возвращает значение `true`, если свойство добавлено успешно; в противном случае возвращается значение `false`.

Обратите внимание на то, что определенная вами функция задания *фактически не может изменить значение свойства*. Она всего лишь в виде аргумента получает новое значение и позволяет на его основе предпринять определенные действия. Значение изменяется независимо от функции задания, т.е. вне контроля программиста. В файле `getset fla` (листинг 19.1) функция задания `setMyProp()` не содержит предложений, но значение все равно устанавливается — это яркая иллюстрация того, что установка значения выполняется "за кулисами".

Хотя это и не является интуитивно понятным, но свойство может изменить функция получения. Свойство будет установлено в любое значение, возвращаемое функцией. Например, приведенная ниже функция получения присоединяет строку "addon" к свойству `myProp` при любом считывании свойства.

```
MyClass.prototype.getMyProp = function () {
    this.myProp += "addon";
    return this.myProp;
};
```

Метод `addProperty()` переписывает существующее свойство, имеющее то же имя, что и свойство в первом аргументе. Следовательно, назначение, выполненное в строке 8, должно происходить *после* предложения `addProperty()`. В противном случае свойство `myProp` было бы существующим свойством и было бы уничтожено вместе со значением `myVal`.

ЛИСТИНГ 19.1. GETSET.FLA

```
1: function MyClass () {
2:     var getMyProp = function () {
3:         return this.myProp;
4:     };
5:     var setMyProp = function (newVal) {
6:     };
7:     this.addProperty("myProp", getMyProp, setMyProp);
8:     this.myProp = "myVal"; // это нужно сделать после addProperty
9: }
10: myObj = new MyClass();
11: trace(myObj.myProp); // myVal
12: myObj.myProp = "myVal2";
13: trace(myObj.myProp); // myVal2
```

Тот факт, что функция установки фактически не изменяет переменную, сначала может показаться существенным недостатком, поскольку это означает, что установленное значение изменить невозможно. Однако установленное свойство можно рассматривать как команду выполнить другие действия, а не как элемент данных. Посмотрим на установленное значение, а затем изменим остальные значения по своему усмотрению. Факт наличия "входных" объектов и свойств, обеспечивающих взаимодействие с классом, и "выходных" объектов и свойств, которые "выполняют задачу", являются частностью реализации, которую можно скрыть внутри класса.

Обратимся к приведенному ниже листингу кода, взятому из файла `addprop fla`. Если значение, переданное функции задания, отличается от "Bad Bart", то строка "okay" передается в функцию индикатора состояния (строка 7), а значение отображается в текстовом поле (строка 8).

С другой стороны, если значением, переданным функции установки, является "Bad Bart", то в функцию индикатора состояния передаются строки "error" и "Bad Bart" (строка 11), а в текстовом поле (строка 12) отображается пустая строка (т.е. ничего). Свойство функции получения/задания `myObj.myProp` успешно установлено в значение "Bad Bart". Несмотря на это, с точки зрения пользователя (или другого программиста, пишущего код, взаимодействующий с данным классом) значение "Bad Bart" в качестве входного отклоняется, и отображается сообщение об ошибке.

```
1: function MyClass () {
2:     var getMyProp = function () {
3:         return this.myProp;
4:     };
5:     var setMyProp = function (newVal) {
6:         if (newVal != "Bad Bart") {
7:             setStatus("okay");
8:             setDisplayText(newVal);
9:         }
10:        else {
11:            setStatus("error", newVal);
12:            setDisplayText("");
13:        }
14:    };
15:    this.addProperty("myProp", getMyProp, setMyProp);
16:    this.myProp = "Beautiful Betsy"; // сделать после addProperty
17: }
```

Еще одна возможность изменения установленного значения заключается в использовании метода `Object.watch()`, описываемого в следующем подразделе. Однако этот метод может занять гораздо больше ресурсов процессора по сравнению с методом `addProperty()`.

Совет

Если вы только думаете о попытке замены свойства функции получения/задания, обратитесь к разделу "Возможные проблемы" в конце этой главы.

МЕТОДЫ WATCH() И UNWATCH()

Методы `watch()` и `unwatch()` соответственно регистрируют и отменяют регистрацию функции обратного вызова, обращение к которой производится при любой попытке изменить определенное свойство. Это очень мощная возможность, однако за ее использование приходится расплачиваться: при поиске все время изменяющегося свойства будет постоянно активизироваться функция обратного вызова, что создает чрезмерную нагрузку на процессор.

Метод `watch()` полезен для отладки программы. Кроме того, его можно использовать в качестве универсального генератора событий, таких как `addListener()`. Однако можно за-

дать только одну точку наблюдения за любым заданным свойством. При задании второй точки наблюдения она заменит предыдущую. В этом заключается отличие данного метода от приемников: для любого заданного события можно указать сколько угодно приемников. С помощью метода `watch()` можно реализовать собственную модель события так, чтобы при изменении свойства "информацию" получало бы любое количество объектов.

Совет

Подробная информация о генераторах событий приведена в главе 16 "Взаимодействие, события и установление последовательности". Соответствующие ссылки приведены на узле <http://www.flashoop.com>.

Метод `watch()` не рекомендуется использовать совместно со свойствами функций получения/задания. Несмотря на то что до какой-то степени он будет работать, некоторые изменения могут быть пропущены, и, кроме того, существует потенциальная опасность создания чрезмерной нагрузки на процессор. Большинство встроенных свойств являются свойствами получения/задания.

Чтобы начать наблюдение за свойством, необходимо выполнить следующие действия: определить функцию обратного вызова и выполнить предложение `watch()`. Форматы обоих действий, а также форматы завершения методов `watch()` и `unwatch()` представлены в табл. 19.8.

ТАБЛИЦА 19.8. ФУНКЦИЯ ОБРАТНОГО ВЫЗОВА И МЕТОДЫ WATCH И UNWATCH

МЕТОД/ ФУНКЦИЯ	ФОРМАТ	ПАРАМЕТРЫ
<code>watch</code>	<code>myObj.watch(свойство, обратный_вызов, данные_пользователя)</code>	<i>свойство</i> — указывает свойство, подлежащее наблюдению (в виде строки). <i>Обратный_вызов</i> — указывает функцию обратного вызова (ссылку) <i>данные_пользователя</i> (опционально) — указывает данные, подлежащие отправке в функцию обратного вызова
Функция обратного вызова	<code>function myFunc (свойство, старое_значение, новое_значение, данные_пользователя) { // предложения }</code>	<i>свойство</i> — содержит изменяемое свойство. <i>старое_значение</i> — содержит предыдущее значение свойства. <i>новое_значение</i> — содержит новое значение свойства. <i>Данные_пользователя</i> — содержит данные, определенные в одноименном параметре предложения <code>watch()</code>
<code>unwatch</code>	<code>myObj.unwatch (свойство)</code>	<i>свойство</i> — указывает свойство для остановки наблюдения

Оба метода, `watch()` и `unwatch()`, при успешном выполнении возвращают значение `true`; в противном случае возвращается значение `false`.

Функция обратного вызова активизируется как метод объекта, содержащий наблюдаемое свойство (как `myObj` в табл. 19.8). Внутри функции обратного вызова ключевое слово `this` относится к объекту `myObj`.

Объект, такой как `myObj`, содержащий свойство, за которым проводится наблюдение, должен существовать при задании точки наблюдения. При удалении объекта точка наблюдения исчезает. Однако точка наблюдения существует независимо от *свойства*, за которым выполняется наблюдение. Точку наблюдения можно задать для еще не существующего свойства, а в силу она вступит, когда свойство будет создано. Аналогичным образом можно удалить свойство, для которого была задана точка наблюдения, без уничтожения этой точки. Если позднее вы снова создадите свойство, то точка наблюдения все так же будет действовать. Для удаления точки наблюдения без удаления объекта воспользуйтесь методом `unwatch()`.

В приведенном ниже коде файла `watch fla` содержится объект `statusObj` со свойством `isOkay`.

```
statusObj = new Object();
statusObj.isOkay = "okay";
```

Следующая строка устанавливает точку наблюдения за свойством `isOkay`:

```
statusObj.watch("isOkay", setStatusText);
```

Функция обратного вызова `setStatusText()` имеет следующий вид (четвертый параметр, *данные_пользователя*, опущен, поскольку он не используется):

```
function setStatusText (prop, oldval, newVal) {
    statusField.text = newVal;
    if (newVal == "okay") {
        statusField.setTextFormat(okayFormat);
    }
    else {
        statusField.setTextFormat(errorFormat);
        setDisplayText("");
    }
}
```

Если новым значением свойства `isOkay` является `"okay"`, то формат текста в поле состояния устанавливается в значение `okayFormat`. Если новое значение свойства `isOkay` отличается от `"okay"`, то формат текста в поле состояния устанавливается в значение `errorFormat`, а поле отображения останется пустым (путем его установления в значение пустой строки).

Выполнение функции обратного вызова иницируется при *попытке* изменения значения свойства. Однако любым фактическим изменением этого свойства управляет функция обратного вызова. Если функция обратного вызова *не* возвращает значение, то значение свойства определено не будет (`undefined`). В только что представленной функции `setStatusText()` такое происходит всегда. Данная функция никогда не возвращает значения, поэтому свойство, за которым проводится наблюдение, всегда не определено.

Если функция обратного вызова возвращает значение, то свойство устанавливается равным возвращенному значению. В связи с тем, что функция обратного вызова в качестве своего второго параметра получает предыдущее значение, она может аннулировать *изменение*, просто вернув предыдущее значение.

Возможность метода `watch()` управлять значением свойства или аннулировать изменения отличает его от метода `addProperty()`. Данная возможность проиллюстрирована в файле `watch2 fla`. Когда функция обратного вызова `myPropHandler()` получает неприемлемое новое значение, она возвращает старое значение, аннулируя изменение (строка 11).

```
1: function myPropHandler (prop, oldval, newVal) {
2:     if (newVal != "Bad Bart") {
3:         myDisplayProp = newVal;
4:         myStatusProp = "okay";
```

```

5:         statusField.setTextFormat(okayFormat);
6:         return newVal; // принять изменение
7:     }
8:     else {
9:         myStatusProp = "error";
10:        statusField.setTextFormat(errorFormat);
11:        return oldVal; // отклонить изменение
12:    }
13: }

```

ОБЪЕКТ MATH

Часто для достижения нужно эффекта движения, как плавного, так и совершенно неестественного, используются математические функции. Как правило, они являются самым "изящным" способом достижения определенной цели, т.е. они проще как концептуально, так и в исполнении.

Кроме того, они нередко являются более гибкими по сравнению с другими подходами, в том числе и построением промежуточных отображений, т.е. с их помощью можно выполнять гораздо более широкий диапазон задач.

На заметку

Комбинируя методы рисования и математические функции, можно добиться восхитительных результатов. В качестве примеров обратитесь к содержащимся на компакт-диске файлам `3Dcube.fl` и `test5.fl` от разработчика Кита Питерса (Keith Peters), а также к описанным в главе 18 "Процедура рисования с помощью ActionScript" файлам `api_flower.fl` и `api_cube.fl` от разработчика Милли Маруани (Millie Maruani).

Объект `Math` имеет такое количество методов с самыми разными областями применения, что им можно посвятить целую книгу. В этой книге приведены примеры только наиболее распространенных методов. Полностью математические функции перечислены в табл. 19.9.

ТАБЛИЦА 19.9. МЕТОДЫ И СВОЙСТВА ОБЪЕКТА MATH

Имя	Метод/ свойство	ФОРМАТ	ОПИСАНИЕ
<code>abs</code>	Метод	<code>Math.abs(x)</code>	Вычисляет абсолютное значение <code>x</code>
<code>Acos</code>	Метод	<code>Math.acos(x)</code>	Вычисляет арккосинус <code>x</code>
<code>As in</code>	Метод	<code>Math.asin(x)</code>	Вычисляет арксинус <code>x</code>
<code>atan</code>	Метод	<code>Math.atan(x)</code>	Вычисляет арктангенс <code>x</code>
<code>atan2</code>	Метод	<code>Math.atan2(y, x)</code>	Вычисляет арктангенс отношения <code>y/x</code> , т.е. угол между положительной осью <code>x</code> и точкой (<code>x</code> , <code>y</code>)
<code>ceil</code>	Метод	<code>Math.ceil(x)</code>	Округляет <code>x</code> в сторону увеличения до ближайшего целого числа
<code>cos</code>	Метод	<code>Math.cos(x)</code>	Вычисляет косинус <code>x</code>
<code>exp</code>	Метод	<code>Math.exp(x)</code>	Вычисляет <code>Math.E</code> в степени <code>x</code>
<code>floor</code>	Метод	<code>Math.floor(x)</code>	Округляет <code>x</code> в сторону уменьшения до ближайшего целого числа
<code>log</code>	Метод	<code>Math.log(x)</code>	Вычисляет натуральный логарифм <code>x</code>

Имя	Метод/ свойство	ФОРМАТ	ОПИСАНИЕ
max	Метод	<code>Math.max(x, y)</code>	Возвращает большее целое из <code>x</code> и <code>y</code>
min	Метод	<code>Math.min(x, y)</code>	Возвращает меньшее целое из <code>x</code> и <code>y</code>
pow	Метод	<code>Math.pow(x, y)</code>	Возводит <code>x</code> в степень <code>y</code>
random	Метод	<code>Math.random()</code>	Возвращает псевдослучайное число между 0 и 1
round	Метод	<code>Math.round(x)</code>	Округляет <code>x</code> до ближайшего целого числа
sin	Метод	<code>Math.sin(x)</code>	Вычисляет синус <code>x</code>
sqrt	Метод	<code>Math.sqrt(x)</code>	Вычисляет квадратный корень из <code>x</code>
tan	Метод	<code>Math.tan(x)</code>	Вычисляет тангенс <code>x</code>
E	Свойство	<code>Math.E</code>	Константа Эйлера и основание натурального логарифма (приблизительно 2,718), традиционно обозначаемая как <code>e</code>
LN2	Свойство	<code>Math.LN2</code>	Натуральный логарифм 2 (приблизительно 0,693)
LOG2E	Свойство	<code>Math.LOG2E</code>	Логарифм <code>Math.E</code> по основанию 2 (приблизительно 1,442)
LN10	Свойство	<code>Math.LN10</code>	Натуральный логарифм 10 (приблизительно 2,302)
LOG10E	Свойство	<code>Math.LOG10E</code>	Логарифм <code>Math.E</code> по основанию 10 (приблизительно 0,434)
PI	Свойство	<code>Math.PI</code>	Отношение длины окружности к ее диаметру (приблизительно 3,14159)
SQRT1_2	Свойство	<code>Math.SQRT1_2</code>	Величина, обратная квадратному корню из 2, т.е. <code>1/Math.SQRT2</code> (приблизительно 0,707)
SQRT2	Метод	<code>Math.SQRT2</code>	Квадратный корень из 2 (приблизительно 1,414)

НАИБОЛЕЕ РАСПРОСТРАНЕННЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ

Наиболее широко используемыми методами объекта `Math` являются `floor()`, `random()` и `round()`.

МЕТОД FLOOR()

Метод `floor()` усекает десятичное число и возвращает только его целую часть. Например:

```
x = Math.floor (8.21) // результат: 8
x = Math.floor (8.5) // результат: 8
x = Math.floor (8.6) // результат: 8
x = Math.floor (-8.21) // результат: -9
x = Math.floor (-8.5) // результат: -9
x = Math.floor (-8.6) // результат: -9
```


МЕТОД RANDOM()

Метод `random()` возвращает десятичное число, большее или равное 0 и меньшее 1; обычно это число представлено 14–17 разрядами. Например:

```
x = Math.random();  
/*  
Примеры результатов:  
0.236938397510414  
0.102950482211518  
0.274189059284604  
0.585484127786702  
0.00277895387391511  
0.80261959452304  
*/
```

МЕТОД ROUND()

Метод `round()` округляет десятичное число до ближайшего целого. Приведем несколько примеров:

```
x = Math.round(8.21) // результат: 8  
x = Math.round(8.5) // результат: 9  
x = Math.round(8.6) // результат: 9  
x = Math.round(-8.21) // результат: -8  
x = Math.round(-8.5) // результат: -8  
x = Math.round(-8.6) // результат: -9
```

В приведенном ниже примере `mcPercent` — это целая процентная часть видеоклипа `myClip`, которая была загружена.

```
mcPercent = Math.round((myClip.getBytesLoaded() /  
myClip.getBytesTotal()) * 100);
```

Чтобы округлить десятичную дробь, сначала умножьте ее на число, являющееся степенью 10, округлите, а затем разделите на то же число. Используемое число определяет количество десятичных разрядов, до которых производится округление. Например, в приведенном ниже коде `y` округляется до одного десятичного разряда.

```
y = .12;  
yr = Math.round(10*y)/10; // yr равно .1
```

В приведенном примере, если `y` — любая положительная десятичная дробь, то `yr` будет равно 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, ИЛИ 1.0.

Совет

Применение округления до одного десятичного разряда будет рассмотрено ниже, в подразделе “Использование тригонометрии для получения и задания направления”.

ГЕНЕРАЦИЯ СЛУЧАЙНОГО ЦЕЛОГО ЧИСЛА от 0 до 9 ВКЛЮЧИТЕЛЬНО

Объединив методы `random()` и `round()`, можно получать случайные целые числа. В данном коде операции выполняются одна за другой:

```
for (i = 0; i < 10; i++) {  
    x = Math.random(); // генерация случайного числа меньше 1  
    x *= 10; // умножение на 10  
    x = Math.round(x); // округление  
    trace(x);  
}
```

Более кратким способом является следующее предложение:

```
trace(Math.round(Math.random()*10));
```

ИСПОЛЬЗОВАНИЕ МЕТОДА `MATH.SQRT()` ДЛЯ ПОЛУЧЕНИЯ И ЗАДАНИЯ ОТНОСИТЕЛЬНОГО РАСПОЛОЖЕНИЯ

Метод `Math.sqrt()` можно использовать и для определения, и для управления положением одного видеоклипа относительно другого либо относительно любой известной точки рабочего поля. Например, метод `Math.sqrt()` можно использовать для поддержания постоянного расстояния между перетаскиваемым видеоклипком и точкой, привязывая таким образом перемещение клипа к определенной дуге.

Использование метода `Math.sqrt()` для управления местоположением основывается на том, что положение клипа в рабочем поле полностью определяется двумя числами: положением x и положением y . Следовательно, перетаскиваемый видеоклип можно привязать к любой траектории, если вы придумаете формулу, которая при наличии одной координаты сможет дать значение другой.

Рассмотрим рис. 19.1. Если указать координату x точки, находящейся на круге, то вы сможете примерно определить, какую координату y необходимо задать для мыши, чтобы объект оставался расположенным на дуге.

Вероятно, вы не удивитесь, что существует математическое выражение, позволяющее выполнить ту же самую задачу. Поместив это выражение в обработчик события входного кадра видеоклипа мыши, можно заставить мышшь оставаться на дуге в каждом кадре, в котором координата x позволяет это сделать. Эта методика продемонстрирована в файле `constrain fla`.

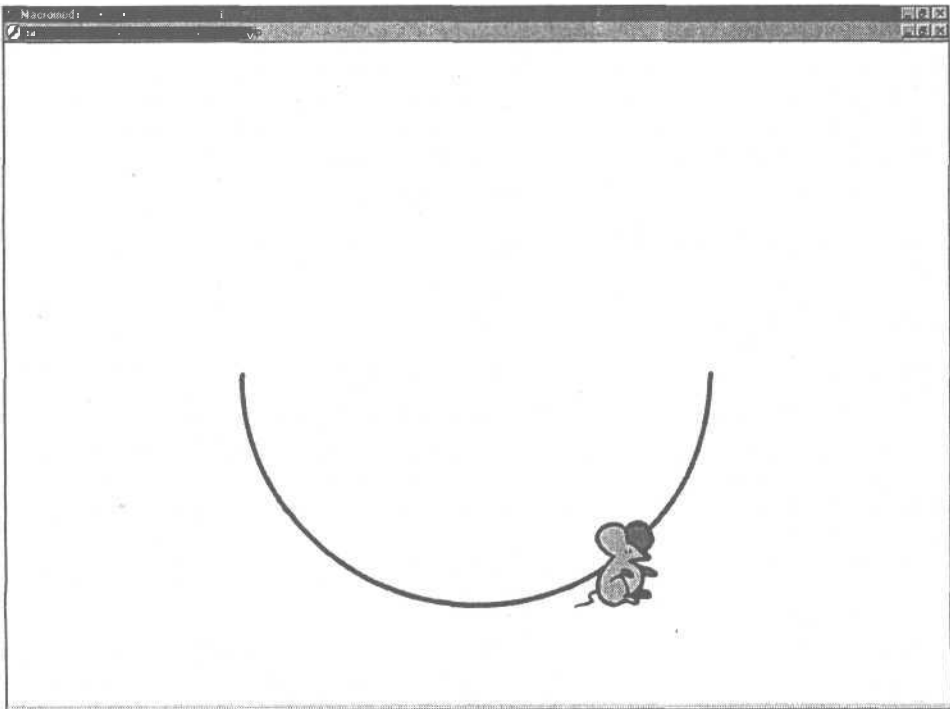


Рис. 19.1. Функция `Math.sqrt()` удерживает мышшь на определенном расстоянии от центральной точки дуги

Математическое уравнение получает свойство мыши `_x`, а затем задает соответствующее свойство `_y`. В уравнении используется тот факт, что каждая точка дуги находится на одном и том же расстоянии от ее центра, равном половине ширины `arc` видеоклипа. Этим расстоянием является радиус дуги, который в формуле обозначается символом `r`. Ниже приведен код `ActionScript`.

```
onClipEvent(load){
    var r = root.arc._width/2; // радиус дуги
    var ctr_x = 400; // положение x центра дуги
    var ctr_y = 300; // положение y центра дуги
}
onClipEvent(enterFrame) {
    y = Math.sqrt( (r * r) - ( (_x - ctr_x) * (_x - ctr_x) )) + ctr_y;
}
```

Как была получена данная формула?

Начнем с того, что расстояние между двумя точками рабочего поля можно свести к разности координат `x` и `y`. На рис. 19.1 видно, что, для того, чтобы добраться от мыши до центра полукруга, не обязательно двигаться по кратчайшему расстоянию. Вместо этого можно подняться вверх (разность `y`), а затем переместиться влево (разность `x`).

По аналогии, если заданы разности `x` и `y`, то будет очевиден прямой путь. Эти три расстояния образуют треугольник, а имея две стороны треугольника, очень легко **вычислить** третью.

В математическом выражении, если вы знаете расстояние по прямой от мыши до центра дуги и разность `x` от мыши до центра дуги, то можете вычислить разность `y`.

В данном случае расстояние от мыши до центра дуги является радиусом дуги, т.е. `r` равно половине ширины видеоклипа дуги (`arc._width / 2`). Разностью `x` является `_x - ctr_x`, где `_x` — координата `x` мыши (при этом имя экземпляра видеоклипа мыши задавать не нужно, поскольку код находится внутри обработчика события мыши), а `ctr_x` — координата `x` центра дуги. Аналогичным образом, разностью `y` является `_y - ctr_y`.

Равенство, позволяющее вычислить разность `y`, является разновидностью *теоремы Пифагора*, которая говорит о том, что $y^2 = r^2 - x^2$, где `y` — разность `y`, `r` — радиус, а `x` — разность `x`.

На языке `ActionScript` данная формула выглядит следующим образом:

$$(y - ctr_y) * (y - ctr_y) = r * r - (x - ctr_x) * (x - ctr_x);$$

Извлекая квадратный корень из обеих частей равенства, получим следующее:

$$y - ctr_y = \text{Math.sqrt}(r * r - (x - ctr_x) * (x - ctr_x));$$

Прибавив к обеим частям равенства `ctr_y`, получим желаемую формулу:

$$y = \text{Math.sqrt}(r * r - (x - ctr_x) * (x - ctr_x)) + ctr_y;$$

ОСНОВНЫЕ ТРИГОНОМЕТРИЧЕСКИЕ МЕТОДЫ `MATH.SIN()` И `MATH.COS()`

Наиболее широко используемыми тригонометрическими методами и, вероятно всего, самыми разносторонними методами объекта `Math` являются `Math.sin()` и `Math.cos()`. Их функциональные возможности будут описаны и продемонстрированы в следующих подразделах. Например, в файле `treeshadow fla` свойство `_rotation` (вращение) видеоклипа используется для создания эффекта падения дерева, эффект удлинения тени при этом достигается с помощью метода `Math.cos`, а для расчета того, насколько далеко находится вершина дерева от земли, используется метод `Math.sin`.

В файле `enemy fla` тригонометрия используется для того, чтобы определить, направлено ли ружье на врага. Тот же способ можно использовать для того, чтобы рассчитать, на сколько градусов (`_rotation`) чудовище должно раздвинуть свои челюсти для того, чтобы проглотить объект заданного размера.

На рис. 19.2, слева, изображен треугольник, расположенный внутри круга, который дает наглядное представление о функциях синуса и косинуса. Тригонометрические методы претворены в жизнь в файле `trigdemo fla` от разработчика Хелен Триоло (Helen Triolo). На рисунке с левой стороны функции синуса и косинуса показаны с помощью катетов треугольника в "единичном круге", радиус r которого равен одной единице длины. Радиус является гипотенузой (самой длинной стороной) треугольника. Радиус и горизонтальный катет образуют угол θ (theta). При вращении радиуса по кругу горизонтальный катет треугольника образует косинус, а вертикальный катет — синус угла θ (theta). На рис. 19.2, справа, показано, как функция синуса (возрастающая и убывающая) отображает синусоидальную волну во времени.

Внимание!

В "единичном круге" предполагается, что вращение на нуль градусов соответствует положению "3 часа". Для видеоклипа вращением на нуль градусов будет любое положение, в котором начинается вращение. Если видеоклип дерева начинается с дерева, стоящего прямо, то в единичном круге дерево будет указывать на 90 градусов (12 часов), а свойство `_rotation` при этом составляет нуль градусов.

Внимание!

Во Flash свойство `_rotation` отсчитывается по часовой стрелке, а в единичном круге подразумевается вращение *против часовой стрелки*. Чтобы естественное вращение Flash лучше соответствовало тригонометрическим функциям, его можно обратить.

По определению синусом является отношение вертикального катета к гипотенузе (y/r), но если длина гипотенузы равна единице ($r = 1$), то y/r равно y . Аналогичным образом, косинус, представляющий собой отношение горизонтального катета к гипотенузе (x/r), равен x , если $r = 1$.

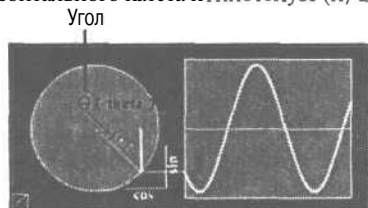


Рис. 19.2. Моментальный снимок файла `trigdemo fla` от разработчика Хелен Триоло

Линии, представляющие синус и косинус, находятся в одном видеоклипе `mcSinCos`. Их длина меняется при масштабировании размеров y (синус) и x (косинус) видеоклипа. Изменение длины описывают приведенные ниже строки кода.

```
mcSinCos._xscale = Math.cos(angle)*100;
mcSinCos._yscale = Math.sin(angle)*100;
```

Множитель 100 необходим из-за того, что свойства `_xscale` и `_yscale` являются процентными величинами от 0 до 100, тогда как синус и косинус дают результаты от 0 до 1.

Изменение размеров видеоклипа с помощью масштабирования

Обычная формула для задания ширины с помощью косинуса имеет следующий вид:
`width = cos(angle) * radius.`

В файле `trigdemo fla` размеры `mcSinCos` изменяются посредством свойств `_xscale` и `_yscale`, а не `_width` и `_height`. Такой подход требует умножения на 100. Чтобы получить число 100, необходимо применить выражение `radius * 2`. Ширина `mcRadius` составляет 50 пикселей.

Если бы видеоклип `mcSinCos` имел размер 100x100 пикселей, то задание его свойства `_xscale` было бы аналогично заданию ширины (свойства `_width`). Например, поскольку 10% от 100 равно 10, то `mcSinCos._xscale = 10` означает то же самое, что и `mcSinCos._width = ю`. В этом случае можно использовать традиционную формулу, а правильным множителем в данном случае будет длина радиуса (50).

Однако фактически клип `mcSinCos` имеет размеры 50x50 пикселей. Следовательно, свойство `_width` всегда равно половине значения `_xscale`. Например, `mcSinCos._xscale = 10` аналогично `mcSinCos._width = 5`. Значит, для задания правильной ширины в обычную формулу необходимо внести поправку: `width = cos(angle) * radius * 2`.

Размер по горизонтали (косинус) равен 100%, когда угол составляет 0 радиан, поскольку `Math.cos(0)` равен 1. Размер по горизонтали равен 0%, когда угол равен `Math.PI/2` радиан (90 градусов), так как `Math.cos(Math.PI/2)` равно 0.

Совет

Вопросы измерения углов в радианах и градусах будут рассмотрены ниже, во врезке "Радианы и градусы".

Совет

При использовании тригонометрических функций получились ошибочные результаты? Обратитесь к разделу "Возможные проблемы" в конце этой главы.

Размер по вертикали равен 0%, когда угол составляет 0 радиан, так как `Math.sin(0)` равен 0. Размер по вертикали равен 100%, когда угол составляет `Math.PI/2` радиан (90 градусов), поскольку `Math.sin(Math.PI/2)` равно 1.

Синусоидальная волна, изображенная на рис. 19.2, справа, создана путем дублирования точки видеоклипа в положении `y` изменяющегося синуса. Код в обработчике события *входного кадра* перемещает каждую точку вправо и удаляет ее, когда достигается определенное положение `x`, `_root.nXStop`:

```
onClipEvent(enterFrame) {
    _parent._x += .5; // перемещение точки вправо
    if (_parent._x > _root.nXStop) { //находится ли точка за указанным
        //положением x position?
        _parent.removeMovieClip(); //если да, то удалить ее
    }
```

Выполняя визуализацию фильмов в единичном круге, вы сможете увидеть, как достигаются эффекты, упомянутые в начале данного подраздела.

ИСПОЛЬЗОВАНИЕ ТРИГОНОМЕТРИИ для ПОЛУЧЕНИЯ и ЗАДАНИЯ ШИРИНЫ и ВЫСОТЫ

В файле `treeshadow.fla` свойство `_rotation` видеоклипа используется для создания эффекта падения дерева, показанного на рис. 19.3. Удлинение тени дерева при падении достигается за счет использования метода `Math.cos`, а расстояние вершины дерева от земли рассчитывается с помощью метода `Math.sin`.

Падающее дерево представляет собой вращающийся радиус (`treeHeight`) в программе. Он является *высотой* видеоклипа `tree` (дерево), поскольку отсчет начинается, когда дерево в рабочем поле расположено вертикально. Тень, которую отбрасывает дерево, является косинусом. Расстояние верхушки дерева от земли при его падении является синусом.

В связи с тем, что речь идет не о единичном круге (`treeHeight` не равно 1), для получения косинуса необходимо разделить горизонтальный катет треугольника на радиус (`treeHeight`), т.е. когда дерево наклонено на половину расстояния до земли (45 градусов), мы получим:

```
shadow / treeHeight = cos(45)
```

Умножив обе части на длину дерева, получим:

```
shadow = cos(45) * treeHeight
```

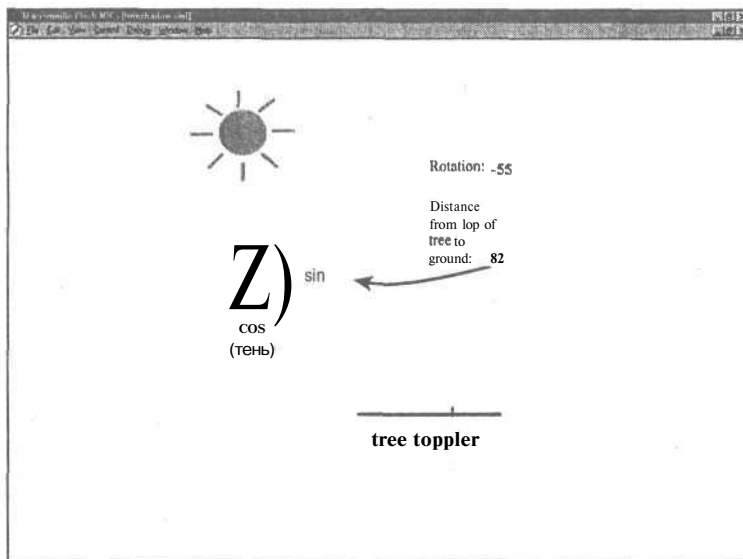


Рис. 19.3. Использование тригонометрии для удлинения тени и измерения расстояния от земли

В приведенной ниже строке кода, взятой из файла `treeshadow.fla`, для задания ширины тени вместо абсолютной величины используется свойство `shadow._xscale`. Однако, в связи с тем, что клип `_root.shadow` имеет ширину 100 пикселей, задание свойства `_xscale` аналогично заданию свойства `_width` (см. выше врезку "Изменение размеров видеоклипа с помощью масштабирования"). Кроме того, в приведенном ниже коде величина вращения преобразуется из градусов в радианы для использования функции косинуса.

```
_root.shadow._xscale = Math.cos(deg2rad(_root.tree._rotation))*treeHeight;
```

Радианы и градусы

В ActionScript тригонометрические функции требуют измерения углов в радианах. Если вы предпочитаете использовать градусы, необходимо применить функцию, преобразующую градусы в радианы:

```
радианы = градусы * pi / 180
```

Теперь вернемся к файлу `treeshadow.fla`. Чтобы получить косинус вращения падающего дерева, используется функция

```
function deg2rad(degrees) {
    return degrees * Math.PI/180;
}
Math.cos(_root.deg2rad(_root.tree._rotation))
```

В табл. 19.10 приведено несколько примеров измерения вращения в градусах и эквивалентные значения в радианах.

На заметку

Число π (я), представляемое в ActionScript как `Math.PI`, является отношением длины окружности к ее диаметру и приблизительно равно 3,14.

Аналогичным образом, синус представляет собой расстояние по вертикали, деленное на радиус, поэтому можно начать с формулы

расстояние от земли / treeHeight = sin(45)

Умножив обе части на длину дерева, получим:

расстояние от земли = treeHeight * sin(45)

ТАБЛИЦА 19.10. ГРАДУСЫ и РАДИАНЫ

ГРАДУСЫ	РАДИАНЫ
Полный круг (360)	2 * Math.PI
Половина круга (180)	Math.PI
Четверть круга (90)	Math.PI / 2
Восьмая часть круга (45)	Math.PI / 4

Для облегчения отображения в файле расстояние округляется. Кроме того, с помощью метода Math.abs оно преобразовывается в абсолютное значение для того, чтобы компенсировать присущее Flash обратное вращение. И наконец, из полученного значения вычитается 1, что приводит к округлению до 0 вместо округления до 1 в каждой предельной точке падения дерева.

```
_root.distanceFromGround =  
Math.round(Math.abs(Math.sin(deg2rad(_root.tree._rotation))  
*_treeHeight)-1);
```

ИСПОЛЬЗОВАНИЕ ТРИГОНОМЕТРИИ для ПОЛУЧЕНИЯ и ЗАДАНИЯ НАПРАВЛЕНИЯ

Чтобы определить, направлено ли оружие на врага, используйте центр вращения оружия в качестве центра единичного круга, как показано на рис. 19.4.

Начнем с того, что оружие находится в горизонтальном положении. Будем вращать его, одновременно проверяя в каждом кадре, равен ли синус вращения разности у между врагом и оружием:

```
if (Math.sin(angle) == y)
```

В файле enemy fla необходимо было добавить еще несколько действий. Например, фильм начинается с ненулевого положения оружия. Вращение "обнуляется" путем вычитания исходного вращения перед тем, как вычислить синус:

```
angle = _root.gun._rotation - gunRotationStart;
```

Естественно, выполняется обычное преобразование в радианы:

```
sinAngle = Math.sin(deg2rad(angle));
```

Окончательная проверка будет иметь очень простой вид:

```
if (sinAngle == yr)
```

Переменная уг в этом выражении является разностью у между оружием и врагом, выраженная в единицах r:

```
var y = _root.gun._y - _root.enemy._y;  
var yr = Math.round(((10*y) / r))/10;
```

Обратите внимание на то, что разность у округляется до одного десятичного знака с помощью метода Math.round().

Совет

Округление до одного десятичного знака рассматривалось выше, в подразделе "Метод round ()".

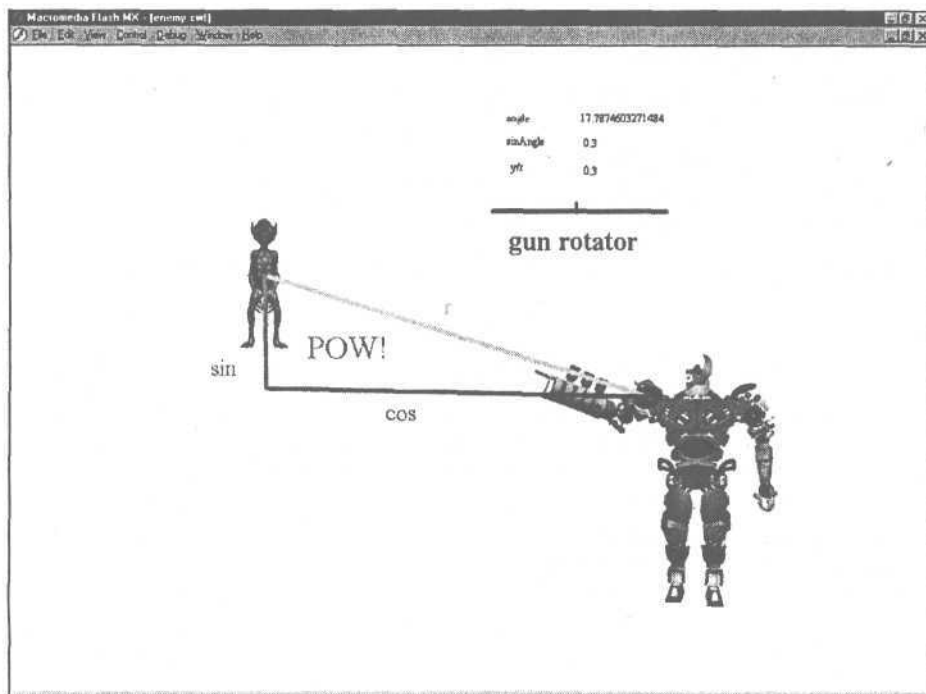


Рис. 19.4. В файле *enemy.fla* метод *Math.sin* используется для того, чтобы определить, когда оружие направлено на врага

Аналогичным образом округляется синус:

```
sinAngle = Math.round(10*sinAngle)/10;
```

И синус, и разность *y* необходимо округлить, иначе при проверке они из-за мельчайших несовпадений никогда не будут оценены как равные.

УПРАВЛЕНИЕ РАСПОЛОЖЕНИЕМ с помощью МЕТОДА *MATH.SIN()*

С помощью метода *Math.sin()* можно варьировать любое свойство видеоклипа, заменяя его на переменную *y*, как в приведенном ниже выражении, представляющем собой общий формат метода *Math.sin()*:

```
y = Math.sin(x);
```

На рис. 19.5 показан результат перемещения бабочки вправо (путем приращения свойства *_x* видеоклипа) на одну и ту же величину в каждом кадре при одновременном перемещении бабочки по вертикали (путем приращения свойства *_y* видеоклипа) с помощью функции *Math.sin()*.

На рисунке точками показана *синусоидальная* траектория, пройденная бабочкой. Ниже показана логика кода, управляющего перемещением бабочки (в виде псевдокода).

```
onClipEvent (enterFrame) {
    if (бабочка еще не достигла цели) {
        переместить бабочку на один шаг вправо;
        переместить бабочку по вертикали - величина зависит от синуса x;
    }
}
```


Функция синуса проходит один полный цикл (от пика до пика) по мере того, как аргумент x изменяется от 0 до 360 (традиционно это визуализируется как градусы круга). В данном примере бабочка проходит одну полную синусоидальную волну путем изменения свойства $_x$ от 0 до 360.

В файле `sine.fla` описанная логика выражена на языке **ActionScript**. В файл внесены две корректировки: добавляется переменная "поправка y " (`yadj`), увеличивающая амплитуду синусоидальной волны. Без такой корректировки движение бабочки вверх и вниз будет совсем незначительным. Кроме того, здесь используется функция `deg2rad()` для преобразования градусов в радианы. Ниже приводится код.

```
function deg2rad (degrees) // преобразование градусов в радианы {
    return degrees * Math.PI/180;
}
bfly.startx = 0; // начальная точка x
bfly.endx = 360; // конечная точка x
bfly.starty = 112;
bfly._x = startx; // положение бабочки (bfly) в начале
bfly._y = starty; // положение бабочки в начале
bfly.xinc = 16; // приращение x для перемещения вправо в каждом кадре
bfly.yadj = 8; // поправка y - увеличение амплитуды синусоидальной волны
}
bfly.onEnterFrame = function() {
    if (bfly._x < bfly.endx) {
        bfly._x += bfly.xinc; // перемещение бабочки на один шаг вправо
        // перемещение бабочки по вертикали; величина зависит от синуса x
        bfly._y += bfly.yadj * Math.sin(deg2rad(bfly._x)); }
}
```

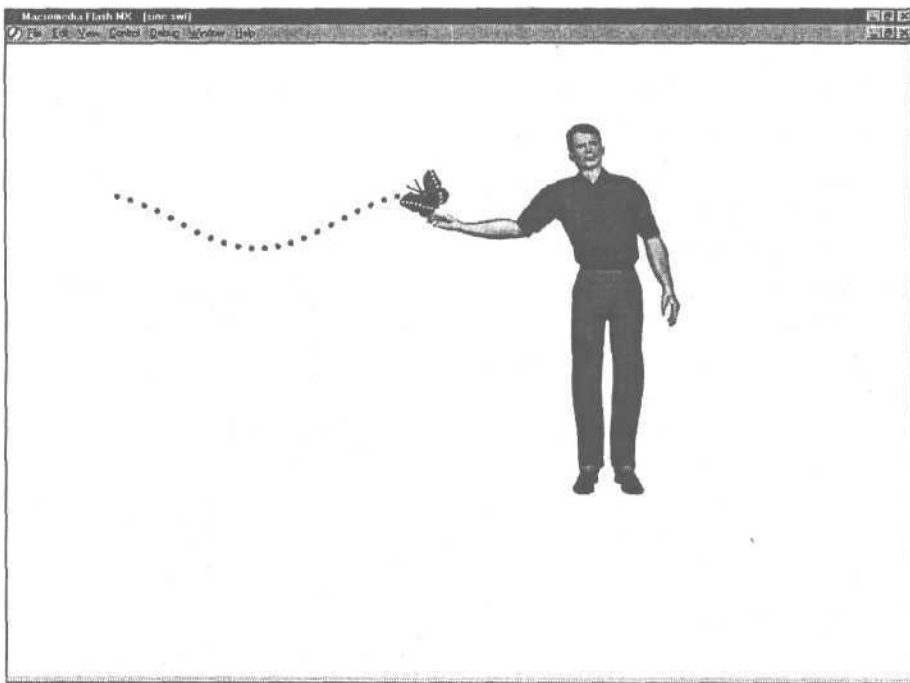


Рис. 19.5. С помощью метода `Math.sin()` можно управлять расположением, скоростью и вращением, добиваясь естественных, плавных преобразований и эффектов. В данном случае метод `Math.sin()` управляет положением бабочки по вертикали

УПРАВЛЕНИЕ СКОРОСТЬЮ с помощью МЕТОДА `MATH.SIN()`

Функцию синуса можно также применить для того, чтобы добиться плавного, естественного ускорения или замедления видеоклипа. На рис. 19.6 показано, что произойдет, если вместо перемещения бабочки вправо на одну и ту же величину в каждом кадре, варьировать приращение с помощью функции `Math.sin()`. Вместо `_x = xinc` используется выражение `_x += xinc + (xadj * Math.sin(deg2rad(_x)))`. Сюда входит и поправочная переменная `xadj`, которая в данном случае установлена равной 8. Эта переменная изменяет величину ускорения и замедления. При значениях от 0 до 180 градусов синус будет положительным, т.е. перемещение бабочки ускоряется при движении вниз. При значениях от 180 до 360 градусов синус будет отрицательным, т.е. перемещение бабочки замедляется при движении вверх.

Напомним, что во Flash понятие "движение вниз" означает увеличение значения свойства `_y`. "Ускорение" означает увеличение величины, прибавляемой к `_x` в каждом кадре. Таким образом, при положительном синусе бабочка будет двигаться вниз с ускорением. И наоборот, при отрицательном синусе бабочка будет двигаться вверх с замедлением. Изменение скорости отображается в виде расстояний по горизонтали между точками.

Замедление означает, что бабочка находится в радиусе от 180 до 360 градусов в большем числе кадров, чем она находится при 0–180 градусах. (На рис. 19.6 каждая точка является кадром. Обратите внимание на то, как много точек во второй половине пути бабочки.) Таким образом выражение `y += yadj * Math.sin(deg2rad(_x))` выполняется большее число раз с отрицательным синусом, чем с положительным, что заставляет бабочку больше двигаться вверх, а не вниз. Этот эффект можно компенсировать, начав с более низкого расположения, заменив `var starty = 175` на `var starty = 112`. (См. файл `sine2.fl` на прилагаемом к книге компакт-диске.)

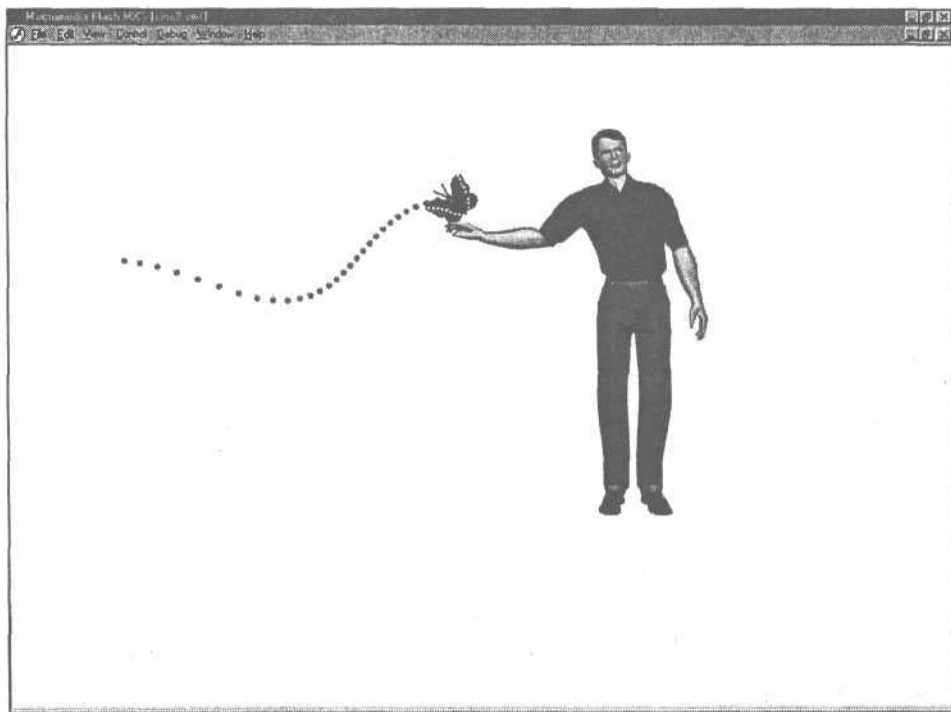


Рис. 19.6. Управление скоростью с помощью метода `Math.sin()`. При перемещении вправо движение бабочки замедляется

ДРУГИЕ ТРИГОНОМЕТРИЧЕСКИЕ МЕТОДЫ

Рассмотрим другие тригонометрические методы, выраженные в показателях синуса и косинуса единичного круга. Заменяем y/r на \sin , а x/r — на \cos , при условии, что r не равно 1. Угол θ во всех случаях указан в радианах.

Метод арксинуса (`asin`) возвращает угол, заданный синусом:

```
theta = asin(sin)
theta = Math.asin(y/r);
```

Метод арккосинуса (`acos`) возвращает УГОЛ, заданный косинусом:

```
theta = acos(cos)
theta = Math.acos(x/r);
```

Метод тангенса (`tan`) возвращает синус, деленный на косинус:

```
sin/cos = tan(theta)
y / x = Math.tan(theta);
```

Метод арктангенса (`atan`) возвращает угол, заданный тангенсом:

```
theta = atan(sin/cos)
theta = Math.atan(y/x);
```

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему массив чисел не сортируется правильно?

По умолчанию метод `Array.sort()` выполняет сортировку в алфавитном порядке. Если задан массив чисел, то данный метод преобразовывает числа в строки, а затем сортирует их в алфавитном порядке. В соответствии с этой процедурой 2 будет "больше", чем 10 000.

Чтобы отсортировать такой массив по числам, в качестве аргумента метода `Array.sort()` необходимо задать функцию сравнения. Приведенный ниже пример демонстрирует сортировку от наименьших чисел до наибольших.

```
numberArray = [3,2,6,3,8,3,9,5];
function byNumber (a, b) {
    return a - b;
}
numberArray.sort (byNumber);
trace (numberArray); // 2,3,3,3,5,6,8,9
```

Совет

Принципы работы функций сортировки рассматривались выше, в подразделе "Методы `sort()` и `sortOn()`".

Почему моя тригонометрическая функция "сходит с ума"?

Одна из самых распространенных причин, по которой тригонометрические функции дают совершенно неожиданные результаты, заключается в том, что вы забываете преобразовать градусы в радианы. Хотя во Flash свойство `_rotation` (вращение) измеряется в градусах, все тригонометрические функции Flash требуют, чтобы углы измерялись в радианах.

Совет

Вопросы измерения углов в радианах и в градусах обсуждались ранее, во врезке "Радианы и градусы".

FLASH ЗА РАБОТОЙ: УДИВИТЕЛЬНЫЕ МАССИВЫ

Массивы позволяют выполнять совершенно невероятные вещи. Например, в содержащемся на компакт-диске файле `amazingarrays fla` показано, что вы можете разбить изображение на 64 маленьких кусочка, перемешать их на экране, а затем сложить обратно, подобно тому, как в идеальном порядке приземляется на землю стая птиц.

"Это одна из моих самых старых демонстраций Flash", — говорит Фотиос Бассайаннис (Fotios Bassayiannis), Flash-разработчик греческого происхождения, живущий в Глазго (<http://fotios.cc>).

"Мой рисунок разбит на 64 маленьких кусочка. Каждому кусочку присвоен индекс от 0 до 64, начиная с левого верхнего угла и заканчивая правым нижним углом фотографии. Каждый раз при нажатии кнопки кусочкам задается случайное исходное размещение. Далее они стараются достичь своих координат назначения".

Программа рассчитывает координаты назначения кусочка на основе его индекса и положения целого изображения.

В данном примере мы не будем приводить построчное объяснение кода, а только попытаемся представить себе, как можно отслеживать все эти маленькие единицы информации без массивов!

У нас есть массив `pic`, содержащий 64 имени экземпляров видеоклипов.

Исходным значениям `x` и `y` каждого "среза" фотографии (т.е. каждого ее кусочка) соответствуют массивы `maxX` и `maxY`.

Кроме того, имеются два "граничных" массива, `limitX` и `limitY`, для ограничения перемещений каждого среза в направлениях `x` и `y`, т.е. граничные массивы содержат конечные координаты `x` и `y` каждого среза.

Манипуляция массивом осуществляется посредством простых циклов `for` и оператора присваивания. Это трио является одной из наиболее мощных комбинаций `ActionScript`.

Для инициализации массивов используется следующий код:

```
for (i = 1; i < 65; i++)
{
    Pic[i] = "mc" + String(i);
    maxX[i] = random(450) + 100; //расположение всех срезов в случай-
ном порядке
    maxY[i] = random(340) + 100;
    setProperty ( Pic[i], _x, maxX[i] );
    setProperty ( Pic[i], _y, maxY[i] );
}
```

Ниже приведен код, определяющий предельные значения координат `x` и `y` каждого среза фотографии.

```
for (i = 1; i < 65; i++)
{
    XmatrixPos = i % XSliceNum;
    YmatrixPos = Math.ceil (i / YSliceNum) ;
    if (XmatrixPos == 0) XmatrixPos = 8;
    if (YmatrixPos == 0) YmatrixPos = 8;
    limitX[i] = XmatrixPos * SliceWidth; //предел по оси x
    limitY[i] = YmatrixPos * SliceHeight; //предел по оси y
}
```

Предельные значения являются координатами точки, в которой должен оказаться каждый срез. Цикл определяет положение матрицы среза путем деления ее порядкового номера (слева направо и для всех строк) на число срезов по осям `x` и `y`.

Приведенное ниже пояснение поможет вам еще больше расчленить программу, если вас она заинтересовала. Фильм `amazingarray fla` состоит из трех кадров, но первый из них

является настроечным. После этого фильм циклически выполняется в кадрах 2 и 3, которые содержат идентичный код.

В кадре 1 создаются все массивы, а также ряд переменных. Кроме того, программа выясняет, к чему относится каждый срез ("положение матрицы среза", переменные `XmatrixPos` и `YmatrixPos`), на основе его индекса (1-64). Этот цикл располагается внизу кадра 1.

Во время циклического выполнения фильма в кадрах 2 и 3 каждый фрагмент фотографии должен перемещаться к своей точке назначения (и в направлении `x`, и в направлении `y`) или оставаться на месте, если он уже достиг этой точки.

Не все фрагменты достигают своих точек назначения в одно и то же время. Чем ближе к точке назначения они находятся, тем медленнее они движутся. (Значения приращения `x` и `y` изменяются для каждого из кусочков по мере его приближения к конечным координатам `x` и `y`.) Значение приращения по оси `x` и `y` для каждого фрагмента в каждом кадре рассчитывается отдельно и временно хранится в переменных `incX` и `incY`.

Каждый раз, когда фрагмент движется в направлении `x` или `y`, программа устанавливает переменную `change` в значение `true`. При каждом выполнении кадра начальным значением переменной `change` является `false`. Если в конце выполнения кадра ее значением останется `false`, это означает, что все фрагменты находятся на своих местах. В этом случае фильм останавливается посредством действия `stop()` и ожидает, когда пользователь нажмет кнопку. Если переменная `change` установлена в значение `true`, это означает, что хотя бы один фрагмент находится не на своем месте и фильм останавливаться не должен.

ИСПОЛЬЗОВАНИЕ ВСТРОЕННЫХ ОБЪЕКТОВ ФИЛЬМОВ

В ЭТОЙ ГЛАВЕ...

Объекты и классы, связанные с фильмами	467
Возможные проблемы	511
Flash за работой: полная автоматизация игры в крестики-нолики с помощью класса Local Connection	512

ОБЪЕКТЫ и КЛАССЫ, СВЯЗАННЫЕ с ФИЛЬМАМИ

В **ActionScript** определен ряд объектов, разработанных специально для создания Web-анимации. Тринадцать таких объектов относятся к категории **Movie**. В инструментальном списке панели **Actions** (Действия) программы **Flash MX** (в левой части панели) они находятся в папке **Movie** (Фильм) категории **Objects** (Объекты). Вот список этих 13 объектов: **Accessibility**, **System.capabilities**, **Button**, **Color**, **Key**, **Mouse**, **MovieClip**, **Selection**, **Sound**, **Stage**, **System**, **TextField** и **TextFormat**.

Это — специфические для **Flash** объекты, не определенные стандартом **ECMA-262** и не реализованные в **JavaScript**. Тем не менее к ним относятся наиболее распространенные классы **ActionScript**, такие как **MovieClip**, **TextField** и **TextFormat**.

LOB

Классы **TextField** и **TextFormat** рассматриваются далее в этой главе.

Кроме того, ниже рассмотрен ряд объектов, которые в первоначальной версии **Flash MX** были недокументированными. Это объекты **Camera**, **LocalConnection**, **Microphone**,

Video, а также еще два объекта (*NetConnection* и *NetStream*), связанные с Flash Communication Server.

Все классы и объекты, рассмотренные в этой главе, перечислены в табл. 20.1.

ТАБЛИЦА 20.1. ОБЪЕКТЫ, СВЯЗАННЫЕ С ФИЛЬМАМИ

ОБЪЕКТ	ОПИСАНИЕ	ОДНОЭЛЕМЕНТНОЕ МНОЖЕСТВО или КОНСТРУКТОР	ГЛАВА
Accessibility (н)	Программа считывания с экрана	Одноэлементное множество	20
Button (н)	Кнопки на экране	Конструктор (*)	16, 17
Camera (н)	Видеокамеры (Web-камеры)	Одноэлементное множество	20
Capabilities (н)	Системные характеристики компьютера, использующиеся с объектом System	Одноэлементное множество	20
Color	Цвета	Конструктор	20
Key (н)	Нажатия клавиш	Одноэлементное множество	16, 20
LocalConnection (н)	Локальная связь между фильмами	Конструктор	20
Microphone (н)	Микрофоны	Одноэлементное множество	20
Mouse	Курсор мыши и щелчки ею	Одноэлементное множество	16, 20
MovieClip	Видеоклип	Конструктор (*)	17
NetConnection (н)	Связь с Flash Communication Server	Конструктор	20
Netstream (н)	Канал связи с Flash Communication Server	Конструктор	20
Selection	Правка выделенного текста и фокуса	Одноэлементное множество	20
Sound	Звук/музыка	Конструктор	20
Stage (н)	Рабочее поле	Одноэлементное множество	20
System (н)	Компьютерная система	Одноэлементное множество	20
TextField (н)	Текстовые поля	Конструктор	20
TextFormat (н)	Форматирование текста в текстовом поле	Конструктор	20
Video (н)	Источник видео в реальном времени	Конструктор	20

(н) — новый во Flash MX;

(*) — "неконструктивный" конструктор; нельзя создавать новые экземпляры с помощью ключевого слова new.

ОБЪЕКТ ACCESSIBILITY



Объект Accessibility (Доступность) реализует возможность Rash-плеера сделать Flash-фильмы доступными для людей с ослабленным зрением. Эта возможность реализуется за счет программы считывания информации с экрана, которая поддерживает функцию Microsoft Active Accessibility (MSAA). Эта программа интерпретирует текст и другие определенные элементы, отображающиеся на экране, и громко зачитывает их. В настоящее время наиболее популярной программой считывания информации с экрана, поддерживающей функцию Active Accessibility, является продукт Window-Eyes от компании GWMicro.

Объект Accessibility имеет только одно документированное свойство — метод `isActive()`, который проверяет, является ли активной программа считывания информации с экрана, совместимая с MSAA. На основе этой информации в фильм можно загрузить более доступный интерфейс или перенаправить пользователя на специальный узел.

После загрузки фильма необходимо подождать несколько секунд и только затем активизировать метод `isActive()`. В противном случае метод может вернуть значение `false`, даже несмотря на то, что программа считывания является активной.

Чтобы определить, поддерживает ли система MSAA (независимо от того, активна ли программа считывания в данный момент), можно использовать свойство `System.capabilities.hasAccessibility`.

Совет

Вопросы доступности будут подробно рассмотрены в приложении А.

КЛАСС BUTTON



Класс Button (Кнопка) является новым во Flash MX. Естественно, кнопки были и в предыдущих версиях программы, но они не относились к системе классов. Данный класс позволяет, к примеру, программным образом определить, является ли объект кнопкой:

```
if ((typeof myObj == "object") && (myObj.constructor == Button))  
    // сделать что-то с объектом
```

Совет

Подробная информация об использовании свойства конструктора для определения класса объекта приведена в главе 19 "Использование встроенных базовых объектов".



Еще одним нововведением Rash MX являются имена экземпляров кнопок. Функция-конструктор Button является "неконструктивной". Кнопки создаются вручную или с помощью метода `AttachMovie()`. (Однако аргумент `initObject` метода `AttachMovie()` с кнопками не работает.)

Совет

Методики создания кнопок аналогичны соответствующим методикам создания видеоклипов, которые рассматривались в главе 17 "Демонстрация мощи видеоклипов".



Класс Button имеет только один метод, `getDepth()`, который возвращает показатель глубины экземпляра кнопки.

Совет

Подробная информация о показателях глубины приведена в главе 17 "Демонстрация мощи видеоклипов".

Класс `Button` имеет пять свойств (`enabled`, `tabEnabled`, `tabIndex`, `trackAsMenu` и `useHandCursor`) и девять событий (`onPress`, `onRelease`, `onReleaseOutside`, `onRollover`, `onRollOut`, `onDragOut`, `onDragOver`, `onKillFocus` и `onSetFocus`).

Совет

События, связанные с кнопками, были рассмотрены в главе 16 "Взаимодействие, события и установление последовательности". Кнопки имеют такие же свойства, как и те, что описаны в главе 17 "Демонстрация мощи видеоклипов".

ОБЪЕКТ `System.capabilities`



Объект `capabilities` является единственным документированным свойством одноэлементного множества `System`. Объект `capabilities` имеет 10 документированных и 7 недокументированных свойств, представляющих характеристики системы, в которой выполняется Flash-плеер. В табл. 20.2 перечислены эти свойства, начиная с документированных (с `language` свойства являются недокументированными).

Все свойства объекта `System.capabilities` предназначены только для чтения. Несмотря на то что их можно задавать и/или изменять, как правило, они не выполняют никаких задач.

В свойстве `serverString` все системные характеристики сведены в одну строку, которую можно передать на сервер связи или приложения. Эта строка (несмотря на то, что в таблице она разбита) является единой и именно в таком виде передается на сервер.

КЛАСС `COLOR`

Каждый экземпляр класса `Color` программным образом управляет цветом и прозрачностью определенного видеоклипа или основного фильма. Изменения цветов можно использовать для моделирования различных условий. Например, постепенное затемнение цвета всего фильма имитирует наступление ночи, а добавив желтый и белый цвет, вы можете на темном фоне выделить какой-то объект. Коэффициент прозрачности можно использовать для постепенного появления и исчезновения изображения, а также для моделирования чего-то полупрозрачного, подобного вуали.

Во Flash цвет имеет четыре компонента: красный, зеленый, синий и прозрачность.

В `ActionScript` существуют два способа работы с цветом.

- Задание или получение нового цвета путем замены существующего цвета без изменения прозрачности. С этой целью используются методы `setRGB()` и `getRGB()`.
- Использование метода `setTransform()` для модификации существующего цвета либо применение метода `getTransform()` для определения, каким образом цвет преобразуется в данный момент. При таком подходе помимо цвета можно задавать и получать коэффициент прозрачности.

В любом случае сначала необходимо создать объект цвета для определенного видеоклипа. Эта операция выполняется с использованием имени экземпляра целевого видеоклипа в качестве строкового параметра в предложении `new`:

```
myColor = new Color("myClip");
```

МЕТОДЫ `setRGB()` И `getRGB()`

Задать цвет видеоклипа можно с помощью метода `setRGB()` и шестизначного шестнадцатеричного числа, представляющего собой три пары разрядов для значений красного, зеленого и синего цветов соответственно. Ниже приведено несколько примеров.

```
myColor.setRGB(0xFF0000); // устанавливает красный цвет клипа myColor
myColor.setRGB(0x00FF00); // устанавливает зеленый цвет клипа myColor
myColor.setRGB(0x0000FF); // устанавливает синий цвет клипа myColor
```

ТАБЛИЦА 20.2. СВОЙСТВА ОБЪЕКТА **SYSTEM.CAPABILITIES**

Свойство	ОПИСАНИЕ	ПРИМЕР	КОММЕНТАРИЙ
hasAudio	Имеет аудио- характеристики	true	
hasMP3	Может воспроизводить MP3-файлы	true	
hasAudioEncoder	Имеет аудиокодер	true	
hasVideoEncoder	Имеет видеокодер	true	
screenResolutionX	Получает разрешение экрана по горизонтали (в пикселях)	1024	
screenResolutionY	Получает разрешение экрана по вертикали (в пикселях)	768	
screenDPI	Получает число точек на дюйм экрана в пикселях	72	
screenColor	Получает цветовую характеристику экрана: цветной, черно-белый или серый	color	
pixelAspectRatio	Получает соотношение размеров в пикселях	1	
hasAccessibility	Поддержка функции доступности	false	
language	Получает язык	en	Английский
OS	Получает операционную систему	Windows 2000	
input	Получает входное устройство	point	Мышь
manufacturer	Получает в виде строки данные о производителе	Macromedia Windows	
serverString	Получает строку для передачи на сервер	A=t&MP3=t&AE=t&VE=t&ACC=f&DEB=t&V=WIN%206%2C0%2C21%2C0&M=Macromedia Windows&R=1024x768&DP=72&COL=color&AR=1.0&I=point&OS=Windows2000&L=en-US	
isDebugger	Является ли активным отладчик	true	
version	Получает версию Flash-плеера	WIN 6,0,21,0	

Совет

Подробная информация о шестнадцатеричном представлении приведена ниже, во врезке "Значения RGB и шестнадцатеричное представление".

Значения RGB и шестнадцатеричное представление

В шестнадцатеричном представлении (по основанию 16) используется 16 нумералов: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Таким образом, F соответствует 15.

В десятичном представлении, т.е. по основанию 10, разряды справа налево возрастают соответственно степени числа 10: 1, 10, 100 и т.д. Каждый разряд в 10 раз больше разряда справа от него. В шестнадцатеричной системе разряды возрастают соответственно степени числа 16: 1, 16, 256 и т.д. Каждый разряд в 16 раз больше разряда справа от него.

Значения RGB (система цветопередачи “красный–зеленый–синий”), связанные с объектом Color, можно рассматривать как три двухразрядных шестнадцатеричных числа, каждое из которых представляет 256 значений, от 0 до 255. Итоговому шестизначному шестнадцатеричному числу предшествуют символы 0x, указывающие на шестнадцатеричное представление.

Результат будет иметь такой формат:

0xRRGGBB

где RR представляет собой двухразрядное шестнадцатеричное число, обозначающее красный цвет; GG — двухразрядное шестнадцатеричное число, обозначающее зеленый цвет; BB GG — двухразрядное шестнадцатеричное число, обозначающее синий цвет.

Web-разработчики знакомы с шестнадцатеричными числами по синтаксису HTML, подобному приведенному ниже.

```
<BODY BGCOLOR="#00FF00>
```

Во Flash палитра цветов заливки (обозначенная пиктограммой с изображением ковши с краской в разделе Colors (Цвета) на панели инструментов) позволяет легко определить шестнадцатеричные значения многих цветов. Щелкните на палитре цветов заливки, а затем проведите инструментом Eyedropper (Пипетка) по нескольким цветам. При этом в поле ввода данных, расположенном в верхней части панели, будет отображено шестнадцатеричное значение соответствующего цвета.

Метод `getRGB()` возвращает текущий назначенный цвет. Например, этот метод можно использовать для сравнения цветов, хранящихся в двух экземплярах класса Color.

Метод `getRGB()` возвращает цвет в виде десятичного числа. Чтобы преобразовать его в шестнадцатеричное число, воспользуйтесь методом `toString()` класса Number, как показано ниже.

```
hexColor = myColor.getRGB().toString(16);
```

МЕТОДЫ `setTransform()` И `getTransform()`

Метод `setTransform()` позволяет преобразовывать все цвета и коэффициенты прозрачности заданного видеоклипа. Например, если есть видеоклип, в котором содержится 100 других клипов различных цветов, то с помощью метода `setTransform()` одной операцией можно удалить из всех клипов немного красного цвета и добавить немного синего. В отличие от этого метода, с помощью метода `setRGB()` все видеоклипы можно было бы только установить в один цвет. Метод `setTransform()` наиболее полезен для внесения изменений в растровые изображения, содержащие много цветов, которые нельзя изменять индивидуально. Метод `setTransform()` позволяет вносить малейшие изменения, почти такие же, как вносятся посредством панели Advanced Effect (Расширенный эффект). (На панели инспектора свойств в поле Color выберите опцию Advanced, а затем щелкните на кнопке Settings (Настройки).)

Прежде чем использовать методы `setTransform()` и `getTransform()`, необходимо создать объект преобразования, имеющий восемь свойств, соответствующих восьми полям ввода данных панели Advanced Effect.

И панель Advanced Effect, и объект преобразования содержат четыре пары чисел, соответствующих красному, зеленому и синему цветам и прозрачности. Первое число каждой пары, *процент преобразования*, является целым числом в диапазоне от -100 до 100, которое служит процентным множителем существующего цвета. Второе число каждой пары, *сдвиг*, является целым числом в диапазоне от -255 до 255 и представляет собой значение цвета. Вы берете результат процентного умножения и добавляете к нему значение сдвига цвета. В виде равенства это выглядит следующим образом:

```
newColor = ( oldColor x transformationPercentage ) + offset
```

Это равенство фактически показано в каждой из четырех строк панели Advanced Effect.

Множителями процентного преобразования являются **ra**, **rb** и **aa** соответственно для красного, зеленого, синего цветов и прозрачности.

Значениями сдвига цвета являются **rb**, **gb**, **bb** и **ab** соответственно для красного, зеленого, синего цветов и прозрачности.

В приведенной ниже строке кода создается объект преобразования с помощью литерала объекта:

```
myTransform = {ra:100,rb:255,ga:100,gb:255,ba:100,gb:255,aa:100,ab:255};
```

После создания объекта преобразования его можно применить к любому экземпляру класса Color следующим образом:

```
myColor.setTransform(myTransform);
```

Цвет видеоклипа соответствует изменениям myColor.

Несколько операций `setTransform()` не являются накопительными. Каждое преобразование выполняется применительно к исходным значениям цветов, назначенных в процессе разработки или с помощью прикладного программного интерфейса рисования.

Метод `getTransform()` возвращает копию текущего объекта преобразования — набор, использовавшийся при выполнении последнего метода `setTransform()`.

ОБЪЕКТ KEY

Объект Key (Клавиша) представляет ввод данных пользователем с клавиатуры. Во Flash MX в этой области внесены существенные усовершенствования, а именно — приемники. Приемники, доступные для событий `onKeyDown` и `onKeyUp`, означают, что все нажатия клавиш можно охватить без необходимости проверки нажатия клавиши в каждом кадре. Вместо этого можно зарегистрировать объект для приема уведомления о событии при нажатии клавиши. Эта процедура требует меньшей нагрузки на процессор. Проверка каждого кадра посредством события кнопки `on(keyPress)` или событий видеоклипов `onClipEvent(keyDown)` либо `onClipEvent(keyUp)` может дать меньшее время отклика.

Объект Key имеет всего четыре метода: `getAscii()`, `getCode()`, `isDown()` и `isToggled()`.

Совет

Описание четырех методов объекта Key и примеры использования методов класса Key с приемниками и без них приведены в главе 16 "Взаимодействие, события и установление последовательности", там же описывается событие `on(KeyPress)`.

Восемнадцать констант, таких как `Key.BACKSPACE` и `Key.TAB`, позволяют ссылаться на часто используемые клавиши без указания их числовых кодов. Например:

```
if (Key.isDown(Key.UP)) {  
    // затем выполните операции  
}
```

Листинги методов и константы, связанные с объектом `Key`, приведены в табл. 20.3.

ТАБЛИЦА 20.3. МЕТОДЫ И КОНСТАНТЫ ОБЪЕКТА `KEY`

Имя	Метод/ свойство/ событие	Формат	Описание
<code>Key.addListener</code>	Метод	<code>Key.addListener(myObject)</code>	Регистрирует объект <code>myObject</code> для получения уведомления об активизации методов <code>onKeyDown</code> и <code>onKeyUp</code>
<code>Key.getAscii</code>	Метод	<code>Key.getAscii()</code>	Возвращает значение ASCII последней нажатой клавиши
<code>Key.getCode</code>	Метод	<code>Key.getCode()</code>	Возвращает код последней нажатой клавиши
<code>Key.isDown</code>	Метод	<code>Key.isDown(код_символа)</code>	Возвращает значение <code>true</code> , если пользователь нажимает клавишу, кодом которой является <code>код_символа</code>
<code>Key.isToggled</code>	Метод	<code>Key.isToggled(код_символа)</code>	Возвращает значение <code>true</code> , если клавиша, кодом которой является <code>код_символа</code> , является активной. На компьютерах PC <code>код_символа</code> клавиши <code><Caps Lock></code> является 20; <code>код_символа</code> клавиши <code><Num Lock></code> — 144, а <code>код_символа</code> клавиши <code><Scroll Lock></code> — 145. Для клавиши <code><Caps Lock></code> можно использовать константу <code>Key.CAPSLock</code>
<code>Key.removeListener</code>	Метод	<code>Key.removeListener(myObject)</code>	Если объект <code>myObject</code> ранее был зарегистрирован с помощью метода <code>Key.addListener</code> , то этот объект удаляется из списка приемников
<code>Key.BACKSPACE</code>	свойство	<code>Key.BACKSPACE</code>	Константа, соответствующая значению кода клавиши <code><Backspace></code> (8)
<code>Key.CAPSLock</code>	свойство	<code>Key.CAPSLock</code>	Константа, соответствующая значению кода клавиши <code><Caps Lock></code> (20)
<code>Key.CONTROL</code>	свойство	<code>Key.CONTROL</code>	Константа, соответствующая значению кода клавиши <code><Control></code> (17)

Имя	МЕТОД/ свойство/ СОБЫТИЕ	ФОРМАТ	ОПИСАНИЕ
<code>Key.DELETEKEY</code>	Свойство	<code>Key.DELETEKEY</code>	Константа, соответствующая значению кода клавиши <Delete> (46)
<code>Key.DOWN</code>	Свойство	<code>Key.DOWN</code>	Константа, соответствующая значению кода клавиши <↓> (40)
<code>Key.END</code>	Свойство	<code>Key.END</code>	Константа, соответствующая значению кода клавиши <End> (35)
<code>Key.ENTER</code>	Свойство	<code>Key.ENTER</code>	Константа, соответствующая значению кода клавиши <Enter> (13)
<code>Key.ESCAPE</code>	Свойство	<code>Key.ESCAPE</code>	Константа, соответствующая значению кода клавиши <Escape> (27)
<code>Key.HOME</code>	Свойство	<code>Key.HOME</code>	Константа, соответствующая значению кода клавиши <Home> (36)
<code>Key.INSERT</code>	Свойство	<code>Key.INSERT</code>	Константа, соответствующая значению кода клавиши <Insert> (45)
<code>Key.LEFT</code>	Свойство	<code>Key.LEFT</code>	Константа, соответствующая значению кода клавиши <←> (37)
<code>Key.PGDN</code>	Свойство	<code>Key.PGDN</code>	Константа, соответствующая значению кода клавиши <Page Down> (34)
<code>Key.PGUP</code>	Свойство	<code>Key.PGUP</code>	Константа, соответствующая значению кода клавиши <Page Up> (33)
<code>Key.RIGHT</code>	Свойство	<code>Key.RIGHT</code>	Константа, соответствующая значению кода клавиши <→> (39)
<code>Key.SHIFT</code>	Свойство	<code>Key.SHIFT</code>	Константа, соответствующая значению кода клавиши <Shift> (16)
<code>Key.SPACE</code>	Свойство	<code>Key.SPACE</code>	Константа, соответствующая значению кода клавиши <Пробел> (32)
<code>Key.TAB</code>	Свойство	<code>Key.TAB</code>	Константа, соответствующая значению кода клавиши <Tab> (9)
<code>Key.Up</code>	Свойство	<code>Key.Up</code>	Константа, соответствующая значению кода клавиши <T> (40)
<code>Key.onKeyDown</code>	Событие	<code>myObject.onKeyDown = function() {};</code>	Запускается при нажатии клавиши
<code>Key.onKeyUp</code>	Событие	<code>myObject.onKeyUp = function() {};</code>	Запускается при отпуске клавиши

Для определения кода определенной клавиши с помощью приведенного ниже фрагмента программы можно создать специальный тестер (он содержится на компакт-диске в файле `keytester fla`).

```
_root.createTextField("myTextField", 1, 50, 50, 100, 50);
myTextField.variable = "к";
myTextField.setNewTextFormat(new TextFormat(null, 20, 0x000000, true));
_root.onEnterFrame = function () {
    кс = String.fromCharCode(Key.getCode()) + " : " + Key.getCode();
};
```

При нажатии клавиши ее числовой код будет отображен в текстовом поле.

Если вы хотите протестировать коды клавиш в среде разработки, то при воспроизведении фильма выполните команду **Control⇒Disable Keyboard Shortcuts** (**Управление⇒Отключить комбинации клавиш**). Это позволит приведенному выше коду охватывать такие клавиши, как `<Backspace>` и `<Tab>`, которые обычно "захватываются" функцией комбинаций клавиш.

ОБЪЕКТ MOUSE

Объект **Mouse** (Мышь) представляет ввод данных пользователем посредством мыши. Подобно объекту **Key**, объект **Mouse** во Flash MX усовершенствован благодаря приемникам `onMouseMove`, `onMouseDown` и `onMouseUp`.

Совет

События и приемники, связанные с мышью, были описаны в главе 16 "Взаимодействие, события и установление последовательности".

Помимо описанных в главе 16 "Взаимодействие, события и установление последовательности" возможностей, связанных с событиями, объект **Mouse** имеет два метода: `show()` и `hide()`. Метод `hide()` позволяет скрыть стандартный курсор мыши и заменить его пользовательским курсором.

Предположим, что вы хотите, чтобы курсор принял вид волшебной палочки. Создайте клип, содержащий графическое изображение волшебной палочки. Если именем экземпляра этого клипа является `magicWand`, код будет выглядеть следующим образом:

```
Mouse.hide();
_root.onEnterFrame = function () {
    magicWand._x = _root._xmouse;
    magicWand._y = _root._ymouse;
};
```

Чтобы вернуться к стандартному виду курсора, воспользуйтесь таким кодом:

```
magicWand._visible = false;
Mouse.show();
```

ОБЪЕКТ SELECTION

Объект **Selection** (Выделение) управляет редактированием выделенного текста и фокусировкой. В этой области во Flash MX внесены два усовершенствования: приемники и имена экземпляров кнопок и текстовых полей.

Совет

Приемники и события объекта `selection` описаны в главе 16 "Взаимодействие, события и установление последовательности", там же приведена подробная информация о выборе и фокусировке.

Помимо возможностей, связанных с событиями, объект **Selection** имеет три метода, связанных с выделенным в данный момент текстом (с *диапазоном выделения*). Каждый из этих методов возвращает в выделенный текст индекс. Индекс отсчитывается от нуля. Это означает, что первым положением будет 0, вторым положением — 1 и т.д.

Метод `getBeginIndex()` возвращает индекс начала диапазона выделения. Метод `getEndIndex()` возвращает индекс конца диапазона выделения.

В обоих случаях, если метод не находит достоверный индекс (например, в связи с тем, что в данный момент нет активного диапазона выделения), он возвращает значение -1.

Метод `getCaretIndex()` возвращает индекс положения мигающего курсора. Если мигающий курсор не отображен, метод возвращает значение -1.

Например, чтобы выяснить, находится ли курсор в конце выделенного текстового поля, воспользуйтесь таким кодом:

```
if ( Selection.getCaretIndex() == Selection.getEndIndex() )  
    // выполните какие-то операции
```

КЛАСС SOUND

Класс **Sound** (Звук) представляет звуки: музыку, звуковые эффекты или речь. Звуковые возможности можно разделить на пять основных категорий: создание звукового объекта, загрузка звуков, получение данных о звуке, управление звуками и использование событий, связанных со звуком.

Совет

События, связанные со звуком, описаны в главе 16 "Взаимодействие, события и установление последовательности". В этой главе будет рассмотрено несколько примеров их использования.

СОЗДАНИЕ ОБЪЕКТА SOUND

Новый объект **Sound** можно создать двумя способами. При одном из них создается звуковой объект, который будет управлять всеми звуками фильма. При другом создается звуковой объект, который управляет звуками определенного видеоклипа. Данные способы проиллюстрированы ниже.

```
allSounds = new SoundO; // управляет всеми звуками фильма  
mcSounds = new Sound("mc1"); // управляет звуками видеоклипа "mc1"
```

ЗАГРУЗКА и ПРИСОЕДИНЕНИЕ ЗВУКОВ



Звуки можно присоединить из библиотеки или загрузить внешние MP3-файлы. Третья возможность заключается в присоединении звукового потока с помощью метода `MovieClip.attachAudio()`.

Возможность загрузки внешних звуковых файлов является нововведением Flash MX и применима только к MP3-файлам. Присоединение звуков из библиотеки приводит к увеличению размеров SWF-файлов и, следовательно, к увеличению первоначальной задержки. Однако присоединенные звуки при выполнении метода `Sound.start` начинают воспроизводиться практически немедленно. Загрузка внешних звуков затрудняет **синхронизацию**, но позволяет сэкономить на размере SWF-файла.

Flash MX поддерживает разные типы сжатия: **ADPCM**, **MP3**, **Raw** и **Speech**. Это может повлиять и на синхронизацию. Чтобы изменить сжатие звука или увидеть, насколько большим будет звуковой файл, воспользуйтесь диалоговым окном **Sound Properties** (Свойства звука).

Совет

Диалоговое окно **Sound Properties** описано в главе 8 "Использование звука".

Внешние MP3-файлы загружают с помощью метода `loadSound()` следующим образом:

```
mySound.loadSound("techno1.mp3", false);
```

Первый параметр, которым является имя подлежащего загрузке звукового файла, указывается в виде URL. Ниже приведен пример указания полного пути к локальному файлу `D:\sound\techno.mp3`.

```
mySound.loadSound("file:///D:/sound/techno.mp3", 1);
```

Указание имени файла в виде имени домена:

```
mySound.loadSound("http:www.somedomain.com/somesound.mp3", 1);
```

Указание имени файла в виде IP-адреса:

```
mySound.loadSound("http://205.188.234.33:8006", 1);
```

Второй параметр, передаваемый методу `loadSound()`, определяет, загружается ли звук в виде *потока* (значение `true` или `1`) или в виде *события*, связанного со звуком (значение `false` или `0`).

Воспроизведение потокового звука начинается, как только в буфере Flash-плеера будет содержаться количество данных, достаточное для воспроизведения звукового файла в течение определенного количества секунд, указанных в свойстве `_soundbuftime` (по умолчанию — 5 секунд). Это время можно изменить следующим образом:

```
_soundbuftime = 15; // в буфере содержится звук, достаточный для  
                    // воспроизведения в течение 15 секунд
```

Событийный звук не начнет воспроизводиться до полного завершения загрузки.

При работе с потоковым звуком не нужно указывать момент начала его воспроизведения — это происходит автоматически. Так, приведенные ниже две строки кода временной шкалы начнут воспроизведение звука.

```
mySound = new Sound();  
mySound.loadSound("cantata.mp3", true); // потоковый звук
```

С другой стороны, для начала воспроизведения событийного звука требуется явное обращение к методу `start()`. Поскольку событийный звук не начнет воспроизводиться до полного завершения загрузки, то обычный подход заключается в помещении метода `start()` в обработчик события `onLoad`. Этот обработчик будет активизироваться при полной загрузке звука, иницируя исполнение метода `start()`, как показано ниже.

```
mySound = new Sound(this);  
mySound.onLoad = function() {  
    this.start();  
};  
mySound.loadSound("cantata.mp3", 0); // событийный звук
```

Обработчик события `onLoad` необходимо определить до загрузки звука.

Обработчик события `onLoad` активизируется и в том случае, когда звук не смог загрузиться. Для выполнения проверки на наличие сбоя используйте параметр, передаваемый в функцию обратного вызова `onLoad`. Этот параметр является булевой величиной, которая при успешном выполнении имеет значение `true`, а при сбое — `false`. Например:

```
mySound.onLoad = function(success) {  
    if (success) {  
        this.start();  
    }  
    else { // здесь обработать ошибку  
    }  
};
```

Звуки из библиотеки присоединяют с помощью метода `attachSound()` следующим образом:

```
mySound.attachSound("ByeBye.wav");
```

Единственным параметром метода `attachSound()` является идентификатор связи звукового файла в библиотеке. Чтобы начать воспроизведение присоединенного звука, необходимо выполнить метод `start()`.

Есть еще и третий потенциальный источник аудиоматериала, который может быть связан со звуковым объектом. Это потоковое аудио в реальном масштабе времени, происходящее либо из локального источника, такого как микрофон, либо из Flash Communication Server.

Ниже приведен пример полной рабочей программы, в которой создается видеоклип (строка 1), назначается поток звука из локального микрофона в переменную (строка 2), поток звука присоединяется к видеоклипу (строка 3), создается звуковой объект, управляющий всеми звуками видеоклипа (строка 4). Затем звуковой объект используется для установки громкости звука в значение 0, т.е. звук полностью заглушается (строка 5).

```
1: createEmptyMovieClip("aud", 1);
2: inputMic = Microphone.getO; // захват потока звука
3: aud.attachAudio(inputMic); // присоединение потока звука к видеоклипу
4: mySound = new Sound("aud"); // управление всеми звуками в видеоклипе "aud"
5: mySound.setVolume(0); // обнуление громкости звука
```

Совет

Возникли проблемы при воспроизведении звука, загруженного с помощью метода `MovieClip.loadMovie()`? Обратитесь к разделу "Возможные проблемы" в конце этой главы.

УПРАВЛЕНИЕ ЗВУКОМ

Вы можете управлять звуком, а именно: начинать и заканчивать его воспроизведение, регулировать звук, баланс динамиков и разделение на стереоканалы.

Перед началом воспроизведения звука можно указать точку начала воспроизведения звукового файла (в секундах).

Есть еще и второй, опциональный параметр, который указывает количество циклов воспроизведения файла.

Метод `start()` имеет такой формат:

```
mySound.start(второйСдвиг, цикл)
```

В приведенном ниже примере звук начинается через 1 секунду и воспроизводится 10 раз. Параметр сдвига не вызывает задержку — Flash-плеер просто пропускает определенное количество звука.

```
mySound.start(1,10);
```

Событие `onSoundComplete` не будет активизировано до тех пор, пока звук не закончит циклически воспроизводиться указанное количество раз.

Метод `stop()` имеет два формата. При выполнении без параметров он останавливает все звуки, которыми управляет определенный звуковой объект. Например:

```
mySound.stop(); // остановка всех звуков, которыми управляет объект mySound
```

Звуки, находящиеся под управлением определенного звукового объекта, определяются в зависимости от того, как был создан этот объект. Например:

```
mySound = new Sound(); // управляет всеми звуками фильма
mySound.stop(); // останавливает все звуки фильма
mySound = new Sound("mc1"); // управляет всеми звуками видеоклипа "mc1"
mySound.stop(); // останавливает все звуки видеоклипа "mc1"
```

Ранее в этой главе, в подразделе "Создание объекта Sound", говорилось о том, что звуковой объект может управлять либо всеми звуками фильма, либо только звуками определенного видеоклипа.

Для остановки всех звуков фильма можно также воспользоваться глобальной функцией `stopAllSounds()`.

Метод `stop()` имеет опциональный параметр, который используется только для звуков, присоединенных с помощью метода `attachSound()`. Этим параметром является тот же идентификатор связи, который первоначально использовался для присоединения звука:

```
mySound.attachSound("ByeBye");
mySound.stop("ByeBye");
```

При использовании метода `stop()` с параметром идентификатора связи звуковой объект, к которому применяется метод `stop()`, не обязательно должен быть тем же самым объектом, к которому присоединен звук (в предыдущем примере это был объект `mySound`). Однако он должен быть объектом, который управляет теми же звуками, что и объект `mySound`. Так, в приведенном ниже примере объекты `globalSound1` и `globalSound2` управляют всеми звуками фильма. Следовательно, объект `globalSound2` можно использовать для остановки звука, присоединенного к объекту `globalSound1`:

```
globalSound1 = new SoundO;
globalSound2 = new Sound();
globalSound1.attachSound("Techno.mp3");
globalSound2.attachSound("ByeBye");
globalSound1.start(0,2);
globalSound2.start();
globalSound2.onSoundComplete = function() {
    globalSound2.stop("Techno.mp3");
}
```

Та же самая схема используется и в том случае, когда оба звуковых объекта управляют звуками одного и того же видеоклипа.

Громкость устанавливается с помощью метода `setVolume()`, который в качестве параметра принимает число в диапазоне от 0 до 100 включительно. По умолчанию используется значение 100. Например, в приведенной ниже строке устанавливается громкость звука, равная половине полной возможной громкости.

```
mySound.setVolume(50);
```

Баланс динамиков, т.е. относительная громкость каждого динамика, регулируется с помощью метода `setPan()`. Этот метод принимает один аргумент — число в диапазоне от -100 до 100 включительно. Если параметр отрицательный, то эта величина вычитается из значения громкости правого динамика, тогда как громкость левого динамика остается максимально возможной. Таким образом, выражение `mySound.setPan(-50)` означает, что левый динамик будет работать с максимальной громкостью, а громкость правого динамика будет равна половине максимальной. Аналогичным образом, если параметр положительный, то соответствующее число вычитается из значения громкости левого динамика, тогда как громкость правого динамика остается максимально возможной. Таким образом, выражение `mySound.setPan(25)` означает, что правый динамик будет работать с максимальной громкостью, а громкость левого будет равна 75% от максимальной.

Максимальная громкость равна либо 100, либо любому другому значению, заданному посредством метода `setVolume()`. Таким образом, если с помощью выражения, подобного `mySound.setVolume(0)`, установить максимальное значение громкости, равное 0, то мето-

ды `setPan()` и `setTransform()` не будут выполнять никаких полезных действий: звук вообще не будет слышен. Например:

```
mySound.setPan(100); // левый динамик отключен, а правый работает с
                     // максимальной громкостью
```

При работе со стереозвуком и использовании метода `setPan()` весь звук левого динамика представляет собой левый стереоканал, а весь звук правого динамика — правый стереоканал. Разделением звука на стереоканалы управляют с помощью метода `setTransform()`.

Для применения этого метода сначала необходимо создать объект с четырьмя свойствами: `ll`, `lr`, `rl` и `rr`. В каждом свойстве содержится число от 0 до 100 включительно. Первая буква имени свойства указывает, какой динамик регулирует данное свойство. Вторая буква указывает, какой стереоканал регулирует данное свойство. Таким образом, свойство `ll` регулирует левый динамик и левый канал; свойство `lr` — левый динамик и правый канал и т.д.

В приведенном ниже примере показано создание объекта преобразования звука с помощью литерала объекта. Объект преобразования звука, будучи примененным к звуковому объекту, полностью подавит левый динамик, а в правый динамик направит 100% звука левого и правого каналов. Этого эффекта невозможно достичь с помощью метода `setPan()`, который никогда не помещает левый канал в правый динамик.

```
soundTransform = { ll : 0 , lr : 0 , rl : 100 , rr : 100 };
```

Объект преобразования звука применяется к звуковому объекту следующим образом:

```
mySound.setTransform(soundTransform);
```

ПОЛУЧЕНИЕ ДАННЫХ о ЗВУКЕ

Можно контролировать ход загрузки звука, определить, когда завершилось воспроизведение звука, а также проверить баланс динамиков и разделение на стереоканалы. Кроме того, можно выяснить, какова длительность звука и какая его часть уже была воспроизведена.



Контроль хода загрузки звука осуществляется посредством метода `getBytesLoaded()`. Скомбинировав его с методом `getBytesTotal()`, можно определить, какой процент звука был загружен:

```
percentLoaded = getBytesLoaded() / getBytesTotal() * 100;
```

В приведенном примере с помощью переменной `percentLoaded` можно определить размеры полосы хода выполнения загрузки, обеспечивая наглядное представление об этом процессе.



Предназначенные только для чтения свойства `duration` и `position` звукового объекта указывают соответственно длительность звука и его воспроизведения (в миллисекундах). Оба свойства являются свойствами получения/задания, поэтому ни с одним из них нельзя использовать метод `Object.watch()`. В приведенном ниже примере переменную `percentPlayed` можно использовать для реализации полосы воспроизведения звука.

```
percentPlayed = mySound.position / mySound.duration * 100;
```

ОБЪЕКТ STAGE



Объект `Stage` (Рабочее поле) является новым во Flash MX. Одно из его основных назначений заключается в том, чтобы разрешить другим объектам принимать событие `Stage.onResize`. Обработчик события `Stage.onResize`, как правило, используется для корректировки размеров или компоновки фильма для соответствия новому размеру окна.

Объект `Stage` имеет пять свойств, два из которых предназначены только для чтения.

- `height`. Указывает высоту в пикселях (только для чтения).
- `width`. Указывает ширину в пикселях (только для чтения).
- `showMenu`. Определяет, располагается ли в верхней части отдельного Flash-плеера панель меню `File-View-Control-Help` (Файл—Вид—Управление—Справка).
- `align`. Строковое свойство, определяющее выравнивание графического содержимого в рабочем поле. Свойство может принимать восемь значений: по верху и левому краю (TL), по верху (T), по верху и правому краю (TR), по левому краю (L), по правому краю (R), по низу и левому краю (BL), по низу (B), по низу и правому краю (BR).

Визуально значения выравнивания будут выглядеть так:

```
TL T TR
L      R
BL B BR
```

Например:

```
Stage.align = "B"; // выравнивание по низу и по середине
```

Интерпретирующая программа `ActionScript` очень снисходительно относится к посторонним элементам, указанным в строке. Она просто ищет первую из подходящих строк, как в следующем примере:

```
Stage.align = "^*&%%$#@WBF"; // устанавливает Stage.align в значение "B"
```

Если в указанной пользователем строке не находится ни одного подходящего элемента, то выравнивание объекта `Stage` задается полностью по центру (в приведенной выше текстовой "диаграмме" это пустое место по середине схемы), а свойство `Stage.align` устанавливается равным пустой строке. Например:

```
Stage.align = ""; // выравнивание полностью по центру
```

- `scaleMode`. Указывает режим масштабирования Flash-фильма в пределах рабочего поля. Это строковое свойство имеет четыре допустимых значения: `"exactFit"`, `"showAll"`, `"noBorder"` и `"noScale"`. Например:

```
Stage.scaleMode = "noScale";
```

В отдельном Flash-плеере (и в среде разработки) свойство `scaleMode` влияет на три параметра графики Flash-фильма: будет ли графическое изображение искаженным или останется пропорциональным; будет ли графическое изображение масштабироваться при изменении пользователем размеров окна плеера; и, если графика масштабируется, будет ли она изменяться так, чтобы полностью заполнять увеличенное или уменьшенное окно плеера.

`"exactFit"` приводит к искажению и заполнению рабочего поля, т.е. графическое изображение не обрезается, и его размеры изменяются так, чтобы полностью заполнить увеличенное или уменьшенное окно.

"showAll" оставляет графическое изображение пропорциональным и масштабирует его так, чтобы не было обрезки, т.е. изображение масштабируется так, чтобы полностью заполнить окно.

"noBorder" устраняет любую "пустую" границу по краям рабочего поля и масштабирует изображение в сторону увеличения. Следовательно, при уменьшении размеров окна графика может быть обрезана.

"noScale" не масштабирует изображение, следовательно, оно может быть обрезано при изменении размеров окна в обоих направлениях.

Если не задавать значения свойства `scaleMode`, то по умолчанию в отдельном Flash-плеере оно будет установлено в значение "showAll", а в среде разработки — в значение "noScale". В браузере свойство `Stage.scaleMode` по умолчанию установлено в значение "showAll".

Поведение браузера, вызываемое по умолчанию заданным свойством `scaleMode`, зависит от установок меню Scale (Масштаб) вкладки HTML диалогового окна Publish Settings (Параметры публикации).

Четыре элемента раскрывающегося меню Scale диалогового окна Publish Settings соответствуют значениям свойства `scaleMode`. Выбранный элемент влияет только на атрибуты дескрипторов HTML-файла. Они определяют, что произойдет при редактировании HTML-файла и изменении исходных атрибутов WIDTH и HEIGHT дескриптора OBJECT. Параметры масштабирования HTML не позволяют корректировать Flash-фильм "на лету" при изменении пользователем размеров окна браузера.

Совет

Подробная информация о диалоговом окне Publish Settings приведена в главе 30 "Оптимизация, публикация и экспортирование фильмов".

Параметры, заданные во вкладке HTML, и значения свойства `Stage.scaleMode` взаимосвязаны. И те и другие могут зависеть от браузера и платформы (например, Windows и Mac). В целом, если свойство `Stage.scaleMode` установлено в значение "noScale", а в раскрывающемся меню Scale вкладки HTML выбран элемент No scale или Default (Show All), то вы можете изменить значения параметров WIDTH и HEIGHT дескриптора OBJECT HTML-файла, но при этом SWF-файл при изменении размеров не исказится.

В табл. 20.4 приведено несколько параметров и показано, что произойдет в браузере Internet Explorer 5.5, работающем под Windows, при изменении атрибутов WIDTH и HEIGHT дескриптора OBJECT HTML-файла. Например, первая строка таблицы говорит о том, что если свойство `Stage.scaleMode` установлено в значение `exactFit`, а в диалоговом окне Publish Settings из раскрывающегося меню Scale вкладки HTML выбран элемент No scale, Exact fit или Default (Show All), то графическое изображение будет масштабировано так, чтобы соответствовать параметрам WIDTH и HEIGHT дескриптора OBJECT HTML-файла и при необходимости будет искажено. С другой стороны, если из раскрывающегося меню Scale вкладки HTML выбран элемент No Border, то графическое изображение будет масштабировано пропорционально (искажение не допускается), чтобы оно соответствовало увеличенным значениям атрибутов WIDTH и HEIGHT дескриптора OBJECT HTML-файла. Например, если WIDTH равно 50, а HEIGHT — 100, то графическое изображение будет иметь высоту 100 пикселей. Если пропорционально измененная ширина превышает 50 пикселей, то графическое изображение будет обрезано по горизонтали.

ТАБЛИЦА 20.4. ВЗАИМОСВЯЗЬ СВОЙСТВА `STAGE.SCALEMODE` И ПАРАМЕТРОВ МАСШТАБИРОВАНИЯ ВКЛАДКИ HTML

STAGE . SCALEMODE	ПАРАМЕТР МАСШТАБИРОВАНИЯ ВКЛАДКИ HTML	РЕЗУЛЬТАТ в INTERNET EXPLORER 5.5.
"exactFit"	No scale, Exact fit или Default (ShowAll)	Непропорциональное масштабирование (форма искажается так, чтобы полностью соответствовать новой высоте и ширине)
	No Border	Пропорциональное масштабирование до больших размеров с обрезкой (при необходимости)
"noScale"	Exact fit или No Border	Пропорциональное масштабирование до больших размеров с обрезкой (при необходимости)
	No scale или Default (ShowAll)	Масштабирование отсутствует
Нет (по умолчанию)	No scale или Default (ShowAll)	Масштабирование отсутствует
	Exact fit	Непропорциональное масштабирование (форма искажается так, чтобы полностью соответствовать новой высоте и ширине)
	No Border	Пропорциональное масштабирование до больших размеров с обрезкой (при необходимости)

ОБЪЕКТ SYSTEM



Объект System представляет компьютерную систему. Он имеет всего одно документированное свойство: `capabilities`.

Совет

I Обратитесь к подразделу "Объект System. `capabilities`" выше в этой главе.

КЛАССЫ TEXTFIELD И TEXTFORMAT

Во Flash 5 возможности `ActionScript` управлять текстовыми полями и контролировать их были ограниченными. Текстовые поля можно было создавать только вручную посредством графического пользовательского интерфейса, и практически все свойства текстовых полей тоже задавались вручную, посредством панели Text Options (Параметры текста). Текстовые поля не были объектами и не имели имен экземпляров.



Во Flash MX текстовые поля были полностью интегрированы в среду `ActionScript`. Все, что можно сделать с текстовыми полями вручную посредством графического пользовательского интерфейса, теперь можно сделать и с помощью `ActionScript`.

Совет

Чтобы познакомиться с текстовыми полями и узнать, какие операции можно выполнять посредством графического пользовательского интерфейса, обратитесь к главе 5 "Работа с текстом".

Существуют четыре возможных способа динамического создания текстового поля. Первые два используются для любого динамически создаваемого текстового поля. Возможно, в зависимости от ваших требований, вам понадобятся последние два способа. Итак, приступим.

1. Создайте текстовое поле.
2. Задайте одно или несколько свойств поля.
3. Создайте объект `TextFormat` и примените его к полю.
4. Создайте и разместите компонент полосы прокрутки так, чтобы он "приклеился" к текстовому полю.

ДИНАМИЧЕСКОЕ СОЗДАНИЕ ТЕКСТОВОГО поля

Текстовое поле можно создать динамически с помощью метода `createTextField` класса `MovieClip`. Формат будет таким:

```
myClip.createTextField ("myTextField", глубина, x, y, ширина, высота)
```

где `myClip` — это видеоклип, в котором создается новое текстовое поле; `myTextField` — имя экземпляра нового текстового поля; *глубина* — показатель глубины текстового поля; *x* и *y* — соответствующие координаты текстового поля, измеряемые в пикселях относительно точки регистрации видеоклипа `myClip`; *ширина* и *высота* — ширина и высота текстового поля (в пикселях).

Например:

```
myClip.createTextField ("myTextField", 2, 100, 100, 25, 12);
```

Данной строкой кода создается новое пустое текстовое поле с именем `myTextField`, являющееся дочерним для видеоклипа `myClip`. Это поле имеет показатель глубины 2 и расположено на расстоянии 100 пикселей вправо и 100 пикселей вниз относительно точки регистрации видеоклипа `myClip`. Тестовое поле имеет ширину 25 пикселей и высоту 12 пикселей.

Вновь созданное текстовое поле имеет ряд свойств. Например, `Rash` устанавливает свойства `_x` и `_y` текстового поля равными соответствующим координатам, передаваемым методу `createTextField()`. Аналогичным образом, свойства `_width` и `_height` устанавливаются равными параметрам ширины и высоты, передаваемым методу `createTextField()`. Свойство `type` устанавливается в значение `dynamic`. Все вновь создаваемые текстовые поля по умолчанию будут динамическими, если свойство `type` явным образом не будет установлено в значение `input`.

Полный перечень свойств, методов и событий текстовых полей представлен в табл. 20.5.

ТАБЛИЦА 20.5. СВОЙСТВА, МЕТОДЫ и СОБЫТИЯ ТЕКСТОВЫХ ПОЛЕЙ

Имя	МЕТОД/ СВОЙСТВО/ СОБЫТИЕ	ФОРМАТ	ОПИСАНИЕ
<code>addListener</code>	Метод	<code>myTextField.addListener(myObject)</code>	Регистрирует объект <code>myObject</code> для получения уведомления о наступлении событий <code>onChanged</code> и <code>onScroller</code> , связанных с текстовым полем <code>myText.Field</code>
<code>getDepth</code>	Метод	<code>myTextField.getDepth()</code>	Возвращает глубину текстового поля <code>myText.Field</code>
<code>getFontList</code>	Метод	<code>TextField.getFontList()</code>	Возвращает массив имен шрифтов. (Заметьте, <code>TextField</code> , а не <code>myText.Field</code> .)

Имя	Метод/ СВОЙСТВО/ СОБЫТИЕ	ФОРМАТ	ОПИСАНИЕ
getNewText Format	Метод	<code>myTextField.getNew TextFormat()</code>	Возвращает копию объекта <code>TextFormat</code> , примененного к новому тексту, вставленному в текстовое поле <code>myTextField</code> вручную или с помощью метода <code>replaceSel()</code>
removeListe ner	Метод	<code>myTextField.remove Listener(myObject)</code>	Удаляет объект <code>myObject</code> из массива приемников, связанных с текстовым полем <code>myTextField</code> , т.е. отменяет <code>myTextField.addListener(myOb ject)</code>
RemoveText Field	Метод	<code>myTextField.Remove TextField()</code>	Удаляет текстовое поле <code>myTextField</code> , если оно было создано с помощью метода <code>MovieClip.createTextField()</code>
setNewText Format	Метод	<code>myTextField.setNew TextFormat(myText Format)</code>	Задаёт <code>myTextFormat</code> в качестве объекта текстового формата нового текста, вставленного в текстовое поле <code>myTextField</code> вручную либо с помощью метода <code>replaceSel()</code>
replaceSel	Метод	<code>myTextField.replace Sel (новый_Текст)</code>	Заменяет строкой новый_Текст любой выделенный текст
setText Format	Метод	<code>myTextField.set TextFormat(myText Format)</code>	Назначает объекту <code>myTextFormat</code> для текста, уже находящегося в текстовом поле <code>myTextField</code>
_alpha	Свойство	<code>myTextField._alpha</code>	Значение прозрачности текстового поля <code>myTextField</code> ; целое число от 0 до 100
autoSize	Свойство	<code>myTextField.auto Size</code>	Строка, указывающая анкерную точку текстового поля <code>myTextField</code> при автоматическом изменении его размеров, чтобы вместить текст (табл. 20.6)
background	Свойство	<code>myTextField.back ground</code>	Булева величина, отображающая (true) или скрывающая (false) фоновую заливку
background Color	Свойство	<code>myTextField.back groundcolor</code>	Указывает цвет фоновой заливки. Целое число, часто в шестнадцатеричном формате, например <code>0xFF0000</code>
border	Свойство	<code>myTextField.border</code>	Булева величина, указывающая на наличие или отсутствие границ текстового поля
borderColor	Свойство	<code>myTextField.border Color</code>	Указывает цвет границы. Целое число, часто в шестнадцатеричном формате, например <code>0xFF0000</code>

Имя	МЕТОД/ СВОЙСТВО/ СОБЫТИЕ	ФОРМАТ	ОПИСАНИЕ
<code>bottomScroll</code>	Свойство	<code>myTextField.Scroll</code>	Целое число, предназначенное только для чтения, которое указывает самую нижнюю видимую строку текстового поля <code>myTextField</code>
<code>embedFonts</code>	Свойство	<code>myTextField.embedFonts</code>	Булева величина, указывающая, используются ли в текстовом поле контуры внедренных (<code>true</code>) или системных (<code>false</code>) шрифтов
<code>_focusrect</code>	Свойство	<code>myTextField._focusrect</code>	Булева величина, указывающая, обрамлено ли текстовое поле желтым прямоугольником, когда оно получает фокусировку
<code>_highquality</code>	Свойство	<code>TextField._highquality</code>	Ссылается на глобальную переменную <code>_highquality</code> , что позволяет сглаживать растровые изображения и устранять неровности контуров. Целое число. 0 означает "не применяется"; 1 означает "устранять неровности и сглаживать растровые изображения, если в фильме нет анимации" (значение по умолчанию); 2 означает "сглаживать растровые изображения и устранять неровности контуров"
<code>_height</code>	Свойство	<code>myTextField._height</code>	Целое число, предназначенное для чтения и записи, указывающее высоту текстового поля <code>myTextField</code> в пикселях. Влияет только на размер ограничительного окошка, но не на размер шрифта
<code>hscroll</code>	Свойство	<code>myTextField.hscroll</code>	Целое число, указывающее текущее горизонтальное положение прокрутки текстового поля <code>myTextField</code> в пикселях
<code>html</code>	Свойство	<code>myTextField.html</code>	Булева величина, указывающая, содержит ли текстовое поле <code>myTextField</code> HTML-текст. Должна предшествовать свойству <code>myTextField.htmlText</code>
<code>htmlText</code>	Свойство	<code>myTextField.htmlText</code>	Строка, содержащая текст формата HTML текстового поля <code>myTextField</code>
<code>length</code>	Свойство	<code>myTextField.length</code>	Целое число, предназначенное только для чтения, которое указывает число символов в текстовом поле <code>myTextField</code>
<code>maxChars</code>	Свойство	<code>myTextField.maxChars</code>	Целое число. Когда в текстовом поле <code>myTextField</code> содержится данное количество символов, Flash отключает вставку символов вручную

Имя	Метод/ свойство/ СОБЫТИЕ	ФОРМАТ	ОПИСАНИЕ
<code>maxhscroll</code>	Свойство	<code>myTextField. maxhscroll</code>	Целое число, предназначенное только для чтения, которое указывает максимально возможное значение в пикселях свойства <code>myTextField.hscroll</code> заданное для текущего текста
<code>maxscroll</code>	Свойство	<code>myTextField. maxscroll</code>	Целое число, предназначенное только для чтения, которое указывает максимально возможное значение в пикселях свойства <code>myTextField.scroll</code> , заданное для текущего текста
<code>multiline</code>	Свойство	<code>myTextField. multiline</code>	Булева величина, указывающая, может ли текстовое поле <code>myTextField</code> содержать несколько строк
<code>_name</code>	Свойство	<code>myTextField._name</code>	Строка, являющаяся именем экземпляра, например <code>"myTextField"</code>
<code>_parent</code>	Свойство	<code>myTextField._parent</code>	Ссылка на видеоклип или кнопку, являющуюся родительской для текстового поля <code>myTextField</code>
<code>password</code>	Свойство	<code>myTextField. password</code>	Булева величина, указывающая, предназначено ли текстовое поле <code>myTextField</code> для ввода пароля. Если она имеет значение <code>true</code> , то символы вводимого пароля Flash отображает в виде звездочек
<code>_quality</code>	Свойство	<code>TextField._quality</code>	Глобальное строковое свойство, указывающее качество отображения фильма. Принимает следующие значения: <code>LOW</code> (контуры графики не сглажены, растровые изображения не сглажены); <code>MEDIUM</code> (контуры графики сглаживаются с помощью пиксельной сетки 2x2, растровые изображения не сглаживаются; подходит для фильмов без текста); <code>HIGH</code> (контуры графики сглаживаются с помощью пиксельной сетки 4x4, растровые изображения сглаживаются, если в фильме нет графики; данное значение используется по умолчанию); <code>BEST</code> (контуры графики сглаживаются с помощью пиксельной сетки 4x4, растровые изображения всегда сглаживаются)

Имя	Метод/ свойство/ СОБЫТИЕ	ФОРМАТ	ОПИСАНИЕ
<code>restrict</code>	Свойство	<code>myTextField.Restrict</code>	Строка, указывающая набор символов, которые пользователь может ввести в текстовое поле <code>myTextField</code> . Символ тире (-) указывает диапазон. Символ вставки (^) означает, что следующие символы исключены, а предыдущие разрешены. В следующем примере строчные буквы разрешены, однако исключена строчная буква w: <code>myTextField.restrict = "a-z^w";</code>
<code>_rotation</code>	Свойство	<code>myTextField._rotation</code>	Целое число, предназначенное для чтения и записи, указывающее текущее количество градусов вращения текстового поля <code>myTextField</code>
<code>scroll</code>	Свойство	<code>myTextField.scroll</code>	Целое число, предназначенное для чтения и записи, указывающее текущее вертикальное положение прокрутки текстового поля <code>myTextField</code> , измеряемое в строках текста
<code>selectable</code>	Свойство	<code>myTextField.Selectable</code>	Булева величина, указывающая, доступно ли текстовое поле <code>myTextField</code> для выделения
<code>_soundbuf time</code>	Свойство	<code>myTextField._soundbuftime</code>	Ссылается на глобальную переменную <code>_soundbuftime</code> . Глобальным образом устанавливает и получает количество времени в секундах, которое звук должен содержаться в буфере перед началом потокового воспроизведения. Целое число, по умолчанию равно 5
<code>tabEnabled</code>	Свойство	<code>myTextField.tabEnabled</code>	Булева величина, указывающая, включено ли текстовое поле <code>myTextField</code> в автоматический порядок перехода по клавише табуляции
<code>tabIndex</code>	Свойство	<code>myTextField.tabIndex</code>	Указывает порядок перехода по клавише табуляции для текстового поля <code>myTextField</code>
<code>text</code>	Свойство	<code>myTextField.text</code>	Строка, предназначенная для чтения и записи, определяющая текст, находящийся в данный момент в текстовом поле
<code>textColor</code>	Свойство	<code>myTextField.textColor</code>	Целое число, предназначенное для чтения и записи, которое указывает цвет текущего текста в текстовом поле. Часто указывается в шестнадцатеричном формате, например 0xFF0000

Имя	МЕТОД/ СВОЙСТВО/ СОБЫТИЕ	ФОРМАТ	ОПИСАНИЕ
<code>textHeight</code>	Свойство	<code>myTextField. TextHeight</code>	Целое число, предназначенное только для чтения, которое указывает высоту фактического текста (а не ограничительного окошка)
<code>textWidth</code>	Свойство	<code>myTextField. textWidth</code>	Целое число, предназначенное только для чтения, которое указывает ширину фактического текста (а не ограничительного окошка)
<code>type</code>	СВОЙСТВО	<code>myTextField.type</code>	Строка, предназначенная для чтения и записи, dynamic или input , которая указывает, является ли текстовое поле <code>myTextField</code> полем ввода текста или динамическим текстовым полем
<code>_url</code>	Свойство	<code>myTextField._url</code>	Строка, предназначенная только для чтения, указывающая URL SWF-файла, создавшего экземпляр текстового поля
<code>variable</code>	Свойство	<code>myTextField. variable</code>	Строка, предназначенная для чтения и записи, указывающая имя переменной, связанной с текстовым полем
<code>_visible</code>	Свойство	<code>myTextField._ visible</code>	Булева величина, предназначенная для чтения и записи, которая определяет, является ли текстовое поле <code>myTextField</code> скрытым (false) или видимым (true)
<code>_width</code>	Свойство	<code>myTextField._width</code>	Целое число, предназначенное для чтения и записи, которое указывает ширину текстового поля <code>myTextField</code> в пикселях. Оно влияет только на ограничительное окошко текстового поля и не влияет на толщину границы или размер шрифта текста
<code>wordwrap</code>	Свойство	<code>myTextField. wordwrap</code>	Булева величина, предназначенная для чтения и записи, которая указывает, можно ли переносить слова в текстовом поле на новую строку
<code>_x</code>	Свойство	<code>myTextField._x</code>	Целое число, предназначенное для чтения и записи, которое указывает координату <i>x</i> текстового поля <code>myTextField</code>
<code>_xmouse</code>	Свойство	<code>myTextField._xmouse</code>	Целое число, предназначенное только для чтения, которое указывает координату <i>x</i> курсора относительно текстового поля <code>myTextField</code>
<code>_y</code>	Свойство	<code>myTextField._y</code>	Целое число, предназначенное для чтения и записи, которое указывает координату <i>y</i> текстового поля <code>myTextField</code>

Имя	Метод/ свойство/ событие	ФОРМАТ	ОПИСАНИЕ
<code>_ymouse</code>	Свойство	<code>myTextField._ymouse</code>	Целое число, предназначенное только для чтения, которое указывает координату у курсора относительно текстового поля <code>myTextField</code>
<code>_yscale</code>	Свойство	<code>myTextField._yscale</code>	Целое число, предназначенное для чтения и записи, в диапазоне от -100 до 100, которое указывает процент масштабирования текстового поля <code>myTextField</code> по вертикали
<code>onChanged</code>	Событие	<code>myTextField.onChanged = function (ИМЯ_текстового_поля) {}</code>	Активизируется при изменении текста в текстовом поле. См. главу 16 "Взаимодействие, события и установление последовательности"
<code>onKillFocus</code>	Событие	<code>myTextField.onKillFocus = function (новый_фокус) {}</code>	Активизируется при потере текстовым полем фокусировки. См. главу 16 "Взаимодействие, события и установление последовательности"
<code>onScroller</code>	Событие	<code>rayTextField.onScroller function (ИМЯ_текстового_поля) {</code>	Активизируется при изменении свойств <code>scroll</code> , <code>maxscroll</code> , <code>hscroll</code> , <code>maxhscroll</code> или <code>bottomscroll</code> текстового поля. См. главу 16 "Взаимодействие, события и установление последовательности"
<code>onSetFocus</code>	Событие	<code>myTextField.onSetFocus = function (старый_фокус) {}</code>	Активизируется при получении фокусировки данным текстовым полем. См. главу 16 "Взаимодействие, события и установление последовательности"

Значения свойства `TextField.autoresize` приведены в табл. 20.6.

ТАБЛИЦА 20.6. ЗНАЧЕНИЯ СВОЙСТВА `TEXTFIELD.AUTORESIZE`

ЗНАЧЕНИЕ	ОДНА СТРОКА	НЕСКОЛЬКО СТРОК
"none" или <code>false</code>	Размеры автоматически не изменяются	Размеры автоматически не изменяются
"left" или <code>true</code>	Расширение вправо	Расширение вниз
"center"	Расширение вправо и влево	Расширение вниз
"right"	Расширение влево	Расширение вниз

ЗАДАНИЕ СВОЙСТВ ТЕКСТОВОГО ПОЛЯ

Для нового текстового поля должно быть установлено хотя бы одно свойство, а может быть, и несколько. Как минимум, нужно выполнить одно или несколько из перечисленных ниже действий.

- Задайте для текстового поля свойство `variable`.

В связи с тем, что новые текстовые поля автоматически являются *динамическими* (в отличие от статических полей и полей ввода данных), то при задании свойства `variable` в текстовом поле сразу же начнет отображаться значение указанной переменной.

На заметку

Текстовые поля статического типа нельзя создать динамически. Однако использовать текстовые поля динамического типа можно точно так же, как и статические.

- Задайте для текстового поля свойство `text`.

Свойство `text` определяет, какой текст будет отображен в поле. И в динамических полях, и в полях ввода данных отображается текст, содержащийся в свойстве `text`.

- Создайте поле для ввода данных. Для этого установите свойство `type` равным `"input"`.

С динамическим полем или полем ввода данных можно связать и свойство `variable`, и свойство `text`, а затем отображать эти свойства попеременно.

Для текстовых полей можно задавать и другие свойства.

- `border = true` — для задания границы текстового поля;
- `multiline = true` — чтобы разрешить отображение в текстовом поле не одной, а нескольких строк.
- `wordwrap = true` — слишком длинные строки текста в текстовом поле будут автоматически переноситься на новую нижнюю строку.

СОЗДАНИЕ И ПРИМЕНЕНИЕ ОБЪЕКТА `TextFormat`

Свойства текстового поля, рассмотренные в предыдущем подразделе, не форматируют текст. Например, с их помощью не реализуется подчеркивание, полужирное и курсивное начертание текста, а также не устанавливается шрифт, размер и цвет текста. Для выполнения подобных операций необходимо создать объект `TextFormat` и применить его к текстовому полю. `TextFormat` является классом или функцией-конструктором. Для создания объекта `TextFormat` он используется вместе с ключевым словом `new` следующим образом:

```
myTextFormat = new TextFormat(шрифт, размер, цвет, полужирный, курсив,
подчеркивание, url, target, align, leftMargin, rightMargin, indent,
leading);
```

Все параметры функции `TextFormat` являются опциональными и могут быть установлены в значение `null`. Это говорит о том, что параметры не определены. Замыкающие параметры (после последнего параметра, который вы хотите задать) могут быть полностью опущены. Ниже приведен пример создания объекта `TextFormat` с полужирным текстом красного цвета.

```
myTextFormat = new TextFormat(null, null, 0xFF0000, true);
```

Свойства объекта `TextFormat` можно задать отдельно, после создания объекта, как показано ниже.

```
myTextFormat.underline = true;
```

Хотя в этом случае код не будет столь кратким, но отдельная установка свойств делает его более удобочитаемым.

ТАБЛИЦА 20.7. СВОЙСТВА И МЕТОД КЛАССА `TEXTFORMAT`

Имя	Метод/ свойство	ФОРМАТ	ОПИСАНИЕ
<code>getTextExtent</code>	Метод	<code>myTextFormat.getTextExtent</code> (<i>текст</i>)	Возвращает объект с двумя свойствами, width и height , указывающими размеры (в пунктах) строки текст при ее форматировании с помощью объекта <code>myTextFormat</code> . (При использовании внедренных шрифтов ширина (width) может быть неточной.)
<code>align</code>	Свойство	<code>myTextFormat.align</code>	Строка, указывающая выравнивание текста. Принимает значения null (нет), "left" (по левому краю), "right" (по правому краю) и "center" (по центру)
<code>blockIndent</code>	Свойство	<code>myTextFormat.blockIndent</code>	Целое число, указывающее величину отступа (в пунктах) всех строк блока текста
<code>bold</code>	Свойство	<code>myTextFormat.bold</code>	Булева величина, указывающая, применено ли к тексту полужирное начертание
<code>bullet</code>	Свойство	<code>myTextFormat.bullet</code>	Булева величина, указывающая, нужно ли форматировать текст в виде маркированного списка
<code>color</code>	Свойство	<code>myTextFormat.color</code>	Целое число, указывающее цвет текста. Часто имеет шестнадцатеричный формат, например 0XFF0000
<code>font</code>	Свойство	<code>myTextFormat.font</code>	Строка, указывающая имя шрифта, например Times New Roman
<code>indent</code>	Свойство	<code>myTextFormat.indent</code>	Целое число, указывающее величину отступа (в пунктах) от левого поля до первого символа каждого абзаца
<code>italic</code>	Свойство	<code>myTextFormat.italic</code>	Булева величина, указывающая, применено ли к тексту курсивное начертание
<code>leading</code>	Свойство	<code>myTextFormat.leading</code>	Целое число, указывающее величину междустрочного интервала (в пунктах)

Имя	МЕТОД/ СВОЙСТВО	ФОРМАТ	ОПИСАНИЕ
<code>leftMargin</code>	Свойство	<code>myTextFormat.leftMargin</code>	Целое число, указывающее величину левого поля (в пунктах)
<code>rightMargin</code>	Свойство	<code>myTextFormat.rightMargin</code>	Целое число, указывающее величину правого поля (в пунктах)
<code>tabStops</code>	Свойство	<code>myTextFormat.tabStops</code>	Массив положительных чисел, указывающих пользовательские точки перехода по клавише табуляции
<code>target</code>	Свойство	<code>myTextFormat.target</code>	Используется вместе со свойством <code>rayText - Format.url</code> . Строка, указывающая окно браузера, в котором следует отображать страницу, связанную гиперссылкой. Значения этого свойства соответствуют атрибутам HTML-дескриптора <code>target</code> , например <code>"_blank"</code> , <code>"_self"</code> , <code>"_top"</code> , <code>"_parent"</code>
<code>size</code>	Свойство	<code>myTextFormat.size</code>	Целое число, указывающее размер текста (в пунктах)
<code>underline</code>	Свойство	<code>myTextFormat.underline</code>	Булева величина, указывающая, применено ли к тексту подчеркивание
<code>url</code>	Свойство	<code>myTextFormat.url</code>	Строка, указывающая URL, с которым связан текст

Соответствие между объектом `TextFormat` и текстовым полем устанавливается посредством метода `setTextFormat` класса `TextField`. Например:

```
myTextField.setTextFormat(myTextFormat);
```

Чтобы установить связь нового объекта `TextFormat` только с новым текстом, введенным в текстовое поле (например, при вводе данных пользователем), воспользуйтесь методом `setNewTextFormat()` класса `TextField`, как показано ниже.

```
myTextField.setNewTextFormat(myTextFormat);
```

Совет

При загрузке нового текста в текстовое поле потерялось форматирование? Обратитесь к разделу "Возможные проблемы" в конце этой главы.

Дескрипторы форматирования текста, внедренные в HTML-текст

Дескрипторы форматирования текста можно внедрить в HTML-текст. Их можно использовать дополнительно или вместо объектов `TextFormat`. Внедренные дескрипторы облегчают применение атрибута к какой-то отдельной части текста. Например, в приведенном ниже файле `htmltags fla` атрибуты `LEFTMARGIN` и `INDENT` применяются к обоим абзацам HTML-текста, тогда как атрибут `BLOCKINDENT` применяется только ко второму абзацу. Имейте в виду, что строка 4 является одной строкой кода.

```

1: _root.createTextField("mytext", 1, 50, 50, 300, 100);
2: myText.html = true;
3: myText.wordwrap = true;
4: myText.htmlText = "<TEXTFORMAT LEFTMARGIN=\"12\" INDENT=\"6\">
<P ALIGN=\"LEFT\"><FONT FACE=\"_sans\" SIZE=\"20\" COLOR=\"#FF0000\">
Это первый абзац, без отступа.
</FONT></P><TEXTFORMAT BLOCKINDENT=\"30\">
<P ALIGN=\"LEFT\"><FONT FACE=\"_sans\" SIZE=\"20\"
COLOR=\"#000000\"> Это второй абзац, с блочным отступом.
</FONT></P></TEXTFORMAT></TEXTFORMAT>";

```

СОЗДАНИЕ И РАЗМЕЩЕНИЕ КОМПОНЕНТА ПОЛОСЫ ПРОКРУТКИ

Можно динамически создать и разместить компонент полосы прокрутки так, чтобы она "была привязана" к текстовому полю. Эта операция, а также другие методики, рассматриваемые в данном подразделе, продемонстрированы на примере файла `textfield fla`:

```

1: _root.createTextField("myTextField", 1, 50, 50, 100, 100);
2: myTextField.border = true;
3: myTextField.multiline = true;
4: myTextField.wordWrap = true;
5: myTextField.text = "При помещении полосы прокрутки в динамическое или" +
6: "текстовое поле ввода данных в рабочем поле полоса прокрутки "+"
7: "автоматически привязывается к ближайшей стороне поля.";
8: _root.attachMovie("FScrollBarSymbol", "sc", 2);
9: myTextFormat = new TextFormat(null, null, 0xFF0000, true);
10: myTextField.setTextFormat(myTextFormat);
11: sc._x = 150;
12: sc._y = 50;
13: sc.setSize(101);
14: sc.setScrollTarget(myTextField);

```

Совет

Подробная информация о компоненте полосы прокрутки приведена в главе 12 "Управление переменными, данными и типами данных".

Чтобы предложение `attachMovie()`, указанное в строке 8, было выполнено успешно, необходимо, чтобы компонент полосы прокрутки содержался в библиотеке с именем связи `FScrollBarSymbol`. При перетаскивании компонента полосы прокрутки с панели `Components` (Компоненты) в рабочее поле и последующем удалении экземпляра из рабочего поля символ с соответствующей связью все равно останется в библиотеке.

ЦЕННЫЕ МУЛЬТИМЕДИЙНЫЕ ОБЪЕКТЫ: VIDEO, MICROPHONE, CAMERA



Класс `video`, описывающий видео- и аудиопоток, идущий с Web-камеры в режиме реального времени, в исходной версии `Rash MX` был недокументированным. Наиболее полезен он при использовании в комбинации с приложением `Flash Communication Server MX`, которое будет кратко рассмотрено в следующем подразделе. Без сервера связи можно получать только локальный видеопоток, а не тот, который происходит из удаленного источника или сети.

Для получения локального видеопотока создайте новый объект `video`, выбрав соответствующий элемент из меню параметров (в правом верхнем углу) панели `Library` (Библиотека), а затем поместите экземпляр объекта в рабочее поле. На панели инспектора свойств задайте имя экземпляра, например `vid`. Тогда приведенный ниже код, помещенный во временную шкалу, определяет видеопоток, идущий с заданной по умолчанию локальной камеры.

```
vid.attachVideo(Camera.get());
```

Метод `Camera.get()` можно использовать либо с объектом `Video`, как в предыдущем примере, либо с объектом `NetStream`. Класс `NetStream` доступен только при работе с приложением `Flash Communication Server`.

Когда фильм пытается получить доступ к камере, Flash-плеер отображает диалоговое окно `Privacy` (Конфиденциальность), требуя от пользователя разрешить или отклонить доступ к камере. Чтобы это диалоговое окно могло открыться, размеры рабочего поля должны быть не меньше 215x138 пикселей. Когда пользователь откликается на запрос диалогового окна, обработчик события `Camera.onStatus()` возвращает информационный объект, соответствующий отклику пользователя, как показано ниже.

```
myCamera = Camera.getO;  
myVideoObject.attachVideo(myCamera);  
myCamera.onStatus = function (infoMsg) {  
    if (infoMsg.code == "Camera.Muted")  
        traceC'User denied access to the camera";  
    if (infoMsg.code == "Camera.Unmuted") {  
        traceC'User allowed access to the camera";  
    }  
};
```

Если при последнем отображении диалогового окна `Privacy` пользователь установил флажок в поле опции `Remember` (Запомнить), то Flash просто разрешит или отклонит доступ, а обработчик события `onStatus()` активизироваться не будет. Чтобы определить, разрешил или отклонил пользователь доступ к камере, без использования обработчика события `onStatus()`, воспользуйтесь свойством `muted` объекта `Camera`.

В табл. 20.8 и 20.9 представлены соответственно свойства и методы объекта `Video`.

ТАБЛИЦА 20.8. СВОЙСТВА ОБЪЕКТА VIDEO

Свойство	ОПИСАНИЕ
<code>deblocking</code>	Число, определяющее поведение фильтра деблокирования, примененного устройством сжатия потокового видео. Значение 0 (по умолчанию) позволяет устройству сжатия применять фильтр по мере необходимости. Значение 1 отключает фильтр. Значение 2 вынуждает устройство сжатия применять фильтр. Данное свойство предназначено для чтения и записи
<code>smoothing</code>	Булева величина, по умолчанию установленная в значение <code>false</code> . Указывает, нужно ли сглаживать (интерполировать) видео при его масштабировании. Плеер должен работать в режиме высокого качества. Данное свойство предназначено для чтения и записи
<code>height</code>	Высота в пикселях. Данное свойство предназначено только для чтения
<code>width</code>	Ширина в пикселях. Данное свойство предназначено только для чтения

Свойства `deblocking` и `smoothing` объекта `Video` конфигурируют видеополтеры для смягчения "блочного" отображения сильно сжатого видео. Фильтр деблокирования уменьшает производительность при воспроизведении, особенно на медленных машинах, а при работе с высокой пропускной способностью он, как правило, не нужен.

ТАБЛИЦА 20.9. МЕТОДЫ ОБЪЕКТА VIDEO

ИМЯ	ФОРМАТ	ОПИСАНИЕ
attachVideo	myVideoObject.attachVideo (источник [null])	Указывает, какой поток видео следует отображать в рамках объекта Video в рабочем поле. <i>Источник</i> — это объект <i>NetStream</i> , который воспроизводит видеопоток, или объект <i>Camera</i> , захватывающий видеопоток. Если <i>источник</i> установлен в значение null , то связь с видеообъектом удаляется
clear	myVideoObject.clear()	Удаляет изображение, отображаемое в видео объекте в текущий момент. Полезно, к примеру, когда связь с сервером прервана, а вы хотите отобразить резервные данные без скрытия видеообъекта

В табл. 20.10–20.12 представлены соответственно методы, свойства и обработчики событий класса *Camera*.

ТАБЛИЦА 20.10. МЕТОДЫ КЛАССА CAMERA

ИМЯ	ФОРМАТ	ОПИСАНИЕ
get	Camera.get ([индекс])	Возвращает видеопоток, идущий с заданной по умолчанию камеры или с камеры, определяемой параметром <i>индекс</i> . Возвращает значение null , если камера не доступна. (Если в системе установлено несколько камер, то пользователь указывает камеру по умолчанию на панели <i>Camera Settings</i> (Параметры камеры) <i>Flash-плеера</i> .)
setKeyFrameInterval	myCamera.setKeyFrameInterval (интервал_ключевых_кадров)	Целое число от 1 до 48, указывающее, какие кадры видео являются ключевыми. Все данные передаются для ключевых кадров. <i>интервал_ключевых_кадров</i> = 1 означает, что каждый кадр является ключевым; значение 2 означает, что каждый второй кадр является ключевым и т.д. По умолчанию установлено значение 15. Метод используется, в первую очередь, во <i>Flash Communication Server</i>
setLoopback	myCamera.setLoopback (compressLocalStream)	Параметр <i>compressLocalStream</i> — это булева величина, указывающая, следует ли использовать сжатый видеопоток для локального просмотра данных, передаваемых камерой. Метод используется, в первую очередь, во <i>Flash Communication Server</i>

Имя	ФОРМАТ	ОПИСАНИЕ
setMode	<code>myCamera.setMode (ширина, высота, fps [, favorSize])</code>	Задаёт атрибуты режима захвата камеры, в том числе высоту, ширину и частоту смены кадров (число кадров в секунду). <i>FavorSize</i> — это опциональная булева величина. Чтобы максимально увеличить частоту смены кадров за счёт высоты и ширины, установите параметр <i>favorSize</i> в значение <code>false</code> . Для поддержки оптимальной высоты и ширины установите параметр <i>favorSize</i> в значение <code>true</code>
setMotionLevel	<code>myCamera.setMotionLevel (чувствительность [, простой])</code>	Параметр <i>чувствительность</i> — это целое число от 0 до 100, указывающее, какое движение требуется для активизации обработчика события <code>Camera.onActivity(true)</code> . По умолчанию установлено значение 50. Параметр <i>простой</i> указывает, сколько миллисекунд должно истечь без движения до активизации обработчика события <code>Camera.onActivity(false)</code> . По умолчанию установлено значение 2000 (2 секунды)
setQuality	<code>myCamera.setQuality (полоса_пропускания, качество_кадров)</code>	Параметр <i>полоса_пропускания</i> указывает максимальную полосу пропускания (в битах в секунду) исходящего видеопотока. Параметр <i>качество_кадров</i> — это целое число от 0 до 100, управляющее качеством картинки. Если параметр <i>полоса_пропускания</i> равен 0, то видеопоток получает полосу пропускания, необходимую для поддержки значения <i>качество_кадров</i> . Если <i>качество_кадров</i> равно 0, то качество картинки корректируется в пределах параметра <i>полоса_пропускания</i>

ТАБЛИЦА 20.11. СВОЙСТВА КЛАССА CAMERA

ФОРМАТ	ОПИСАНИЕ
<code>myCamera.activityLevel</code>	Величина перемещения, обнаруживаемая камерой; число в диапазоне от 0 (отсутствие перемещения) до 100 (максимум)
<code>myCamera.bandwidth</code>	Максимальная пропускная способность, которую может использовать текущий исходящий видеопоток (в байтах). Свойство предназначено только для чтения

ФОРМАТ	ОПИСАНИЕ
<code>myCamera.currentFps</code>	Скорость захвата данных камерой (кадров в секунду). Свойство предназначено только для чтения
<code>myCamera.fps</code>	Желательная скорость захвата данных камерой (кадров в секунду). Свойство предназначено только для чтения. Задается с помощью метода <code>setMode()</code> (см. табл. 20.10)
<code>myCamera.height</code>	Текущая высота захвата (в пикселях). Свойство предназначено только для чтения
<code>myCamera.index</code>	Индекс камеры, отображаемый в массиве, возвращаемом свойством <code>Camera.names</code> . Свойство предназначено только для чтения
<code>myCamera.keyFrameInterval</code>	Целое число от 1 до 48, указывающее, какие кадры видео являются ключевыми. Все данные передаются для ключевых кадров. Значение 1 означает, что каждый кадр является ключевым; значение 2 означает, что каждый второй кадр является ключевым, и т.д. По умолчанию установлено значение 15. Используется, в первую очередь, в приложении Flash Communication Server. Свойство предназначено только для чтения
<code>myCamera.loopback</code>	Предназначенная только для чтения булева величина, которая указывает, является ли локальное представление видеопотока сжатым или несжатым
<code>myCamera.motionLevel</code>	Величина перемещения, необходимая для активизации обработчика события <code>Camera.onActivity(true)</code> . Свойство предназначено только для чтения
<code>myCamera.motionTimeOut</code>	Число миллисекунд между моментом завершения обнаружения перемещения и временем активизации обработчика событий <code>Camera.onActivity(false)</code> . Свойство предназначено только для чтения
<code>myCamera.muted</code>	Предназначенная только для чтения булева величина, которая указывает, разрешил или отклонил пользователь доступ к камере
<code>myCamera.name</code>	Имя камеры, соответствующее аппаратной части устройства. Свойство предназначено только для чтения
<code>myCamera.names</code>	Массив строк, содержащий имена всех доступных устройств захвата видео, в том числе видеокарт и камер. Свойство предназначено только для чтения
<code>myCamera.quality</code>	Число от 1 до 100, указывающее текущий уровень качества картинки. Чем сильнее сжатие, тем ниже качество. Значение 1 соответствует минимальному качеству и максимальному сжатию. Значение 100 означает максимальное качество без сжатия. Свойство предназначено только для чтения
<code>myCamera.width</code>	Текущая ширина захвата в пикселях. Свойство предназначено только для чтения

ТАБЛИЦА 20.12. ОБРАБОТЧИКИ СОБЫТИЙ КЛАССА CAMERA

Имя	ФОРМАТ	ОПИСАНИЕ
onActivity	<code>myCamera.onActivity = function(activity) { }</code>	Активируется, когда камера начинает или заканчивает обнаружение перемещения. <i>Activity</i> — это булева величина, установленная в значение <code>true</code> , когда камера начинает обнаружение движения, и в значение <code>false</code> , когда камера останавливает обнаружение
onStatus	<code>myCamera.onStatus = function(unfoObject) { }</code>	Активируется, когда пользователь разрешает или запрещает доступ к камере

В табл. 20.13–20.15 представлены соответственно методы, свойства и обработчики событий класса `Microphone`.

ТАБЛИЦА 20.13. МЕТОДЫ КЛАССА MICROPHONE

Имя	ФОРМАТ	ОПИСАНИЕ
get	<code>Microphone.get ([индекс])</code>	Возвращает заданный по умолчанию или указанный аудиопоток либо <code>null</code> , если микрофон отсутствует
setGain	<code>myMicrophone.setGain (усиление)</code>	Указывает величину в диапазоне от 0 до 100, на которую микрофон должен усилить сигнал
setRate	<code>myMicrophone.setRate (КГц)</code>	Указывает частоту в КГц, на которой микрофон должен захватывать звук
setSilenceLevel	<code>myMicrophone.setSilenceLevel (уровень [, время_простоя])</code>	Указывает уровень звука (от 0 до 100), требуемый для активизации микрофона. Можно задать опциональный параметр <i>время_простоя</i> , задающий количество миллисекунд, в течение которых микрофон является неактивным, перед тем как Flash активизирует обработчик события <code>Microphone.onActivity (false)</code> . По умолчанию этот параметр установлен в значение 2000 (2 секунды)
setUseEchoSuppression	<code>myMicrophone.setUseEchoSuppression (подавление)</code>	Параметр <i>подавление</i> — это булева величина, указывающая, нужно ли в аудиокодеке использовать функцию подавления эха

ТАБЛИЦА 20.14. СВОЙСТВА КЛАССА MICROPHONE

ФОРМАТ	ОПИСАНИЕ
<code>myMicrophone.activityLevel</code>	Величина звука, от 0 до 100, улавливаемая микрофоном
<code>myMicrophone.gain</code>	Величина, на которую микрофон усиливает сигнал перед его передачей. Значение в диапазоне от 0 до 100. По умолчанию задано значение 50

ФОРМАТ	ОПИСАНИЕ
<code>myMicrophone.index</code>	Индекс текущего микрофона
<code>myMicrophone.mute</code>	Булева величина, указывающая, разрешил или запретил пользователь доступ к микрофону
<code>myMicrophone.name</code>	Имя текущего устройства захвата звука, возвращаемое аппаратным обеспечением захвата звука
<code>myMicrophone.names</code>	Массив строк, содержащий имена всех имеющихся устройств захвата звука, в том числе звуковых карт и микрофонов
<code>myMicrophone.rate</code>	Частота захвата звука (в КГц)

ТАБЛИЦА 20.15. ОБРАБОТЧИКИ СОБЫТИЙ КЛАССА MICROPHONE

ИМЯ	ФОРМАТ	ОПИСАНИЕ
<code>onActivity</code>	<code>myMicrophone.onActivity = function(activity) {}</code>	Активизируется, когда микрофон начинает или заканчивает обнаружение звука. <i>Activity</i> — это булева величина, установленная в значение <code>true</code> , когда микрофон начинает обнаружение звука, и в значение <code>false</code> , когда микрофон заканчивает обнаружение
<code>onStatus</code>	<code>myMicrophone.onStatus = function(infoObject) {}</code>	Активизируется, когда пользователь разрешает или запрещает доступ к микрофону. Использование объекта <i>infoObject</i> аналогично его применению в обработчике событий <code>Camera.onStatus</code>

СЕТЕВЫЕ СОЕДИНЕНИЯ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ: ОБЪЕКТЫ NETCONNECTION И NETSTREAM



Объекты `NetConnection` и `NetStream` связаны с приложением `Rash Communication Server MX`, позволяющим двум и более SWF-файлам посредством IP-сети или Internet обмениваться потоковым аудио/видео и текстом в режиме реального времени. Например, `Flash Communication Server MX` можно использовать для реализации приложений голосовых и видеоконференций и обмена мгновенными сообщениями.

Приложение `Flash Communication Server` предназначено только для реализации связи в режиме реального времени. При использовании других типов данных и содержимого `Flash Communication Server MX` может поддерживать связь с такими серверами, как `Macromedia ColdFusion MX Server`, `Macromedia JRun` или `Microsoft .Net`.

В данной книге приложение `Flash Communication Server` детально не рассматривается. В этой главе дается лишь краткий обзор того, как данное приложение связано с `Flash Player` и с программой разработки `Rash MX`.

Приложения `Rash Communication Server` реализуются в виде двух модулей: серверного и клиентского. Клиентский модуль представляет собой SWF-файл, выполняющийся в приложении `Rash 6 Player`. Таким образом, программа разработки `Flash MX` используется для соз-

Дания приложений, использующих службы Flash Communication Server. Для расширения функциональных возможностей приложений можно также использовать серверные сценарии. Кроме того, компания Macromedia для отладки приложений предоставляет в распоряжение пользователей программы NetConnection Debugger и инспектор Communication App.

Для взаимодействия с Flash Communication Server приложение Flash 6 Player сначала устанавливает соединение посредством объекта NetConnection, а затем открывает один или несколько каналов связи на этом соединении. Каждый канал называется *сетевым потоком* и представлен объектом NetStream. Каждый поток представляет собой односторонний канал, с помощью которого можно либо опубликовать поток на сервере, либо воспроизвести поток, поступающий с сервера.

В табл. 20.16 перечислены клиентские методы и свойства объекта NetConnction, а также единственный обработчик события.

ТАБЛИЦА 20.16. МЕТОДЫ, СВОЙСТВА И ОБРАБОТЧИК СОБЫТИЯ ОБЪЕКТА NETCONNECTION

Имя	Метод/ свойство/ событие	ФОРМАТ	ОПИСАНИЕ
call	Метод	<code>myConnection.call(удаленный_метод , результирующий_Объект null [,пара-метр1...параметрN])</code>	Активизирует команду или метод на сервере
close	Метод	<code>myConnection.close()</code>	Закрывает связь с сервером
connect	Метод	<code>myConnection.connect(целевой_URL, [параметр1 , ... параметрN])</code>	Соединяет с приложением на Flash Communication Server
isConnected	Свойство	<code>myConnection.isConnected</code>	Булево значение, указывающее, соединен ли Flash 6 Player с сервером
url	Свойство	<code>myConnection.url</code>	Строка, представляющая целевой URL, указанный в методе NetConnection.connect()
onStatus	Событие	<code>myConnection.onStatus = function(infoObject) {}</code>	Активизируется при изменении статуса или при передаче ошибки объекту NetConnection. Объект <i>infoObject</i> имеет три свойства: <i>code</i> (код), <i>level</i> (уровень) и <i>description</i> (описание)

В табл. 20.17 перечислены методы и свойства объекта Netstream, а также его единственный обработчик события.

ТАБЛИЦА 20.17. МЕТОДЫ, СВОЙСТВА И ОБРАБОТЧИК СОБЫТИЯ ОБЪЕКТА NETSTREAM

Имя	Метод/ свойство/ событие	ФОРМАТ	ОПИСАНИЕ
attach Audio	Метод	myNetStream.attach Audio (источник)	Метод публикации. Связывает источник аудио с исходящим потоком
attch Video	Метод	myNetStream.attach Video (источник [null [, snapshotMillise- conds])	Метод публикации. Присоединяет видео или моментальный кадр определенного источника к исходящему потоку. Параметр <i>источнику</i> указывает объект Camera для начала захвата или <i>null</i> для завершения захвата. Параметр <i>snapshotMilliseconds</i> указывает, является ли видеопоток непрерывным (если параметр <i>snapshotMilliseconds</i> опущен), представляет ли он собой отдельный кадр (о или любое отрицательное значение) или последовательность отдельных кадров, используемых для создания покадровой съемки (любое положительное значение, указывающее задержку между захваченными кадрами)
close	Метод	myNetStream.close()	Останавливает публикацию или воспроизведение потока, освобождая его для другого применения
pause	Метод	myNetStream.pause([pauseResume])	Абонентский метод. Приостанавливает или возобновляет воспроизведение потока. Параметр <i>pauseResume</i> — это опциональная булева величина, указывающая, что нужно сделать: приостановить (<i>true</i>) или возобновить (<i>false</i>) поток. Если этот параметр опустить, то метод <i>pause</i> переключится от приостановки (при его первом вызове в потоке) к возобновлению воспроизведения
receive Audio	Метод	myNetStream.receive Audio (receive)	Абонентский метод. Параметр <i>receive</i> — это булева величина, указывающая, воспроизводится ли в потоке входящее аудио

Имя	Метод/ свойство/ СОБЫТИЕ	ФОРМАТ	ОПИСАНИЕ
play	Метод	<code>myNetStream.play (имя_публикации false [,start [,length [, сброс_списка_ воспроизведения]])</code>	Абонентский метод. Воспроизводит аудио, видео и текст с сервера Flash Communication Server. Параметр <i>имя_публикации</i> — это имя, установленное с помощью метода <code>myNetStream.publish(имя_публикации)</code> . Параметр <i>start</i> — это опциональное целое число, которое указывает начальное время (в секундах) и является ли поток "живым" либо записанным (табл. 20.18). Параметр <i>length</i> — это опциональное целое число, которое указывает длительность воспроизведения (в секундах) (табл. 20.19). Параметр <i>сброс_списка_воспроизведения</i> — это опциональная булева величина, указывающая, нужно ли сбрасывать (устранять) любые элементы текущего списка воспроизведения и нужно ли воспроизводить их немедленно (<i>true</i> — по умолчанию) или поставить в очередь и воспроизводить после элементов текущего списка воспроизведения (<i>false</i>)
publish	Метод	<code>myNetStream.publish (p (имя_публикации false [, как_публиковать])</code>	Метод публикации. Отправляет потоковое аудио, видео и текст от клиента на сервер Flash Communication Server (при этом поток может быть записан). Параметр <i>как_публиковать</i> — это опциональная строка. Если он установлен в значение <i>record</i> (записать), то Flash и опубликует поток, и запишет его в файл <code>.FLV</code> с тем же именем, что и <i>имя_публикации</i> . Файл хранится на сервере в каталоге приложений и будет записан поверх любого другого файла с тем же именем. Если параметр <i>как_публиковать</i> установлен в значение <i>append</i> (добавить), то Flash опубликует и запишет поток, но не поверх существующего файла, а в его конец. Если параметр <i>как_публиковать</i> опущен или установлен в значение <i>live</i> (вживую), то Flash опубликует поток без записи и в каталоге приложений на сервере удалит файл <code>.FLV</code> , имеющий то же имя, что и имя публикации

Имя	МЕТОД/ СВОЙСТВО/ СОБЫТИЕ	ФОРМАТ	ОПИСАНИЕ
receiveVideo	Метод	<code>myNetStream.receiveVideo(receiveFPS)</code>	Абонентский метод. Параметр receive — это булева величина, указывающая, воспроизводится ли в потоке входящее видео. Параметр <i>FPS</i> указывает частоту смены кадров видео. Указывается только один из двух параметров
seek	Метод	<code>myNetStream.seek(число_секунд)</code>	Абонентский метод; ищет определенную позицию в воспроизводящемся в данный момент записанном потоке
send	Метод	<code>myNetStream.send(имя_обработчика[, параметр1, ..., параметрN])</code>	Метод публикации. Позволяет широкополосную передачу данных всем клиентам-подписчикам. Параметр <i>имя_обработчика</i> — это имя метода получаемого объекта NetStream. Обработчик активизируется при получении сообщения. Параметры будут преобразованы в последовательную форму и могут относиться к данным любого типа
NetStream.setBufferTime	Метод	<code>myNetStream.setBufferTime(число_секунд)</code>	Параметр <i>число_секунд</i> — это целое число, указывающее, на сколько секунд следует заполнять данными буфер. При наличии публикуемого потока Flash начинает пропускать кадры, когда буфер заполнен. При наличии абонентского потока Flash заносит в буфер данные на указанное <i>число_секунд</i> перед началом воспроизведения
bufferLength	Свойство	<code>myNetStream.bufferLength</code>	Число секунд, на которое в настоящий момент буфер заполнен данными
bufferTime	Свойство	<code>myNetStream.bufferTime</code>	Время (в секундах), назначенное для занесения данных в буфер с помощью метода <code>myNetStream.setBufferTime</code>
currentFPS	Свойство	<code>myNetStream.currentFPS</code>	Текущая частота смены кадров потока, указываемая в количестве кадров в секунду
Time	СВОЙСТВО	<code>myNetStream.time</code>	Число секунд, прошедшее с начала воспроизведения или публикации
OnStatus	Событие	<code>myNetStream.onStatus = function(infoObject)</code> { }	Активизируется при изменении статуса или при передаче ошибки объекту NetStream

ТАБЛИЦА 20.18. ПАРАМЕТР START МЕТОДА **NETSTREAM.PLAY()**

ЗНАЧЕНИЕ	ОПИСАНИЕ
-2 (по умолчанию) или любое другое отрицательное значение, кроме -1	Flash сначала ищет "живой", а затем — записанный поток. Если ни один из них не найден, программа ожидает "живой" поток
-1	Flash ищет только "живой" поток. Если он не найден, Flash ожидает в течение числа секунд, определяемых параметром <i>length</i> , а затем начинает воспроизведение следующего элемента из списка воспроизведения. Если параметр <i>length</i> установлен в значение -1, Flash ожидает бесконечно. (Подробная информация о параметре <i>length</i> приведена в табл. 20.19.)
0 или положительное число	Flash ищет записанный поток и начинает его воспроизведение через число секунд, определяемое параметром <i>start</i> , после начала потока. Если записанный поток не найден, Flash немедленно переходит к следующему элементу в списке воспроизведения

ТАБЛИЦА 20.19. ПАРАМЕТР LENGTH МЕТОДА **NETSTREAM.PLAY()**

ЗНАЧЕНИЕ	ОПИСАНИЕ
-1 (по умолчанию) или любое другое отрицательное значение, кроме -1	Flash непрерывно воспроизводит "живой" поток либо воспроизводит записанный поток целиком
0	Flash воспроизводит один кадр через число секунд, определяемое параметром <i>start</i> , после начала записанного потока (параметр <i>start</i> должен быть нулем или положительным числом)
0 или положительное число	Flash воспроизводит первое число секунд, определяемое параметром <i>length</i> , "живого" или записанного потока. Если поток длится меньшее количество секунд, чем параметр <i>length</i> , то воспроизведение потока по его окончании остановится

Все взаимосвязи между Flash 6 Player и Flash Communication Server реализуются посредством протокола Real Time Messaging Protocol (RTMP), а не посредством протокола HTTP, использующегося для взаимосвязи между Web-серверами. Протокол RTMP обеспечивает постоянный канал двухсторонней связи.

Для публикации или воспроизведения аудио и видео в режиме реального времени с помощью Flash Communication Server необходимо выполнить пять основных этапов. Первые четыре всегда аналогичны, а пятый этап варьируется в зависимости от того, что выполняется: публикация или воспроизведение.

- Создайте объект `NetConnection` с помощью метода `new NetConnection()`.
- Установите связь с Flash Communication Server с помощью метода `myNetConnection.connect("rtmp://имя_сервера/имя_приложения/имя_экземпляра_приложения")`.
- Создайте сетевой поток для данного соединения с помощью метода `new NetStream(соединение)`.

- Присоедините определенный видео- и/или аудиопоток. При выполнении публикации можно захватить локальный поток из микрофона или из камеры посредством предложения `myNetStream.attachAudio(Microphone.get())` и `myNetStream.attachVideo(Camera.get())`. При воспроизведении потока, идущего с сервера, команды будут похожи, но вместо `Microphone.get()` и `Camera.get()` следует использовать *имя_публикации*, определенное при публикации потока на сервере.
- При выполнении публикации используйте метод `myNetStream.publish(имя_публикации)` для назначения имени данного потока и отправьте его на Flash Communication Server. При выполнении воспроизведения вызовите метод `myNetStream.play(имя_публикации)`.

Приведенный ниже код публикует аудио и видео в одном потоке и воспроизводит их в другом с помощью одного соединения (этот код не является особенно полезным, но с его помощью можно быстро продемонстрировать функциональные возможности).

```
1: myNetConnection = new NetConnection();
2: myNetConnection.connect("rtmp://mySvr.myDomain.com/App");
3: myNetStream_out = new NetStream(myNetConnection);
4: myNetStream_out.attachAudio(Microphone.get());
5: myNetStream_out.attachVideo(Camera.get());
6: myNetStream_out.publish("stock_quotes_082202");
7: myNetStream_in = new NetStream(myNetConnection);
8: myVideoObject.attachVideo(myNetStream_in);
9: myNetStream_in.play("stock_quotes_082202");
```

Обратите внимание на то, что сервер публикаций двумя отдельными предложениями (строки 4 и 5) объединяет видео и аудио в исходящий поток. С другой стороны, пользователь только присоединяет видео (строка 8). Звук автоматически воспроизводится посредством стандартного устройства пользователя. Однако аудиопоток можно присоединить к видеоклипу с помощью метода `MovieClip.attachAudio(myNetStream_in)`. Затем можно создать объект `Sound` для управления различными свойствами аудиопотока.

Совет

Примеры использования метода `MovieClip.attachAudio()` были приведены ранее в этой главе, в подразделе "Загрузка и присоединение звуков", а также в главе 17 "Демонстрация мощи видеоклипов".

КЛАСС LOCALCONNECTION



Класс `LocalConnection`, новый и недокументированный в первой версии Flash MX, позволяет взаимодействовать нескольким SWF-файлам, выполняющимся на одном компьютере. Один фильм может инициировать обработчик события другого фильма, а сами фильмы могут обмениваться произвольными данными. Функция локального соединения не зависит от браузера и, следовательно, работает независимо от того, поддерживает ли данный браузер JavaScript. Эта функция также работает во Flash 6 Player и в проекторах Flash.

Чтобы фильм мог посредством локального соединения взаимодействовать с другими объектами, сначала нужно создать экземпляр класса `LocalConnection`. Его можно будет использовать для взаимосвязи по любому количеству *связей* или виртуальных каналов. Каждое соединение имеет собственное имя и является взаимно-однозначным каналом связи с другим фильмом. Для передачи данных посредством того или иного соединения необходимо воспользоваться методом `send()` экземпляра локального соединения. Для получения данных необходимо воспользоваться методом `connect()`.

Функция-конструктор имеет очень простой формат:

```
myLocalConnection = new LocalConnection();
```

Класс `LocalConnection` имеет четыре метода: `send()`, `connect()`, `close()` и `domain()`. Первые два метода используются при работе с локальным соединением соответственно для передачи и получения SWF-файлов. В SWF-файле, передающем и получающем данные, используются и метод `send()`, и метод `connect()`.

Кроме того, если взаимодействующие фильмы не находятся в одном и том же домене, то для утверждения доменов, с которых будут поступать данные, в принимающем фильме необходимо создать обработчик `allowDomain()`. В отправляющем фильме можно определить обработчик события `onStatus`, в результате чего при активизации метода `send()` фильм получит уведомление об успешном или неудачном выполнении этой операции. Обработчик события `onStatus` будет активизирован, когда метод `send()` вернет значение `true`. (Возвращение значения `true` командой `send()` говорит о том, что формат команды был приемлемым, но не о том, что данные фактически были отправлены.) В качестве параметра обработчик события `onStatus` получает объект со строковым свойством `level` (уровень). Если это свойство установлено в значение `status`, это означает, что данные были отправлены успешно. Если значением свойства `level` является `error`, то данные не были отправлены успешно. Например, это произойдет в случае, если на данном соединении нет ни одного приемника.

Приведенный ниже код отображает в текстовом поле состояние соединения и имя, идентифицирующее SWF-файл.

```
myName = "tictactotallyautomatic";
xLC.onStatus = function(infoObject) {
    if (infoObject.level == "error") {
        _root.statusField.text = myName+" Connection xLC failed.";
    }
    else _root.statusField.text = myName+" Connection xLC succeeded.";
}
```

В табл. 20.20 приведены методы и обработчики события класса `LocalConnection`.

ТАБЛИЦА 20.20. МЕТОДЫ И ОБРАБОТЧИКИ СОБЫТИЙ КЛАССА `LOCALCONNECTION`

Имя	МЕТОД/ ОБРАБОТЧИК СОБЫТИЯ	ФОРМАТ	ОПИСАНИЕ
send	Метод	<code>myLocalConnection.send</code> (<i>имя_соединения</i> , <i>метод</i> [<i>p1</i> ... <i>pN</i>])	Отправляет данные по каналу <i>имя_соединения</i> . Возвращает значение <code>true</code> , если формат команды приемлемый
connect	Метод	<code>myLocalConnection.connect</code> (<i>имя_соединения</i>)	Начинает прием на канале <i>имя_соединения</i>
close	Метод	<code>myLocalConnection.close</code> (<i>имя_соединения</i>)	Удаляет соединение. Возвращает значение <code>true</code> , если операция прошла успешно, и значение <code>false</code> , если данное соединение не существует
domain	Метод	<code>myLocalConnection.domain()</code>	Возвращает строку с именем домена SWF-файла, содержащим экземпляр класса <code>LocalConnection</code> , например <code>macromedia.com</code>

Имя	Метод/ ОБРАБОТЧИК СОБЫТИЯ	ФОРМАТ	ОПИСАНИЕ
<code>allowDomain</code>	Обработчик события	<code>myLocalConnection.allowDomain = function (домен_отправителя) {}</code>	Будучи созданным в приемнике, должен возвращать значение <code>true</code> , если отправляющий домен <code>домен_отправителя</code> является приемлемым
<code>onStatus</code>	Обработчик события	<code>myLocalConnection.onStatus = function (infoObject) {}</code>	Посредством свойства <code>infoObject.level</code> уведомляет об успешном или неудачном выполнении команды <code>send()</code>

Метод `send()` имеет следующий формат:

```
myLocalConnection.send(имя_соединения, метод[p1...pN])
```

Параметры *имя_соединения* и *метод* представляют собой строки. Опциональные параметры, обозначенные как *p1...pN*, могут быть данными любого типа. Фильм, который в соединении *имя_соединения* является приемником, будет автоматически исполнять любой обработчик события, указанный в параметре *метод*. Параметры *p1–pN* будут передаваться этому методу.

Рассмотрим пример метода `send()`. Пусть экземпляр локального соединения имеет имя `xLC`, соединение называется `xmove`, обработчик события имеет имя `move()`, а переменная — `num`. (Этот пример взят из кода, рассматриваемого в конце этой главы, в разделе "Flash за работой".)

```
xLC.send("xmove", "move", num);
```

Со стороны приемника метод обработчика события поставлен в соответствие определенному соединению посредством метода `connect()`. Формат очень простой:

```
myLocalConnection.connect(имя_соединения);
```

Рассмотрим пример использования соединения с именем `xmove` и экземпляром локального соединения, имеющим имя `oLC`:

```
oLC.connect("xmove");
```

В принимающих фильмах для утверждения или отклонения соединений на основе домена отправителя используется метод `allowDomain()`. Интерпретирующая программа `ActionScript` в качестве параметра передает имя домена обработчику события `allowDomain()`. Если метод `allowDomain()` возвращает значение `true`, соединение разрешается. В приведенном ниже примере будут разрешены только соединения, установленные `allowedDomain.com`.

```
xLC.allowDomain = function(senderDomain){
    return (senderDomain == "allowedDomain.com");
}
```

Отправители и приемники могут указывать домен отправителя в явном виде, как в приведенных ниже примерах.

```
xLC.send("flashoop.com:xmove", "move", num);
oLC.connect("flashoop.com:xmove");
```


Если в имени соединения содержится символ двоеточия, то интерпретирующая программа **ActionScript** предполагает, что имя домена указано в явном виде. Если в имени соединения нет символа двоеточия, то по умолчанию перед именем соединения указывается домен фильма, содержащий экземпляр класса **LocalConnection**. Таким образом, два SWF-файла, выполняющиеся в одном и том же домене, могут использовать имя соединения без имени домена, как в приведенных ниже примерах.

```
xLC.send("xmove", "move", num);  
oLC.connect("xmove");
```

Для SWF-файлов, загруженных с локального жесткого диска, именем домена будет **localhost**, т.е. два приведенных ниже выражения будут эквивалентными, если отправляющий фильм был загружен с локального жесткого диска.

```
oLC.connect("xmove");  
oLC.connect("localhost:xmove")
```

Если при загрузке SWF-файлов с разных доменов вы хотите утвердить соединение без проверки имени домена, выполните два указанных ниже действия.

- Начните имя соединения с символа подчеркивания. Тогда сообщения будут отправляться без имени домена. Например:

```
oLC.connect("_myConnection");  
xLC.send("_myConnection", "move", num);
```

- Для приемника создайте обработчик события **allowDomain()**, который всегда будет возвращать значение **true**, как в следующем примере:

```
oLC.allowDomain = function(senderDomain){  
    return true;  
};
```

Соединения, имена которых начинаются с символа подчеркивания, будут разрешены даже без наличия метода **allowDomain()**, если фактическим доменом является **localhost**.

Локальные соединения не имеют встроенного мультивещания. Например, если должны взаимодействовать три фильма, то каждый фильм должен использовать три соединения: одно — для приема и по одному — для взаимодействия с каждым из остальных двух фильмов. Например, один фильм может принимать данные на соединении **incominga**, другой фильм — на соединении **incomingb**, а третий — на соединении **incomingc**. В этом случае соответствующий код в первом фильме будет выглядеть следующим образом:

```
1: myLocalConnection = new LocalConnection();  
2: myLocalConnection.connect("incominga");  
3: myLocalConnection.onReceive = function(message) {  
4:     // здесь указывается код обработчика события  
5: }  
6: function sendAll() {  
7:     myLocalConnection.send("incomingb", "onReceive", _root.input.text);  
8:     myLocalConnection.send("incomingc", "onReceive", _root.input.text);  
9: }
```

Код во втором фильме будет выглядеть аналогичным образом, за исключением того, что в строке 2 будет указано соединение **incomingb**, а в строке 7 — **incominga**. В третьем фильме в строке 2 будет указано соединение **incomingc**, а в строке 8 — **incominga**.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему я не могу получить доступ к звуку фильма, загруженного через Web?

Вот типичный сценарий проблемы. Пусть у вас есть фильм `soundsInHere`. Звуковой файл фильма `soundsInHere.fl` содержится в библиотеке фильма. Он имеет имя связи "ByeBye", а его код в основной временной шкале выглядит так:

```
mySound = new Sound();  
mySound.attachSound("ByeBye");  
mySound.start();
```

При тестировании фильма `soundsInHere` звук воспроизводится нормально. Но при загрузке файла `soundsInHere.swf` в другой фильм с помощью метода `loadMovie()` звук не воспроизводится.

Решение проблемы заключается в добавлении в файл `soundsInHere.fl` ключевого слова **this** следующим образом:

```
mySound = new Sound(this);  
mySound.attachSound("ByeBye");  
mySound.start();
```

Если снова опубликовать фильм `soundsInHere.swf`, звук будет воспроизводиться.

Во Flash 5, равно как и во Flash MX, такая ситуация является проблемной (комментарии о ней имеются на Web-узле www.macromedia.com/support/flash/ts/documents/attached_sound.htm).

Почему при загрузке в текстовое поле нового текста теряется форматирование?

Пусть создано текстовое поле `myText`, для которого задано свойство `text` и применен формат `myTextFormat`. Код работает нормально. Однако при изменении текста путем назначения нового значения свойству `text` текстовое поле возвращается к своей исходной конфигурации, заданной на панели инспектора свойств.

В добавление к методу `setTextFormat()` необходимо использовать метод `setNewTextFormat()`. Таким образом, задается формат нового входящего текста. Его необходимо задать всего один раз.

Приведенный ниже код иллюстрирует способ задания формата. При возвращении к исходному форматированию проблема возникает из-за того, что в вашем коде отсутствует строка, подобная строке 7.

```
1: _root.createTextField ("myText", 2, 100, 100, 200, 20);  
2: myTextFormat = new TextFormat();  
3: myTextFormat.Color = 0xFF0000;  
4: myTextFormat.font="verdana";  
5: myText.text = "Исходный текст";  
6: myText.setTextFormat(myTextFormat);  
7: myText.setNewTextFormat(myTextFormat);  
8: myText.text="Какой-то новый текст.";
```

FLASH ЗА РАБОТОЙ: ПОЛНАЯ АВТОМАТИЗАЦИЯ ИГРЫ В КРЕСТИКИ-НОЛИКИ С ПОМОЩЬЮ КЛАССА LOCALCONNECTION



Рассмотренная в *этой* главе функция локального соединения позволяет взаимодействовать двум отдельным SWF-файлам. Это открывает возможность написания двух отдельных, взаимодействующих друг с другом программ. Например, два программиста могут создать для какой-то игры разные программные “роботы”. Выигрывает тот робот, код ActionScript которого лучше соответствует стратегии данной игры.

Фильмы `tictactoe_lc.fl` и `player_lc.fl` показывают, как можно играть в крестики-нолики по локальному соединению с двумя SWF-файлами. Оба фильма созданы на основе файла `tictactoe.fl` (рассмотренного в разделе “Flash за работой” главы 13 “Использование операторов”), представляющего собой код автоматической игры в крестики-нолики в пределах одного SWF-файла, но с двумя виртуальными игроками, один из которых рисовал крестики, а другой — нолики. В файлах `tictactoe_lc.fl` и `player_lc.fl` функции двух виртуальных игроков разделены на два отдельных SWF-файла.

В файле `tictactoe_lc.swf` принимаются решения относительно движений “х”. В нем же реализуется отображение игрового поля и осуществляется проверка выигрышной комбинации при каждом ходе. В файле `player_lc.swf` принимаются решения относительно движений “о”, несмотря на то, что фактически они выполняются в файле `tictactoe_lc.swf` от имени файла `player_lc.swf`. В обоих фильмах метод `move()`, связанный с объектом локального соединения, инициируется при получении фильмом (по локальному соединению) сообщения, в котором говорится, каким был ход другого игрока.

Основная последовательность действий такая.

1. Игрок `tictactoe_lc.swf` делает ход “х” и отправляет сообщение игроку `player_lc.swf`.
2. Сообщение из файла `tictactoe_lc.swf` инициирует метод `move()`, соответствующий объекту локального соединения `oLC` в файле `player_lc.swf`. Этот метод локально в файле `player_lc.swf` записывает ход “х”, принимает решение относительно хода “о” и отправляет сообщение в файл `tictactoe_lc.swf`.
3. Сообщение из файла `player_lc.swf` инициирует метод `move()`, соответствующий объекту локального соединения `oLC` в файле `tictactoe_lc.swf`. Этот метод выполняет и записывает ход “о”, принимает решение относительно хода “х” и **отправляет** сообщение в файл `player_lc.swf`.

Шаги 2 и 3 повторяются до тех пор, пока файл `tictactoe_lc.swf` не определит, что кто-то из игроков победил или что все девять клеток игрового поля заполнены.

Каждый ход основывается на получении значения от 0 до 8 включительно. Девять чисел соответствуют девяти клеткам игрового поля в крестики-нолики. Для получения числового значения в обоих фильмах используются идентичные функции `getFreeSquare()`. Выбор клетки является случайным, но при этом из него исключаются уже заполненные клетки. Функции `getFreeSquare()` можно модифицировать так, что для принятия решения относительно своего хода два игрока будут использовать разные стратегии.

В листинге 20.1 приведен код файла `tictactoe_lc.fl`, относящийся к локальному соединению.

ЛИСТИНГ 20.1. ИЗ ФАЙЛА TICTACTOE_LC.FLA

```
1: xLC = new LocalConnection();
2: xLC.move = function (num) {
3:     j++;
4:     if (j < 9) {
5:         if (whoseMove) {
6:             num = getFreeSquare();
7:             xmoves(num);
8:             xLC.send("xmove", "move", num);
9:             whoseMove = !whoseMove;
10:            checkResult();
11:        }
12:    } else {
13:        omoves(num);
14:        whoseMove = !whoseMove;
15:        checkResult();
16:        xLC.move();
17:    }
18: }
19: }
20: xLC.connect("omove");
```

В строке 1 создается объект локального соединения xLC, который будет использоваться для отправки ходов "x" из файла `tictactoe_lc.swf` в файл `player_lc.swf`.

В строках 2–19 выполняется метод `move()`, соответствующий объекту локального соединения xLC. Метод `xLC.move()` выполняет и ходы "x", и ходы "o". (Сообщение о том, какой ход "o" будет сделан, поступает из файла `player_lc.swf`.)

В строке 2 файлом `player_lc.swf` для ходов "o" предоставляется параметр `num`. Для ходов "x" этот параметр не используется, поскольку число для каждого хода "x" определяется в строке 6.

Переменная `j`, приращение которой задается в строке 3, а тестирование происходит в строке 4, подсчитывает все ходы и обходит предложения в методе `move()` после того, как будут сделаны девять ходов.

Переменная `whoseMove`, проверка которой выполняется в строке 5, имеет значение `true`, когда наступает очередь хода "x", и значение `false`, когда наступает очередь хода "o".

В строке 7 выполняется ход "x".

В строке 8 отправляется сообщение в файл `player_lc.swf`. Файл `player_lc.swf` откликается на это сообщение. Когда из файла `player_lc.swf` поступает обратное сообщение, активизируется метод `xLC.move()` и инициируется выполнение хода "o".

Строка 9 изменяет значение переменной `whoseMove` с `true` на `false`, так что при следующей активизации метода `xLC.move()` будет выполняться код, реализующий ход "o" (строки 13–16).

В строке 10 производится обращение к функции `checkResult()`, которая проверяет, привел ли последний сделанный ход к образованию выигрышной комбинации. При наличии выигрышной комбинации функция `checkResult()` завершит игру.

В строке 13 выполняется ход "o".

Строка 14 изменяет значение переменной `whoseMove` с `false` на `true`, так что при следующей активизации метода `xLC.move()` будет выполняться код, реализующий ход "x" (строки 6–10).

В строке 15, как и в строке 10, выполняется обращение к функции `checkResult()`.

В строке 16 вызывается метод `xLC.move()`, инициирующий ход "x". Обратите внимание, что здесь параметр не указывается, поскольку для выполнения хода "x" он не требуется (см. примечание для строки 2). Вызов метода `xLC.move()` начинает весь процесс сначала.

В строке 20 метод `connect()` используется для подготовки файла `tictactoe_lc.swf` к приему локального соединения `omove`. Практика показывает, что лучше всего выполнять метод `connect()` только после создания метода (в данном случае `oLC.move()`), который будет обрабатывать данное соединение. В противном случае вы можете не получить сообщение для данного соединения, поскольку обработчик еще не существует. В данном случае такая ситуация фактически произойти не может, поскольку файл `player_lc.swf` отправляет сообщение только в ответ на сообщение, инициированное файлом `tictactoe_lc.swf`. Таким образом, невозможно, чтобы файл `player_lc.swf` отправил сообщение до того, как файл `tictactoe_lc.swf` будет полностью подготовлен.

В листинге 20.2 приведен код файла `player_lc fla`, относящийся к локальному соединению.

ЛИСТИНГ 20.2. КОД ФАЙЛА `PLAYER_LC.FLA`

```
1: oLC.move = function(param) {
2:     if (param == 10) { // игра окончена
3:         initArray();
4:         _root.statusField.text = "игра окончена";
5:         return;
6:     }
7:     squares[param] = null; // записать ход x
8:     num = getFreeSquare 0; // получить ход o
9:     squares[num] = null; // записать ход o
10:    oLC.send("omove", "move", num); // отправить ход o
11: }
12: oLC.connect("xmove"); // подготовить к получению "xmove"
```

В строках 1—11 представлен код обработчика ходов, указанный в строке 8 листинга 20.1.

Основными в этом листинге являются строки 10 и 12.

Строка 10 отправляет к локальному соединению `omove`, где файл `tictactoe_lc.swf` подготавливается к приему (см. строку 20 листинга 20.1).

Строка 12 подготавливает файл `player_lc.swf` к приему в локальном соединении `xmove`, которое использует файл `tictactoe_lc.swf` для отправления сообщений (см. строку 8 листинга 20.1).

Кроме того, в строке 2 показано, что при завершении игры файл `tictactoe_lc.swf` передает в виде параметра значение 10.

Также обратите внимание на строку 9, в которой показано, что когда клетка "занята", значение соответствующего элемента в массиве `squares` (клетки) устанавливается равным `null`. При поиске свободной клетки функция `getFreeSquare()` ищет элемент массива, не установленный в значение `null`.

УПАКОВКА ДАННЫХ и ФУНКЦИЙ с ПОМОЩЬЮ ПОЛЬЗОВАТЕЛЬСКИХ ОБЪЕКТОВ

В ЭТОЙ ГЛАВЕ...

Пользовательские объекты	515
Размещение локальных свойств в иерархии наследования	516
"Потеря" объекта <code>prototype</code> существующего класса при использовании оператора <code>new</code>	517
Подмена наследуемых свойств	520
Доступ к <code>надклассу</code> с помощью оператора <code>super</code>	523
Свойство <code>constructor</code>	524
Создание иерархии классов с помощью свойства <code>_proto_</code>	528
Разработка <code>надклассов</code> и <code>подклассов</code> : связи типа "являться" и "иметь"	532
Возможные проблемы	534

ПОЛЬЗОВАТЕЛЬСКИЕ ОБЪЕКТЫ

Пользовательские объекты создаются путем создания новой функции-конструктора, определяющей новый класс, и последующего использования ключевого слова `new` для создания новых объектов, принадлежащих этому классу.

Совет

Основные методики создания пользовательских объектов рассматривались в главе 15 "Объединение предложений в функции".

Пользовательские объекты позволяют расширить мощь и разносторонность объектов и классов применительно к решению любых задач. Если встроенные объекты реализуют общую концепцию объектно-ориентированного программирования, то благодаря пользовательским объектам можно выполнять очень широкий круг задач, используя при этом в полном объеме функциональные возможности ООП.

Применение пользовательских объектов порождает ряд вопросов, которые при использовании встроенных объектов, как правило, не возникают. Один из таких вопросов заключается в необходимости принятия решения об уровне, на котором должны располагаться локальные свойства.

РАЗМЕЩЕНИЕ ЛОКАЛЬНЫХ свойств в ИЕРАРХИИ НАСЛЕДОВАНИЯ

Функция-конструктор может наделить создаваемые с ее помощью экземпляры как локальными, так и коллективными свойствами. Например, пусть функция-конструктор `Bird` (птица) задает для каждого из своих экземпляров локальное свойство `makingNoise` (создание шума) и коллективный метод `move()` (движение). Чтобы указать, издает ли конкретный экземпляр в данный момент времени какой-то шум, он должен иметь собственную копию свойства `makingNoise`.

```
function Bird ( ) {  
    this.makingNoise = false; // локальное свойство "makingNoise"  
}  
Bird.prototype.move = function ( ) { }; // коллективный метод move()  
myBird = new Bird();  
myBird.hasOwnProperty("makingNoise"); // true
```

Оператор `new` (как говорилось в главе 15 "Объединение предложений в функции") отвечает за связывание объекта `myBird` с коллективным методом `move` О объекта-прототипа `Bird.prototype` и создание локального свойства `makingNoise` для экземпляра объекта `myBird`. При создании нового экземпляра свойство `makingNoise` всегда устанавливается в значение `false`.

Если вы не хотите, чтобы свойство `makingNoise` всегда устанавливалось в значение `false`, можно передать функции-конструктору аргумент, который будет использоваться в качестве значения свойства `makingNoise`. В приведенном ниже примере свойство `myBird.makingNoise` устанавливается в значение `true`.

```
function Bird (makingNoise) {  
    this.makingNoise = makingNoise; // локальное свойство  
    "makingNoise"  
}  
Bird.prototype.move = function ( ) { }; // коллективный метод move()  
myBird = new Bird(true);  
myBird.hasOwnProperty("makingNoise"); // true
```

При создании с помощью оператора `new` иерархий наследования нескольких классов коллективные свойства все равно остаются в объекте-прототипе надкласса. Локальные свойства создаются в объекте-прототипе подкласса. Иными словами, объект-прототип подкласса фактически становится экземпляром надкласса. Продолжая предыдущий пример, мы увидим следующее:

```
function Raptor ( ) { }  
Raptor.prototype = new Bird(false);  
Raptor.prototype.hasOwnProperty("makingNoise"); // true
```

Свойство `makingNoise` будет коллективным, поскольку оно содержится в объекте-прототипе. Однако в данном случае при наличии коллективного свойства `makingNoise` код

работать не будет. Необходимо иметь возможность определить, издает ли шум каждая отдельная птица. Следовательно, необходимо, чтобы каждый экземпляр на самом нижнем уровне иерархической структуры имел свою копию свойства `makingNoise`.

Этой цели можно добиться путем вставки в функцию-конструктор `Raptor` (хищник) оператора `super`.

Совет

Оператор `super` будет подробно описан далее в этой главе, в подразделе "Доступ к надклассу с помощью оператора `super`".

Кроме того, в функцию-конструктор `Raptor` необходимо добавить аргумент (как в приведенном ниже примере) так, чтобы для экземпляров самого нижнего уровня можно было задать значение свойства `makingNoise`. В **предположении**, что экземпляры функции `Raptor` представляют собой самый нижний уровень иерархической структуры, поставленная задача будет выполнена.

```
function Raptor (makingNoise) {
    super(makingNoise);
}
Raptor.prototype = new Bird(false);
Raptor.prototype.hasOwnProperty("makingNoise"); // бесполезно, но правильно
myBird = new Raptor(true);
myBird.hasOwnProperty("makingNoise"); // правильно и полезно
```

Если функция-конструктор `Raptor` имеет дочерний класс, то функции-конструкторы этого класса можно модифицировать точно так же, как и `Raptor`.

В приведенном ниже фильме `newbird fla`, имеющемся на прилагаемом к книге компакт-диске, создается иерархия `Bird-Raptor-Eagle-myBird` (птица—хищник—орел—объект `myBird`), в которой объект `myBird` имеет свойство `makingNoise`, установленное в значение `true`, даже несмотря на то, что все остальные свойства `makingNoise` установлены в значение `false`. Коллективные свойства объектов-прототипов `Raptor.prototype` и `Eagle.prototype`, вероятно, будут для вас бесполезны, поэтому их можно удалить.

```
function Bird (makingNoise) {
    this.makingNoise = makingNoise; // локальное свойство
    "makingNoise"
}
Bird.prototype.move = function () { }; // коллективный метод move()
function Raptor (makingNoise) {
    super(makingNoise);
}
Raptor.prototype = new Bird(false);
function Eagle (makingNoise) {
    super(makingNoise);
}
Eagle.prototype = new Raptor(false);
myBird = new Eagle(true);
```

"ПОТЕРЯ" ОБЪЕКТА PROTOTYPE СУЩЕСТВУЮЩЕГО КЛАССА ПРИ ИСПОЛЬЗОВАНИИ ОПЕРАТОРА NEW

Оператор `new` имеет одно ограничение, о котором нужно знать: если подкласс имеет существующий объект `prototype`, то при установлении связи этого подкласса с надклассом объект "теряется".

Что в данной ситуации означает слово "теряется"?

Объект `prototype` является свойством, состоящим из двух частей: имени и значения. Имя свойства является указателем на его значение, а значение само является объектом. Если убрать указатель, объект останется безымянным и, следовательно, на него нельзя будет ссылаться и использовать его (по крайней мере, с данным именем). На один и тот же объект могут ссылаться два указателя, один из которых можно убрать и использовать только другой. Если же удалить все указатели, то объект становится совершенно бесполезным. Тогда к делу должна приступить функция "уборки мусора", которая восстановит использующуюся память, стирая остатки ненужных объектов.

В процессе создания связи с **надклассом** имя свойства прототипа подкласса будет указывать на новый объект. Имя свойства остается таким же, как и раньше (прототип), но оно больше не будет указывать на тот же объект. Следовательно, класс "отрезается" от своего бывшего объекта `prototype`.

Предположим, к примеру, что в иерархии `Bird-Raptor-Eagle-BaldEagle` (описанной в главе 15 "Объединение предложений в функции") необходимо заменить класс верхнего уровня на новый и получить иерархию `Predator-Raptor-Eagle-BaldEagle`. Можно ли добиться выполнения поставленной задачи с помощью приведенного ниже кода?

```
function Predator () {  
}  
Raptor.prototype = new Predator();
```

Нет! Но почему? Ведь данная схема выглядит очень разумно. Вы установили связь класса `Raptor` с классом `Predator` и ничего не меняли в ветви `Raptor` иерархии наследования. Получим ли мы в результате иерархию `Predator-Raptor-Eagle-BaldEagle`. Если да, то член класса `BaldEagle`, например `baldyl`, больше не будет иметь доступ к свойствам, таким как `movement: "fly"` объекта `Bird.prototype`.

Практически все происходит почти наоборот. Вместо того чтобы убрать из цепочки наследования класс `Bird`, вырезается класс `Raptor`. (Этот процесс немного сложен, но разобраться в нем поможет рис. 21.1, на котором показан путь, который проходит интерпретирующая программа `ActionScript` при поиске свойств. Стрелки выполняют те же функции, что и свойство `_proto_` в `ActionScript`: они указывают на прототип функции-конструктора.)

Интерпретирующая программа создает новый объект-прототип `Raptor.prototype` и связывает его с объектом `Predator.prototype` посредством нового свойства `Raptor.prototype._proto_`.

Совет

Информация о свойстве `_proto_` содержится ниже, во врезке "Свойство `_proto_`".

Свойство `_proto_`

Каждый объект, созданный с помощью оператора `new`, получает свойство `_proto_` (обратите внимание на два символа подчеркивания по обеим сторонам слова). Обычно это свойство скрыто (например, в цикле `for-in` оно не отображается). Тем не менее для реализации наследования свойство `_proto_` является ключевым.

Одной из частей процесса создания нового объекта является задание интерпретирующей программой `ActionScript` свойства `_proto_` с целью указания на объект `prototype` функции-конструктора. Например, частью процесса создания объекта `baldyl` является установка интерпретирующей программой `ActionScript` свойства `baldyl._proto_` для указания на объект `prototype` функции-конструктора `BaldEagle`. Таким образом, две приведенные ниже строки ссылаются на один и тот же объект.

```
baldyl._proto_
```

BaldEagle.prototype

Интерпретирующая программа ActionScript выполняет поиск в объекте `prototype`, когда она не может найти свойство в самом объекте. Как же интерпретирующая программа переходит от объекта (`baldy1`) к объекту `prototype` конструктора (`BaldEagle.prototype`)? Нужный путь указывает свойство `_proto_`.

При желании впоследствии можно "повторно указать" свойство `_proto_` для ссылки на другой объект. Интерпретирующая программа будет выполнять поиск там, куда указывает свойство `_proto_`.

Кроме того, имеется свойство `BaldEagle.prototype._proto_`. Так же как свойство `baldy1._proto_` указывает интерпретирующей программе, где следует выполнять поиск в случае обнаружения свойства в объекте `baldy1`, свойство `BaldEagle.prototype._proto_` указывает путь к следующему объекту `prototype`, в котором следует выполнить поиск, если свойство не будет найдено в объекте `BaldEagle.prototype`.

Однако свойство `Eagle.prototype._proto_` продолжает указывать на *предыдущий* объект `Raptor.prototype`, который, в свою очередь, продолжает указывать на объект `Bird.prototype`. Цепочка наследования остается неизменной за исключением следующего.

- Изменения в объекте `Raptor.prototype` не влияют на класс `Eagle` (или `BaldEagle`), поскольку свойство `Eagle.prototype._proto_` не указывает на новый объект `Raptor.prototype`.
- Изменения в объекте `Bird.prototype` не влияют на класс `Raptor`, поскольку новое свойство `Raptor.prototype._proto_` указывает на объект `Predator.prototype`.

Иными словами, класс `Raptor` и новый объект `Raptor.prototype` полностью отсоединены от своей бывшей цепочки наследования. Бывший объект `Raptor.prototype`, теперь вызываемый только посредством свойства `Eagle.prototype._proto_`, остается в цепочке.

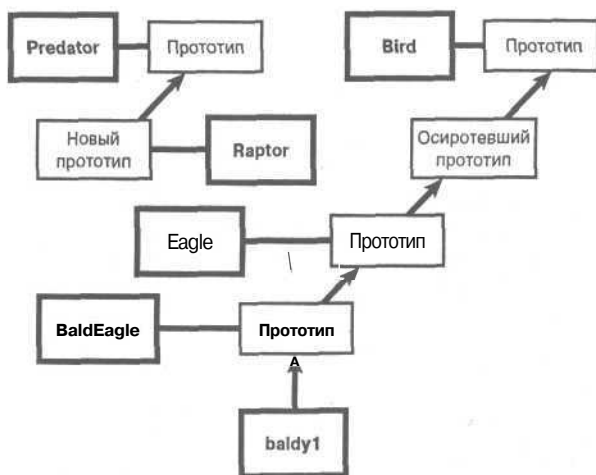


Рис. 21.1. Стрелки показывают путь, который проходит интерпретирующая программа ActionScript при поиске свойств. Как и свойство `_proto_`, они указывают на класс объекта `prototype`

А член класса `BaldEagle`, `baldyl` вообще не претерпевает никаких изменений.

Если интерпретирующая программа `ActionScript` ищет объекты `prototype` по мере прохождения по цепочке наследования, то предполагаемая цель будет достигнута при установлении связи класса `Raptor` с классом `Predator`. Функция-конструктор `Bird` и объект, который ранее был известен как `Raptor.prototype`, будут находиться вне цепочки наследования.

Однако интерпретирующая программа не ищет объекты `prototype`, а идет по пути, указанному свойством `_proto_`.

Если после присоединения класса `Raptor` к классу `Predator` вы хотите продвинуться дальше и с помощью оператора `new` получить искомую иерархию `Predator-Raptor-Eagle-BaldEagle`, необходимо повторно присоединить класс `Eagle` к классу `Raptor`, а класс `BaldEagle` — к классу `Eagle`. И наконец, необходимо повторно создать объект `baldyl`, который в противном случае будет продолжать использовать старую цепочку наследования, совершенно не подозревая о произошедших изменениях! Он будет продолжать ссылаться на всю цепочку предыдущего объекта `prototype`, ранее соответствовавшую классам `BaldEagle`, `Eagle`, `Raptor` и `Bird`.

Весь описанный процесс показан в файле `new1 fla`, код которого приводится ниже. В целях тестирования для исходного объекта `Predator` назначено свойство `cName`.

```
function Predator () {  
}  
Predator.prototype.cName = "Predator";  
Raptor.prototype = new Predator();  
Eagle.prototype = new Raptor();  
BaldEagle.prototype = new Eagle();  
baldyl = new BaldEagle();  
trace (baldyl.cName); // "Predator"
```

Если вы больше не собираетесь использовать класс `Bird`, его можно удалить.

Выполнение указанных шагов позволит создать искомую иерархию. Однако за это придется заплатить дорогую цену: необходимо разрушать и повторно создавать прототипы и экземпляры.

Путем "повторного" указания свойства `_proto_` вы сможете очень просто изъять класс `Bird` из цепочки наследования и заменить его на класс `Predator`, больше ничего не меняя.

Совет

Процедура повторного указания свойства `_proto_` рассматривается далее в этой главе, в подразделе "Создание иерархии классов с помощью свойства `_proto_`".

Создания цепочек наследования путем манипулирования непосредственно свойством `_proto_` компания `Macromedia` рекомендует избегать. Кроме того, свойство `_proto_` не определено в стандарте `ECMA-262`.

Однако в `ActionScript` свойство `_proto_` является документированным и не исключенным. Некоторые программисты предпочитают работать именно с ним, поскольку в некоторых случаях (таких как описанный выше) оно является более эффективным и менее деструктивным по сравнению с оператором `new`.

ПОДМЕНА НАСЛЕДУЕМЫХ СВОЙСТВ

Описанная в главе 15 "Объединение предложений в функции" цепочка наследования `Bird-Raptor-Eagle-BaldEagle` является всего лишь скелетной конструкцией, поскольку три нижних класса не играют никакой полезной роли. Единственное, что может делать подкласс, — это добавлять новое свойство. Например, поскольку хищники (`Raptor`) являются птицами, преследующими добычу (`prey`), то класс `Raptor` может добавлять свойство `prey` (как показано ниже, оно может быть представлено в виде массива элементов меню `Raptor`):

```
Raptor.prototype.prey = ["mice", "rabbits", "fish"]
```

Если подкласс определяет свойство, а родительский класс уже имеет свойство с тем же именем, то свойство подкласса "подменяет" родительское. При обращении к свойству объекта в пределах цепочки наследования интерпретирующая прогамма **ActionScript** начинает с низа цепочки и выполняет поиск снизу вверх, отыскивая свойство с этим именем. Как только свойство с этим именем будет найдено, поиск прекращается. Если подкласс имеет свойство с тем же именем, что и свойство надкласса, то интерпретирующая прогамма никогда не найдет свойство **надкласса**.

Пусть, например, создан класс **Bird** с методом `move()`, в который помещен код, необходимый для того, чтобы заставить птицу летать:

```
function Bird () { } // конструктор Bird
Bird.prototype.move = function () {
    // предложения, заставляющие птицу летать
    trace("fly"); // поместить это предложение для тестирования
};
```

Пусть теперь необходимо включить в программу птицу гагарку (**auk**). Эта птица, которую еще называют "арктическим пингвином", почти не имеет оперения и не умеет летать. Следовательно, стандартный метод `move()` класса **Bird** для нее не годится. Для включения в систему этой особой птицы можно создать новый класс со своим собственным методом `move()`, который подменит метод `move()` класса **Bird**:

```
1: function Auk () { } // конструктор Auk
2: Auk.prototype = new Bird(); // Auk наследует Bird
3: auk1 = new Auk(); // экземпляр Auk
4: auk1.move(); // отображает "fly" - но это не то, что нам нужно
5: Auk.prototype.move = function () {
6:     // предложения, заставляющие гагарку ходить вперевалку (waddle)
7:     trace ("waddle"); // для тестирования
8: }
9: auk1.move() ,- // отображается "waddle" - это уже лучше!
```

(Этот тестовый код содержится в файле `auk1 fla` на прилагаемом к книге компакт-диске.)

Обратите внимание на то, что наследование класса **Bird** (строка 2) необходимо задать до подмены свойства (строки 5, 6). Если сначала создать метод `Auk.prototype.move()`, а затем выполнить команду `Auk.prototype = new Bird()`, то оператор `new` не будет перенаправлять существующее свойство `Auk.prototype` на новый объект класса **Bird**, а оставит существующий метод в объекте, на который свойство `Auk.prototype` больше не указывает. Свойство `Auk.prototype` будет завершаться методом `move()` класса **Bird**.

Совет

Если вы совсем запутались при подмене методов получения/задания (устанавливаемых посредством предложения `addProperty()`), обратитесь к разделу "Возможные проблемы" в конце этой главы.

Подмена, произошедшая в подклассе (**Auk**), не влияет на надкласс (**Bird**) и на другие классы, наследующие его. В классе **Bird** метод `move()` остается неизменным. Если вы хотите изменить метод `move()` класса **Bird**, его необходимо *переписать*, а не просто подменить.

Совет

Процедура переписывания рассмотрена ниже, в подразделе "Переписывание наследуемых свойств".

ПОДМЕНА СВОЙСТВ В ОТДЕЛЬНЫХ ЭКЗЕМПЛЯРАХ

Свойство можно подменить в отдельном экземпляре, а не в целом классе. Например, если птица `auk1` в данный момент плавает, то для экземпляра `auk1` можно определить локальный метод `move ()` следующим образом:

```
auk1.move = function () {  
    // предложения, заставляющие гагарку грести (paddle)  
    trace ("paddle"); // для тестирования  
};  
auk1.move(); // отображается "paddle"
```

Однако при этом экземпляр `auk1` не будет иметь возможности доступа к коллективному методу `move ()` в течение всего времени, пока существует свойство экземпляра. Если удалить свойство экземпляра, то объект `auk1` снова будет иметь доступ к коллективному свойству:

```
delete auk1.move;  
auk1.move(); // снова отображается "waddle" (ходить вперевалку)
```

Подмена свойств в отдельном экземпляре может быть удобной, если почти все члены класса имеют определенное значение свойства и только один или два из них имеют другое значение; либо если вы хотите временно изменить поведение или значение.

ОТСУТСТВИЕ ПОДМЕНЫ В ГЛОБАЛЬНЫХ ОБЪЕКТАХ

Глобальные объекты можно использовать в том виде, в каком они есть, и к ним можно добавлять свойства, но изменять значения встроенных свойств глобальных объектов нельзя. Например, при попытке изменить значения свойства `Math.PI` вы обнаружите, что сделать это невозможно:

```
Math.PI = 12;  
trace(Math.PI); // все равно 3.14159265358979
```

Это справедливо и в том случае, когда свойство является методом. Например, в приведенном ниже коде в методе `Math.sqrt ()` осуществляется попытка умножения на 2, вместо того, чтобы извлекать квадратный корень. Если бы это можно было сделать, то метод `Math.sqrt (4)` дал бы значение 8. Но, как видно из кода, этого не произошло:

```
trace(Math.sqrt(4)); // "2" - квадратный корень из 4 равен 2  
Math.sqrt = function (arg) {return arg * 2;} // вместо этого умножаем на 2  
trace(Math.sqrt(4)); // все равно "2", т.е. извлекается квадратный корень
```

Может показаться, что с глобальными объектами вообще ничего нельзя сделать. Однако далее в этой главе, во врезке "Взлом объекта Math", будет описан способ выполнения такой задачи.

ПЕРЕПИСЫВАНИЕ НАСЛЕДУЕМЫХ СВОЙСТВ

Два приведенных ниже предложения демонстрируют разницу между подменой и переписыванием. Первое предложение изменяет метод `move ()` только для членов класса `Auk`. Второе предложение изменяет его для класса `Bird` и всех потомков — классов `Raptor`, `Eagle`, `BaldEagle` и `Auk`. (Предложения, реализующие функции, указываются внутри круглых скобок.)

```
Auk.prototype.move = function () { }; // подмена в подклассе  
Bird.prototype.move = function () { }; // переписывание в надклассе
```

ДОСТУП К НАДКЛАССУ С ПОМОЩЬЮ ОПЕРАТОРА SUPER



При работе с иерархиями наследования чаще всего приходится сталкиваться с двумя следующими ситуациями.

- Необходимо создать подкласс с методом, подменяющим метод надкласса. Однако при этом иногда нужно иметь доступ к поведению надкласса либо объединить метод подкласса с собственным добавленным поведением надкласса.
- Есть родительский класс с экземплярами и подкласс с экземплярами. (Например, класс `Bird` с экземплярами и класс `Auk` с экземплярами.) Родительский класс задает локальное свойство своим экземплярам. Вы хотите, чтобы локальной копией этого свойства обладали также экземпляры подкласса.

Обе эти ситуации можно разрешить с помощью оператора `super`, работающего только внутри функции. Он имеет специальные параметры для обращения к надклассу.

ДОСТУП К МЕТОДУ НАДКЛАССА С ПОМОЩЬЮ ОПЕРАТОРА SUPER

Пусть, к примеру, нужно, чтобы гагарка приобрела исключительные качества и научилась летать. Это означает, что вам нужен доступ к методу `move()` родительского класса `Bird`. Для доступа к методу родительского класса применяется оператор `super`, как продемонстрировано в строке 3 приведенного ниже кода.

```
1: Auk.prototype.move = function (supremeEffort) {  
2:   if (supremeEffort)  
3:     super.move();  
4:   else trace ("waddle"); // предложения, заставляющие гагарку  
                           //ходить вперевалку  
5: };  
6: auk1.move(true); // выполняется Bird.prototype.moveO,  
                   //отображается "fly"  
7: auk1.moveO; // отображается "waddle"
```

Любое значение аргумента `supremeEffort`, являющееся истинным, приводит к тому, что данная версия `Auk.prototype.move` будет выполнять метод `move()` надкласса. Это происходит в строке 6. Если аргумент `supremeEffort` не дает значения `true`, то `Auk.prototype.move` выполняет предложение `else`. Это происходит в строке 7.

Оператор `super` позволяет получить доступ к методу надкласса, даже несмотря на то, что в подклассе он подменен. Эту возможность можно использовать для добавления чего-либо к родительскому методу, если возникнет такая необходимость. Например, гагарку можно заставить ходить вперевалку (поведение подкласса), а потом летать (коллективное родительское свойство). Это реализуется посредством следующего кода:

```
Auk.prototype.move = function O {  
  trace ("waddle"); // предложения, заставляющие гагарку  
                    // ходить вперевалку  
  super.move();  
};  
auk1.move(); // отображается "waddle" и "fly"
```

Данная методика может пригодиться, если родительский метод имеет сложную функциональность, а в подкласс необходимо добавить что-то простое.

ПОЛУЧЕНИЕ ЛОКАЛЬНЫХ СВОЙСТВ НАДКЛАССА В ЭКЗЕМПЛЯРАХ ПОДКЛАССА

Приведенная ниже функция-конструктор `Bird` задает для каждого из своих экземпляров локальное свойство `singing` (пение) и коллективный метод `move()`. Каждому экземпляру необходима собственная копия свойства `singing` для того, чтобы указать, поет ли птица в данный момент. Как правило, свойство, определяющее состояние, не является коллективным, тогда как метод, в котором заключено поведение, очень легко предоставить в коллективное использование.

```
function Bird ( ) {  
    this.singing = false; // локальное свойство "singing"  
}  
Bird.prototype.move = function () { }; // коллективный метод move  
myBird = new Bird();  
myBird.hasOwnProperty("singing"); // true
```

Пусть теперь необходимо создать функцию-конструктор `Auk` так, чтобы каждый ее экземпляр имел собственное локальное свойство `singing`. Можно, конечно, продублировать код `this.singing = false` из конструктора `Bird` в конструктор `Auk`. Однако в этом нет необходимости. Вместо этого можно использовать оператор `super` для доступа к конструктору надкласса из конструктора подкласса, как показано ниже.

```
function Auk ( ) {  
    super();  
}  
Auk.prototype = new Bird();  
myAuk = new Auk();  
myAuk.hasOwnProperty("singing"); // true
```

Совет

Другие примеры использования оператора `super`, в том числе его использование в многоуровневой иерархии класса, приведены выше, в подразделе "Размещение локальных свойств в иерархии наследования".

Совет

При использовании оператора `super` получили ненужные свойства надкласса? Обратитесь к разделу "Возможные проблемы" в конце главы.

СВОЙСТВО CONSTRUCTOR

Каждый элемент данных и каждый объект при создании получают свойство `constructor` как часть, заимствованную у класса `Object`.

Совет

О свойстве `constructor` вы узнали в главе 19 "Использование встроенных базовых объектов".

Свойство `constructor` является ссылкой на класс функции-конструктора. Однако, в отличие от оператора `super`, свойство `constructor` доступно при наличии объекта, а не только в пределах методов и функций-конструкторов.

В `ActionScript` свойство `constructor` является недокументированным, но оно определено в стандарте `ECMA-262`.

В приведенном ниже примере показано, как можно использовать свойство `constructor` для исполнения функции-конструктора.

```
function MyFunc() {
    trace("running MyFunc");
}
myObj1 = new MyFunc();
myObj1.constructor(); // отображается "running MyFunc"
```

Прототип (`prototype`), будучи объектом, также имеет свойство `constructor`. При первом создании функции-конструктора (или любой другой функции) ее свойство `constructor` указывает на саму функцию, как можно увидеть из приведенного ниже примера.

```
MyFunc.prototype.constructor(); // отображается "running MyFunc"
```

Однако при выделении функции-конструктора в подкласс для него с помощью функции-конструктора надкласса создается новый объект `prototype`. Естественно, что свойство `constructor` прототипа подкласса теперь будет ссылаться на функцию, его создавшую. Например:

```
function Bird () {}
function Raptor () {}
Raptor.prototype = new Bird();
myRaptor = new Raptor();
trace(Raptor.prototype.constructor == Bird); // true
trace(myRaptor.constructor == Raptor); // true
```

СОЗДАНИЕ ОБЪЕКТОВ-ПОТОМКОВ с помощью СВОЙСТВА

CONSTRUCTOR

Свойство `constructor` позволяет членам класса создавать потомков, как в приведенном ниже примере.

```
function Auk() {
    this.localProp = "локальное свойство";
}
Auk.prototype.sharedProp = "коллективное свойство";
sammy = new Auk();
susie = new sammy.constructor(); // sammy создает потомка, susie
```

Последняя строка этого кода эквивалентна предложению
`susie = new Auk();`

Выражение `sammy.constructor()` позволяет исполнять конструктор при задании только имени объекта класса. Имя самого класса знать не нужно. Таким образом, можно написать функцию, объект которой создает другие объекты в своем классе, без необходимости жесткого кодирования имени класса в функции. Если затем передать имя объекта в функцию в виде аргумента, мы получим родовую функцию "создания потомков":

```
function makeSib (objName) {
    sib = new objName.constructor();
    return sib;
}
susie = makeSib(sammy); // эквивалент susie = new sammy.constructor();
```


ИСПОЛЬЗОВАНИЕ СВОЙСТВА CONSTRUCTOR для ПРЕОБРАЗОВАНИЯ ЭЛЕМЕНТАРНЫХ ТИПОВ ДАННЫХ В ОБЪЕКТЫ

Свойство `constructor` можно использовать также для преобразования элементарных типов данных в объекты. Рассмотрим функцию, выполняющую эту операцию:

```
function makeObj(primitive) {  
    obj = new primitive.constructor(primitive);  
    return obj;  
}
```

Может показаться **странным**, что этот подход работает, потому что примитивы не являются объектами и обычно не обладают свойствами, такими как `constructor`. Однако интерпретирующая программа `ActionScript` считает по-другому:

```
name1 = "Bob Smith";  
trace(typeof name1); // "string" - строковый примитив  
trace(typeof name1.constructor); // "function"
```

Интерпретирующая программа `ActionScript` знает, что свойство `constructor` является функцией. Если бы в строке не было функции-конструктора, то оператор `type` вернул бы значение `"undefined"`. Таким образом, "за кулисами" примитивов скрывается малая толика "объектности". Фактически примитивы, помимо свойства `constructor`, обладают и рядом других свойств:

```
trace(typeof name1.valueOf); // "function"  
trace(typeof name1.toString); // "function"  
trace(typeof name1.__proto__); // "object"  
trace(typeof name1.length); // "number"  
trace(name1.length); // 9
```

Вы можете узнать свойства `valueOf()` и `toString()`, как унаследованные у класса `Object`. Поэтому совершенно не удивительно, что примитивы можно преобразовывать в объекты.

Но зачем это нужно?

Предположим, что из текстового файла вы скопировали список имен и присваиваете их переменным, как в предыдущем примере с `"BobSmith"`. Пусть теперь необходимо добавить свойство, например номер телефона:

```
name1.phone = "555-555-5555";  
trace(name1.phone); // ничего не отображается  
trace(typeof name1.phone); // "undefined"
```

Этот код не работает. Свойства к примитивам добавить нельзя.

Никаких проблем. Воспользуйтесь методом `makeObj()` для преобразования строкового примитива в объект:

```
name1 = makeObj(name1);  
trace(typeof name1); // "object"
```

Вот и все. Строка преобразована в объект. Теперь попытаемся добавить номер телефона:

```
name1.phone = "555-555-5555";  
trace(typeof name1.phone); // "string"  
trace(name1); // "Bob Smith"  
trace(name1.phone); // "555-555-5555"
```

Код выполняется!

ИСПОЛЬЗОВАНИЕ ФУНКЦИИ-КОНСТРУКТОРА для "БЕЗОПАСНОГО"

ХРАНЕНИЯ

Функции являются объектами и могут иметь свойства. Функция-конструктор является подходящим местом хранения данных о **Классе**, которые не являются свойствами экземпляров класса. Например, в функции-конструкторе можно хранить имя класса:

```
function Auk ( ) {  
}  
auk1 = new Auk();  
Auk.cName = "Auk"; // хранит имя класса в свойстве
```

Теперь свойство `constructor` можно использовать для создания родовой функции с целью извлечения значения свойства:

```
function getObjectType (obj) {  
    return obj.constructor.cName;  
}  
trace(getObjectType (auk1)); // "Auk"
```

Зачем хранить имя функции в виде ее свойства? Ведь имя функции можно хранить в объекте `prototype`, и тогда для его извлечения не нужно использовать свойство `constructor`:

```
Auk.prototype.cName = "Auk";  
trace(auk1.cName); // "Auk"
```

С помощью прототипа доступ осуществляется гораздо проще и в большинстве ситуаций это — лучший выбор. Однако в использовании для хранения данных свойства функции есть одно преимущество. Вспомните, что прототип `Auk` "осиротел", если вставить `Auk` в цепочку наследования с помощью оператора `new`. Например:

```
// функция-конструктор Auk  
function Auk() {}  
// функция-конструктор Bird  
function Bird ( ) {}  
Auk.prototype.cName = "Auk";  
Auk.prototype = new Bird();  
auk2 = new Auk();  
trace(auk2.cName); // undefined  
trace(typeof auk2.cName); // "undefined"
```

Старое свойство `Auk.prototype.cName:Auk` является частью осиротевшего объекта-прототипа. Новый объект `Auk.prototype` является пустым — он не имеет свойства `cName`.

С другой стороны, если хранить имя класса в свойстве функции-конструктора, не нужно будет беспокоиться о том, что вы используете оператор `new` и, таким образом, "теряете" прототип класса. К имени класса все равно можно получить доступ в самом классе.

Функцию `getObjectType` можно использовать с глобальным объектом, как показано ниже.

```
Math.constructor.cName = "Math"; // хранить имя в свойстве cName  
trace(getObjectType (Math)); // "Math"
```

Подтверждение того, что объект `Math` относится к типу `"Math"`, вряд ли покажется особо полезным. Однако использование функции `getObjectType` является хорошей практикой программирования, а поддержка глобальных объектов приблизит вас к достижению этой цели.

СОЗДАНИЕ ИЕРАРХИИ КЛАССОВ с помощью СВОЙСТВА `_PROTO_`

Иерархии классов можно **создавать** путем задания указателя свойства `prototype._proto_` подкласса на свойство объекта `prototype` надкласса. Например, приведенная ниже строка делает Raptor подклассом Predator:

```
Raptor.prototype.__proto__ = Predator.prototype;
```

Фактически этот единственный шаг выполняет задачу, которая ранее в этой главе была описана как весьма сложная: изменение иерархии `Bird-Raptor-Eagle-BaldEagle` на `Predator-Raptor-Eagle-BaldEagle`.

Совет

Проблема изменения иерархии Bird-Raptor-Eagle-BaldEagle была рассмотрена ранее в этой главе, в подразделе "Потеря" объекта prototype существующего класса при использовании оператора new".

На рис. 21.2 продемонстрирован процесс задания наследования путем переуказания свойства `proto`.

Если класс `Bird` больше использоваться не будет, его можно удалить.

В схеме, основанной на свойстве `_proto_`, не используется оператор `new`. В ней не создается новый объект `Raptor.prototype`. В существующем объекте `Raptor.prototype` ничего не меняется, за исключением того, что свойство `_proto_` отсоединяется от класса `Bird` и присоединяется к классу `Predator`.

Кроме того, с помощью `_proto_` можно создать всю исходную цепочку. Предполагая, что функция-конструктор уже существует, код выглядел бы следующим образом:

```
Raptor.prototype.__proto__ = Bird.prototype;
```

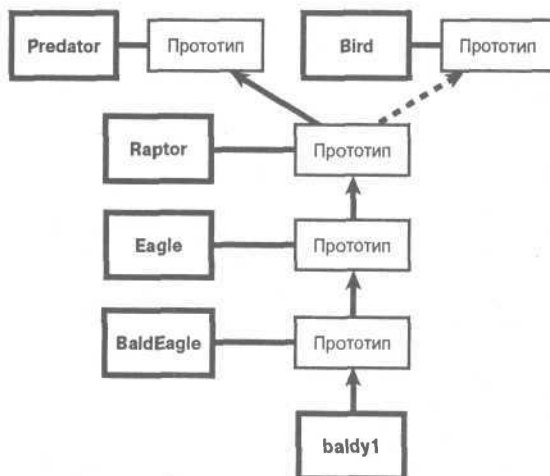


Рис. 21.2. Стрелки показывают путь, который проходит интерпретирующая программа ActionScript при поиске свойств. Пунктирной стрелкой вверх показана бывшая связь `proto`, а сплошной — новая связь `proto`

```
Eagle.prototype.__proto__ = Raptor.prototype;  
BaldEagle.prototype.__proto__ = Eagle.prototype;
```

В результате будет получена та же самая иерархия **Bird-Raptor-Eagle-BaldEagle**, которая была создана в главе 15 "Объединение предложений в функции" с помощью оператора `new`.

При создании наследования с помощью `_proto_` самая большая проблема заключается в переносе локальных свойств из надкласса в экземпляры класса более низкого уровня.

Совет

Процесс переноса локальных свойств из **надкласса** в экземпляры обсуждался ранее в этой главе, в подразделе "Размещение локальных свойств в иерархии наследования".

При выполнении только переуказания свойства `_proto_` вы вообще не получите никаких локальных свойств, поскольку не будут выполняться никакие функции-конструкторы. Кроме того, при этом не будет работать оператор `super`. Чтобы обойти эту проблему, выполните следующее.

- Для выполнения функций-конструкторов используйте свойство `constructor`.
- Создайте свойство `_constructor_` (обратите внимание на символы подчеркивания по обе стороны от слова) в прототипе подкласса, а затем используйте оператор `super`, как обычно.

При любом из этих подходов для облегчения собственной жизни можно написать функцию, в которой содержался бы код задания наследования. В языке Java в этом контексте используется ключевое слово `extends`. В **ActionScript** можно использовать любое имя функции, но все же слово `extends` будет понятно более широкому кругу пользователей. (Слово `extends` зарезервировано в **ActionScript** для использования в будущем. Поэтому, чтобы избежать возможных конфликтов, используйте имя `myExtends`.) Из-за того что оба подхода основываются на использовании свойства `_proto_`, сущность функции будет такой:

```
Function.prototype.extends = function (superclass)  
{  
  this.prototype.__proto__ = superclass.prototype;  
}
```

ИСПОЛЬЗОВАНИЕ СВОЙСТВА `CONSTRUCTOR` для ИСПОЛНЕНИЯ ФУНКЦИЙ-КОНСТРУКТОРОВ

При использовании свойства `constructor` для исполнения функций-конструкторов можно воспользоваться стандартной и понятной функцией `extends()`. После ее выполнения вы будете знать, что `subclass.prototype._proto_ = superclass.prototype`, поскольку это именно то, что делает функция `extends()`.

Дальнейшей целью является исполнение конструктора надкласса при заданном подклассе. Ниже описано, как это можно сделать.

- Поскольку оператор `new` для создания наследования надкласса не использовался, то свойство надкласса `prototype.constructor` все равно должно указывать на саму функцию-конструктор надкласса.
- В связи с тем, что свойство надкласса `prototype._proto_` указывает на прототип надкласса, то выражение вида `subclass.prototype._proto_.constructor` указывает на `prototype.constructor` надкласса. Как только что было отмечено, это означает саму функцию-конструктор надкласса.
- Таким образом, для выполнения конструктора надкласса можно использовать выражение следующего вида: `subclass.prototype._proto_.constructor()`.

В этом состоит основной подход. Его можно использовать для создания функции `mySuper()`, которая подменяет функциональность метода `super()`. Сущность функции будет такой:

```
mySuper() {
    subclass.prototype.__proto__.constructor();
}
```

В этой функции *subclass* — это псевдокод, обозначающий подкласс. Поскольку вызов функции `mySuper()` будет осуществляться из конструктора подкласса (в данном случае `Raptor`), то для подкласса можно использовать `arguments.caller`. Теперь функция выглядит следующим образом:

```
mySuper() {
    arguments.caller.prototype.__proto__.constructor();
}
```

Кроме того, необходимо иметь возможность передавать аргументы в конструктор надкласса так, чтобы экземпляры самого нижнего уровня могли задавать значения своих локальных свойств. Функция `mySuper()` не определяет никаких аргументов. Однако в неявном виде все аргументы, передаваемые в любую функцию, содержатся в объекте `arguments`. Воспользуемся этим фактом, а также методом `Function.apply()` для получения имеющего такой вид эквивалента `super()`:

```
mySuper() { // на основе функции superCon() от Роберта Пеннера
    arguments.caller.prototype.__proto__.constructor.apply
    (this, arguments);
}
```

Совет

Объект `arguments` и метод `Function.apply()` были рассмотрены в главе 15 "Объединение предложений в функции".

И, наконец, последний вопрос заключается в том, где следует определить эту функцию. Применимым может быть целый ряд вариантов. Например, в фильме `newrap1 fla`, код которого приведен ниже, функция помещена в объект `Raptor.prototype`.

```
1: Function.prototype.extends = function (superclass)
2: {
3:   this.prototype.__proto__ = superclass.prototype;
4: }
5: function Bird (singing) {
6:   this.singing = singing; // локальное свойство "singing"
7: }
8: Bird.prototype.move = function () { }; // коллективный метод move()
9: Raptor.prototype.mySuper = function() {
10:   arguments.caller.prototype.__proto__.constructor.apply (this,
    arguments);
11: };
12: function Raptor () {
13:   this.mySuper(arguments);
14: }
15: Raptor.prototype.extends(Bird);
16: newRap = new Raptor(true);
17: trace(newRap.hasOwnProperty("singing")); // true
18: trace(typeof newRap.move); // function
19: trace(Raptor.prototype.hasOwnProperty("singing")); // false
20: trace(newRap.singing); // true
```

Весь код перед строкой 16 является заданием иерархии наследования. Код, расположенный после строки 16, используется для тестирования.

В строке 16 выполняется конструктор `Raptor`, начиная процесс создания нового объекта `Raptor`.

Внутри конструктора `Raptor` ключевое слово `this` относится к родовому объекту, который находится в процессе отнесения к членам класса `Raptor`. Этот объект имеет доступ к `Raptor.prototype` и, таким образом, может исполнить функцию `mySuper()`. Это реализуется в строке 13 путем передачи аргумента, в свою очередь переданного в конструктор `raptor` посредством предложения строки 16.

Внутри функции `mySuper()` ключевое слово `this` все равно относится к родовому объекту, а аргумент остается таким же, каким и был раньше. В строке 10 и `this`, и `arguments` используются с методом `Function.apply()` для вызова конструктора надкласса.

СОЗДАНИЕ СВОЙСТВА `_constructor_` В ПРОТОТИПЕ ПОДКЛАССА

Создание свойства `_constructor_` в прототипе подкласса основано на следующей недокументированной особенности. Если после выполнения основанного на свойстве `_proto_` метода `extends()` объект `subclass.prototype._constructor_` указывает на надкласс, то стандартная встроенная функция `super()` работает корректно.

В фильме `newrap2.fl` свойство `subclass.prototype._constructor_` указывает на надкласс в пределах метода `extends()`. Затем свойство `subclass.prototype._constructor_` скрывается (т.е. в циклах `for-in` оно показано не будет) с помощью недокументированной функции `ASSetPropFlags()`, чтобы попытаться продублировать способ, которым Flash выполняет операции в ответ на обычное предложение `new`. Вне метода `extends()` оставшаяся часть кода является полностью стандартной.

Совет

Сравните приведенный ниже код с кодом, рассмотренным ранее в этой главе, в подразделе "Размещение локальных свойств в иерархии наследования".

Код файла `newrap2.fl` выглядит следующим образом:

```
1: Function.prototype.extends = function (superclass)
2: {
3:     this.prototype.__proto__ = superclass.prototype;
4:     this.prototype.__constructor__ = superclass;
5:     ASSetPropFlags(this.prototype, ["__constructor__"], 1);
6: }
7: function Bird ( ) {
8:     this.singing = false; // локальное свойство "singing"
9: }
10: Bird.prototype.move = function ( ) { }; // коллективный метод
move()
11: function Raptor ( ) {
12:     super();
13: }
14: Raptor.extends(Bird);
15: newRap = new Raptor();
16: trace(newRap.hasOwnProperty("singing")); // true
17: trace(typeof newRap.move); // function
18: trace(Raptor.prototype.hasOwnProperty("singing")); // false
```

РАЗРАБОТКА НАДКЛАССОВ и ПОДКЛАССОВ: связи ТИПА "являться" и "ИМЕТЬ"

Большинство методик объектно-ориентированного программирования заключается в разработке иерархий **надклассов** и подклассов, позволяющих оптимально промоделировать область действия задачи. Создание таких иерархий далеко от точной науки, и два программиста, решающих одну и ту же проблему, никогда не придут к созданию совершенно одинаковых решений. Тем не менее есть одно общее правило, которое редко нарушается (если нарушается вообще): подкласс должен иметь связь типа "являться" по отношению к своему **надклассу**.

Связь ТИПА "являться"

Каждый и любой член подкласса должен принадлежать и его надклассу. Например, белоголовый орел (BaldEagle) "является" орлом (Eagle). Орел "является" хищником (Raptor). Хищник "является" птицей (Bird). Не может быть, чтобы белоголовый орел не был орлом.

Если вы затрудняетесь создать логическую иерархию, начните с того, что подвергните ее простому тестированию. Если между каждым подклассом и **надклассом** не существует связи типа "являться", то, вероятно, структуру иерархии придется доработать.

СВЯЗЬ ТИПА "ИМЕТЬ"

Если вы обнаружите, что между двумя объектами программы существует связь типа "иметь", то один объект может быть свойством другого. Например, птица имеет оперение. Такая взаимосвязь называется "композицией", например "Птица состоит из оперения (и нескольких других **элементов**)". Объекты, имеющие такую связь, не всегда являются хорошими **надклассами** и подклассами друг для друга.

Еще одним типом связи "иметь" является клиентская связь, когда один объект использует другой объект. Например, велосипедист использует велосипед. Опять же, "велосипед" можно сделать свойством "велосипедиста", а не подклассом. Клиентская связь является очень широкой, и к ней могут относиться взаимоотношения типа "ученик—учитель" и "работодатель—работник".

ПРОЯСНЕНИЕ сложных СЛУЧАЕВ

Отличия между связями типа "иметь" и "являться" не всегда могут быть такими простыми, как это кажется на первый взгляд. Например, можно сказать: "Внутри каждого композитора "имеется" математик". Однако это не слишком отличается от утверждения типа "Каждый композитор также является математиком".

На помощь при прояснении сложных случаев может прийти вопрос "Можно ли изменить такую взаимосвязь?" Если ее можно изменить, то, вероятно, это связь типа "иметь"; если нет, то, скорее всего, это связь типа "являться".

Например, если вы считаете, что каждый композитор *должен* также быть математиком, то вы имеете дело со связью типа "являться", даже несмотря на то, что в ее описании могут встречаться слова "иметь" или "иметься".

ДЕТАЛИЗАЦИЯ ИЕРАРХИЙ КЛАССОВ: АБСТРАКЦИЯ и КОНКРЕТИЗАЦИЯ

В главе 15 "Объединение предложений в функции" упоминалось о том, что у программистов есть стимул для создания сильно обобщенных классов.

Однако не во всех случаях сразу же становится очевидно, какие свойства должны иметь такие классы. Часто бывает легче сначала создать ряд определенных классов, выяснить, какие свойства у них общие, а затем "абстрагировать" родительский класс с этими свойствами. Такой тип абстрагирования, при котором общие свойства "выносятся за скобки", называется "вынесением". Вынесение устраняет необходимость выполнения дублирования в подклассах и в целом является полезной возможностью.

В результате вынесения мы получим подклассы с меньшим количеством свойств. В некоторых случаях для охвата непредвиденных отличий в области действия задачи приходится создавать новые подклассы, часто с добавлением свойств. Например, можно создать сильно обобщенный метод `move()` и назначить его для класса `MovieClip`, предоставляя стандартный способ перемещения видеоклипов. Однако при разработке игры в настольный теннис вы можете обнаружить, что видеоклипу теннисного мячика нужен специализированный метод `move()` и другие специальные свойства. Для выполнения этой задачи можно создать подкласс `pingPongBall`, обладающий собственными свойствами и методом `move()`. Такой тип детализации иерархии называется "конкретизацией". Он усложняет иерархию за счет создания новых более сложных классов, и поэтому для его применения должна существовать веская причина.

Взлом объекта Math

В целом изменять значения встроенных свойств глобальных объектов нельзя, так как это может причинять неудобства. Предположим, есть программа, в которой используется много математических функций, в том числе использующих целые числа, и вы хотите изменить "точность" представления данных таким образом, чтобы округление производилось не до ближайшего целого, а до десятых долей целого. С этой целью к объекту `Math` можно добавить несколько новых методов, с помощью которых будет выполняться округление до ближайшей десятой доли числа. Затем в программе необходимо изменить все ссылки на объект `Math`. Насколько легче была бы жизнь, если бы можно было изменить только функции объекта `Math`!

Способ достижения этой цели существует. Он основывается на том факте, что объект `Math` является свойством объекта верхнего уровня (`Object`). Способ заключается во "взломе" имени свойства `Math` и задании для него указателя на объект, созданный с помощью оператора `new`.

Ниже приведем пример кода.

```
1: x = 10.21
2: trace("1 : "+Math.floor(x)); // "10" - вызывает оригинал
3: myMath = new Object();
4: myMath.__proto__ = Math;
5: myMath.oldFloor = Math.floor;
6: myMath.floor = function (arg) {
7:     arg *= 10;
8:     var result = myMath.oldFloor (arg);
9:     return result/10;
10: }
11: Object.prototype.Math = myMath;
12: trace("2 : "+Math.floor(x)); // "10.2" - вызывает
    // пользовательское свойство
```

Сначала создается новый объект (строка 3). Затем используется свойство `__proto__`, чтобы новый объект выполнял поиск в объекте `Math` в том случае, когда он не может найти вызываемое свойство локально (строка 4). Этот шаг очень важен, поскольку "реальный" объект `Math` будет "скрыт" за вашим объектом, и доступ к нему можно будет получить только посредством этой связи `proto`. Если вы хотите добавить новое свойство и при этом все равно иметь возмож-

ность доступа к старому, то в новом объекте создайте свойство, указывающее на старое (строка 5). Теперь создайте новую математическую функцию (строки 6-10).

В строке 11 происходит "преступление". Здесь вы "крадете" имя "Math" в `Object.prototype` и создаете его указатель на новый объект.

Вот и все. Строка 12 только подтверждает, что эта процедура работает: `Math.floor()` теперь выполняет округление до одного десятичного разряда, в отличие от строки 2, где округление проводилось до целого числа.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему я не могу подменить методы получения/задания ?

При использовании метода `addProperty()` обычным способом функции получения/задания подменять нельзя. Например, предположим, что есть такое предложение:

```
function MyClass (> {  
    this.addProperty("myProp",  
        MyClass.prototype.getMyProp,  
        MyClass.prototype.setMyProp);  
}
```

Методы `MyClass.prototype.getMyProp()` и `MyClass.prototype.setMyProp()` будут выполняться, когда свойство `myProp` соответственно считывается или записывается.

Теперь создадим подкласс с методами, имеющими те же самые имена, что и в прототипе:

```
function MySubClass () {}  
MySubClass.prototype = new MyClass();  
MySubClass.prototype.getMyProp = function () {  
    // предложения  
};  
MySubClass.prototype.setMyProp = function (arg) {  
    // предложения  
};
```

И наконец, мы создаем экземпляр подкласса и получаем или задаем свойство `myProp`:

```
myObj = new MySubClass();  
trace(myObj.myProp); // получение  
myObj.myProp = "Gary Good"; // задание
```

Вы надеетесь, что будут выполняться методы *подкласса*. Но этого не произойдет. Выполняться будут методы *надкласса*. Это не дефект, а просто реалии жизни.

Почему при использовании оператора `super()` я получаю ненужные свойства из надкласса?

В целом предложение `super()` следует выполнять в первой строке конструктора подкласса. При этом последовательные предложения в конструкторе подкласса могут подменять свойства, созданные надклассом. Если предложение `super()` будет выполнено в конструкторе позднее, то возникает риск того, что предложение в конструкторе надкласса может подменить предложение в конструкторе подкласса, а это не совсем то, как обычно работают классы.

Предположим, к примеру, что создан универсальный класс `Window`, который Предоставляет функциональность, необходимую для создания системы управления окнами во Flash.

Этот класс задает для всех своих экземпляров два локальных свойства (помимо многих других): `minMax` — булеву величину, указывающую, развернуто или свернуто окно в данный момент, и `bg` — шестиразрядное шестнадцатеричное число, определяющее фоновый цвет.

```
function Window () = {  
  this.minMax = true; // развернуто  
  this.bg = 0xFFFFFF; // белый фон по умолчанию  
}
```

Теперь предположим, что для определенного узла необходимо создать окна с растровым фоном. Для этого необходимо создать новый класс, `BitWindow`; сделать его подклассом `Window`, подменить свойство `bg` родительского класса и выполнить предложение `super()` так, чтобы экземпляры класса `Bitwindow` получили остальные локальные свойства (в данном случае — `minMax`) из головного класса.

Однако вы ошиблись и выполнили предложение `super()` *после* подмены свойства `bg`. В результате экземпляры класса `Bitwindow` получают свойство `bg` из головного класса, а не растровое свойство `bg`.

```
function Bitwindow () {  
  this.bg = "bg.jpg"; // растровый фон по умолчанию  
  super(); // Ой!  
}  
BitWindow.prototype = new Window();  
win1 = new BitWindow();  
trace(win1.bg); // 16777215 (то же, что и 0xFFFFFF) - не то, что нужно!
```

Если поместить предложение `super()` в первую строку конструктора `Bitwindow`, то экземпляры класса `Bitwindow` получат растровое свойство `bg`.

КОМПОНЕНТЫ

В этой главе...

Использование встроенных компонентов	537
Создание новых компонентов	545
Возможные проблемы	558

ИСПОЛЬЗОВАНИЕ ВСТРОЕННЫХ КОМПОНЕНТОВ

Во Flash 5 компания Macromedia представила функцию **Smart Clips** (Интеллектуальные клипы), разработанную как способ создания анимации с параметрами. **Smart Clips** — это видеоклипы с назначенными параметрами. Путем добавления действий и сценариев можно создавать переключатели, меню и другие элементы, реагирующие на щелчки мышью. Благодаря появлению компонентов во **Rash MX** эта возможность перенесена на качественно новый уровень.

Несмотря на преимущества возможностей создания анимации, получения коллективного доступа к данным и повторного использования кода, главный недостаток функции **Smart Clips** (который быстро обнаружили профессионалы Flash) заключался в том, что она не соответствовала объектно-ориентированной конструкции **ActionScript**. А поскольку **ActionScript** провозглашался как самое лучшее изобретение Flash, то эта проблема для очень многих разработчиков была весьма актуальной.

Разработчики могут не только создавать специализированный пользовательский интерфейс для компонентов, но и использовать функцию динамического предварительного просмотра. Это позволяет увидеть, как именно будут выглядеть используемые компоненты. Процедура создания специализированного пользовательского интерфейса и функция динамического предварительного просмотра будут рассмотрены далее в этой главе, при описании процедуры создания собственных компонентов.

На панели **Components** (Компоненты) показаны встроенные компоненты Flash. Можно сразу же заметить, что они разработаны для использования в формах.

Существуют два способа добавления Flash-компонентов в файлы. Самый очевидный из них заключается в перетаскивании экземпляра компонента с рабочего поля в файл. Второй способ заключается в добавлении компонентов программным образом с помощью метода `AttachMovie`. Для этого во временной шкале выделите кадр, в котором предполагается разместить компонент, откройте панель Actions (Действия) в режиме Expert и создайте функцию, предназначенную для создания экземпляра компонента. В приведенном ниже коде показано, как добавить экземпляр компонента `CheckBox` (Флаговое поле) с именем `myCheckBox` и со значением `unchecked` (флажок не установлен).

```
_root.attachMovie("FCheckBoxSymbol", "checkbox01", Z);
_root.checkbox1.setValue(false);
_root.checkbox1.setLabel("myCheckbox");
```

Для компонентов можно создать функции изменения обработчика событий, которые будут активизироваться при выборе пользователем элемента меню. Лучше всего создать функцию обработчика, определяющую действия, а затем использовать ее в качестве параметра компонента. Обработчик изменения всегда принимает, по крайней мере, один параметр, представляющий собой измененный экземпляр компонента.

В приведенном ниже примере рассматривается функция обработчика `onChange` для двух флаговых полей, каждое из которых является отдельным компонентом. В функции используются предложения `if-else`, позволяющие увидеть, какое из флаговых полей является выделенным, и затем на основе этого открыть соответствующее окно списка:

```
function onChange(component)
{
    if (component._name=="checkbox01") {
        Box1_mc.setEnabled(component.getValue());
    } else if (component._name=="checkbox02") {
        Box2_mc.setEnabled(component.getValue());
    }
}
```

Каждый компонент имеет набор параметров, определяющих его внешний вид и поведение. На рис. 22.1 изображен компонент `CheckBox` с доступными для него параметрами.

КОМПОНЕНТ `ComboBox`

Компонент `ComboBox` (Комбинированный управляющий элемент) предоставляет несложный способ создания простого выпадающего меню, позволяющего пользователю выбрать нужный элемент из списка.

В приведенном ниже примере рассматривается процедура формирования комбинированного окна, предоставляющего пользователю список чисел, из которых он может выбрать нужное. К примеру, при создании формы для заказа какого-то товара такой список будет очень удобен для указания количества единиц выбранного товара. Начните с перетаскивания экземпляра компонента `ComboBox` в рабочее поле.

1. Выделите комбинированное окно в рабочем поле. При этом на панели инспектора свойств будут отображены параметры компонента. Присвойте имя экземпляру компонента.
2. На панели инспектора свойств выделите параметр `Labels` (Надписи) и щелкните на значке с изображением лупы, в результате чего откроется всплывающее окно `Values` (Значения).
3. Параметр `Labels` содержит список значений, которые пользователь может выбрать из раскрывающегося меню.
4. В открывшемся окне `Values` щелкните на знаке "плюс", чтобы вывести новое значение.

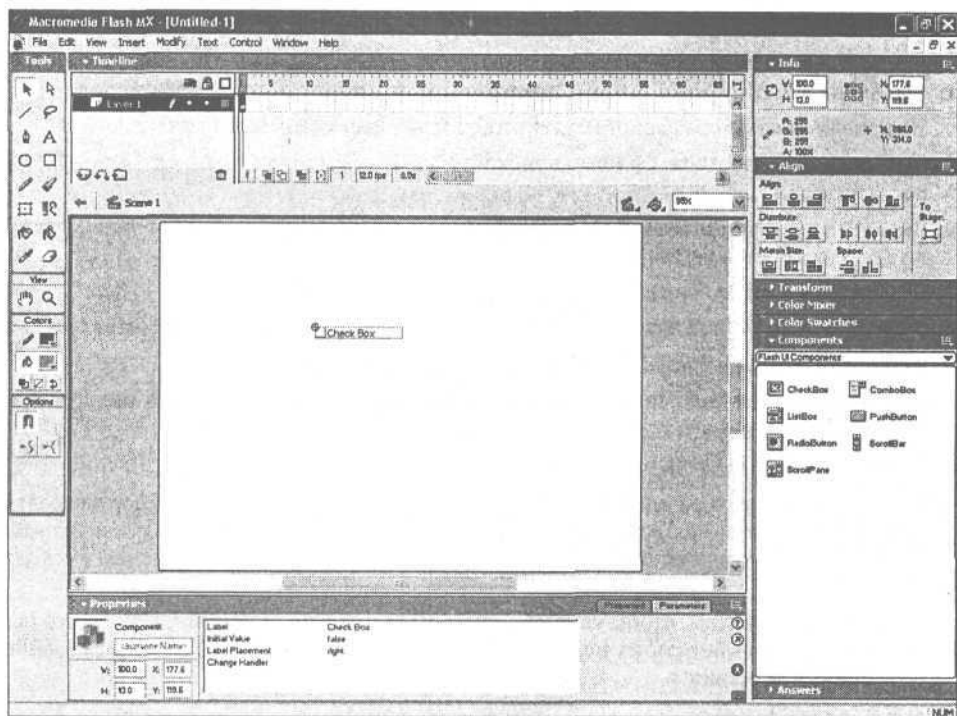


Рис. 22. 1. Для каждого компонента можно определить параметры, управляющие его внешним видом и работой

5. Выделите поле Default Value (Значение по умолчанию) и в качестве первого значения введите 1. Щелкните на знаке "плюс", чтобы вывести следующее значение.
6. Выделите поле Default Value и введите значение 2. Таким же образом введите значения 3, 4 и 5.
7. Щелкните на кнопке ОК, чтобы закрыть окно Values.
8. Протестируйте фильм для проверки работы файла .fla. Вы должны увидеть комбинированное окно с имеющимися элементами для выбора (рис. 22.2).

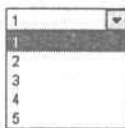


Рис. 22.2. Комбинированное окно позволяет посетителям выбирать нужный элемент из меню с заданными вами пунктами

КОМПОНЕНТ ListBox

Компонент **ListBox** позволяет создать список элементов, которые можно выделить. Пользователи могут выбирать нужные элементы из представленного вами списка. Элементы в поле списка представлены не в виде меню, а в формате списка. Кроме того, в отличие от комбинированного окна, пользователь может выбрать больше одного элемента. Для формирования

поля списка начните с перетаскивания экземпляра компонента `ListBox` в рабочее поле, а затем выполните следующие действия.

1. С помощью инструмента `Aggow` (Стрелка) выделите поле списка в рабочем поле. На панели инспектора свойств будут отображены параметры поля списка.
2. На панели инспектора свойств присвойте имя экземпляру поля списка, например `readoptions`.
3. Выделите параметр `Labels`, а затем щелкните на значке с изображением лупы, в результате чего откроется всплывающее окно `Values` (Значения).
4. Щелкните на знаке "плюс", чтобы вывести новое значение.
5. Выделите поле `Default Value` и в качестве первого значения введите `Television`.
6. Щелкните на знаке "плюс", чтобы вывести следующее значение.
7. Выделите поле `Default Value` и введите `Newspaper`. Таким же образом введите значения `Magazine`, `Book` и `Internet`.
8. Щелкните на кнопке `OK`, чтобы закрыть окно `Values`.
9. На панели инспектора свойств выделите параметр `Select Multiple` (Выбор нескольких элементов), а затем из всплывающего меню выберите пункт `True`. Активизирование данной опции позволит пользователям выбирать из списка одновременно несколько элементов.
10. Протестируйте фильм, чтобы увидеть поле списка в действии. Имейте в виду, что пользователи могут выбирать из него несколько элементов. На рис. 22.3 показан данный компонент в действии.



Рис. 22.3. Компонент `ListBox` позволяет выбрать несколько элементов. Обратите внимание на то, что элементы представлены в виде списка, а не в виде меню

КОМПОНЕНТ CHECKBOX

Помимо того, что флаговые поля позволяют сделать выбор типа "да" или "нет" (а это их наиболее распространенная область применения в `Internet`), еще один способ их применения заключается в предоставлении пользователям возможности выбирать или снимать выделение всех элементов поля списка всего одним щелчком мышью. При наличии в списке большого количества элементов возможность выделения всех их сразу будет для пользователя гораздо лучшей опцией, чем работа с полным полем списка и выбор нужных элементов по отдельности.

Как и при работе с другими компонентами, начните с перетаскивания экземпляра в рабочее поле. Затем выполните следующие действия.

1. Выделите флаговое поле в рабочем поле и присвойте экземпляру компонента имя `checkbox_All`.
2. Выберите параметр `Label`, а затем введите `Select all options` (Выбрать все элементы). Если весь текст надписи не виден на рабочем поле, расширьте флаговое поле с помощью инструмента `Free Transform` (Свободное преобразование).
3. Выберите параметр `Change Handler` (Изменить обработчик), а затем введите `onSelectAll`.

Параметр, указанный в шаге 3, является именем функции, которая вызывается, когда во флаговом поле устанавливается или снимается флажок.

Эту операцию мог бы выполнить приведенный ниже сценарий.

```
function onSelectAll(checkbox) {  
    if (checkbox.getValue() == true) {  
        readoptions.setSelectedIndices([0,1,2,3,4]);  
        readoptions.setEnabled(false);  
    } else {  
        readoptions_lb.setSelectedIndices(null);  
        readoptions_lb.setEnabled(true);  
    }  
}
```

В связке с полем списка, процедура создания которого была описана в предыдущем подразделе, данный сценарий работает следующим образом. Сначала он проверяет, было ли выделено пользователем поле списка, т.е. `checkbox.getValue == true`. Затем вызывается метод `setSelectedIndices` для компонента `ListBox`. Затем этот метод передает массив элементов, определяющих опции поля списка. Число индексов должно быть равно числу опций поля списка, а первым индексом должен быть 0.

После этого вызывается метод `setEnabled` поля списка, который передает значение `false` для отключения поля списка.

При тестировании фильма будут показаны поле списка, рассмотренное в предыдущем подразделе, а также флаговое поле. Параметр `Change Handler` используется для указания функции изменения обработчика, которая вызывается при изменении флагового поля или переключателя.

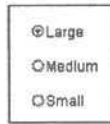
Совет

! Функции были рассмотрены в главе 15 "Объединение предложений в функции".

ПЕРЕКЛЮЧАТЕЛИ

Переключатели позволяют пользователю выбрать одну из нескольких опций. Известные также под названием кнопок групп опций, они могут использоваться для того, чтобы обеспечить пользователю возможность выбрать, например, опции его возраста или подобные, из которых правильной является только одна. Начините с перетаскивания трех экземпляров компонента переключателя в рабочее поле, а затем выполните следующие действия.

1. Выделите первый (верхний) переключатель в рабочем поле и присвойте экземпляру имя, например `Large_rb`.
2. Выберите параметр `Label` и введите `Large`.
3. Выберите параметр `Initial State (Исходное состояние)`, а затем из всплывающего меню выберите значение `True`. Переключатель `Large` будет выделен по умолчанию.
4. Выберите параметр `Group Name`, а затем введите `sizeGroup`.
5. Повторите шаги 1–4 для второго (среднего) переключателя и задайте его параметры следующим образом: введите `medium_rb` в качестве имени экземпляра, `Medium` — для параметра `Label`, `false` — для параметра `Initial State` и `sizeGroup` — для параметра `Group Name`.
6. Повторите шаги 1–4 для третьего (нижнего) переключателя и задайте его параметры следующим образом: введите `small_rb` в качестве имени экземпляра, `Small` — для параметра `Label`, `false` — для параметра `Initial State` и `sizeGroup` — для параметра `Group Name`.
7. Протестируйте файл `.fla`. Вы должны увидеть три переключателя с выбранной опцией `Large`. Щелкните в полях остальных опций и убедитесь, что их также можно выбрать. Созданная группа опций изображена на рис. 22.4.



*Рис. 22.4. Группа переключателей позволяет выбрать только одну опцию. Если выбирается несколько опций, проверьте, не является ли параметр **Group Name** для всех переключателей одним и тем же*

НАЖИМНЫЕ КНОПКИ

Нажимные кнопки используются для отправки формы на сервер или для сброса данных формы. Независимо от того, создана ли форма с помощью Flash или какой-то другой Web-технологии, кнопка является последним элементом, на котором щелкает пользователь перед отправкой данных. Большинство форм имеет две кнопки: одну — для отправления формы, а другую — для сброса данных в случае ошибки. Для создания нажимной кнопки выполните следующие действия.

1. Перетащите экземпляр нажимной кнопки в рабочее поле и выделите его. Параметры экземпляра будут указаны на панели инспектора свойств.
2. В текстовое поле Instance Name введите **btn_reset**.
3. Для параметра Label введите значение Reset.
4. Для параметра Click Handler (Обработчик щелчка) введите **onReset**. Это — имя функции, которая будет вызываться после щелчка пользователем на кнопке Reset (Сброс).
5. Повторите шаги 1–4 для второй нажимной кнопки и задайте ее параметры следующим образом: введите **btn_submit** в качестве имени экземпляра, **submit** — для параметра Label и **onSubmit** — для параметра Click Handler. На рис. 22.5 изображены две кнопки со свойствами для кнопки Submit.
6. При тестировании фильма будут изображены две кнопки, на каждой из которых можно щелкнуть. Попробуйте щелкнуть на кнопке Reset, чтобы убедиться, что при этом будут сброшены все остальные компоненты.

ПРОКРУЧИВАЮЩЕЕСЯ ТЕКСТОВОЕ ПОЛЕ

В прокручиваемом текстовом поле, обычно расположенном в нижней части формы, отображается сообщение обратной связи с пользователем, либо же оно используется для приема от пользователя текста произвольной формы. Ниже описано, как задать компонент для отображения сообщения в конце процедуры заказа товара. В программе не существует отдельного компонента Scrolling Text Box (Прокручиваемое текстовое поле), и поэтому вы можете прийти в замешательство, не зная, с чего начать. Не волнуйтесь. Вскоре вы увидите, что процедура вставки на самом деле очень проста.

1. Выделите инструмент Text (Текст). Затем откройте панель инспектора свойств, если она до сих пор не отображена. Для этого из меню Window (Окно) выберите пункт Properties (Свойства).
2. Присвойте экземпляру имя **txtmessage**.
3. Из раскрывающегося меню Type (Тип) выберите пункт Dynamic (Динамический), а затем щелкните на кнопке Show Border Around Text (Отобразить границу вокруг текста).
4. Из раскрывающегося меню Line Type (Тип строк) выберите пункт Multiline (Многострочный).

5. С помощью панели инспектора свойств измените размеры поля сообщения так, чтобы оно соответствовало макету формы. Размер может быть любым, который вы считаете подходящим.
6. Перетащите экземпляр компонента ScrollBar (Полоса прокрутки) и отпустите его справа от поля сообщения. Полоса прокрутки автоматически присоединится к текстовому полю, делая его прокручиваемым.
7. Выделите полосу прокрутки, которая присоединена к текстовому полю Messages (Сообщения) в рабочем поле. Ее свойства будут отображены на панели инспектора свойств. В качестве значения параметра Target TextField (Целевое текстовое поле) введите `txtmessage`, как показано на рис. 22.6.

На заметку

Во Flash MX динамические текстовые поля и поля ввода данных являются экземплярами объекта ActionScript TextField, и имена им могут быть присвоены так же, как видеоклипам и другим объектам.

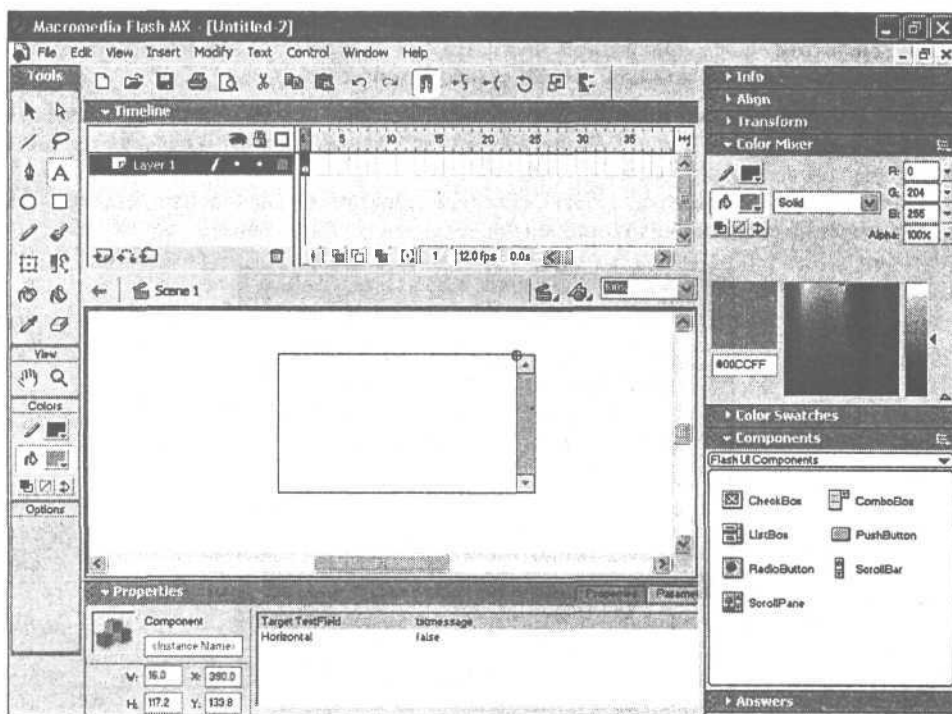


Рис. 22.6. К полосе прокрутки добавляются свойства, а целевое текстовое поле совпадает с именем текстового поля, в котором размещена полоса прокрутки

ПОЛОСА ПРОКРУТКИ

Из самого названия компонента следует, что полоса прокрутки позволяет пользователю прокручивать текстовое поле. Как вы узнали в предыдущем подразделе, в котором описывалась процедура создания прокручиваемого текстового поля, данную функциональность реализует компонент ScrollBar (Полоса прокрутки).

Если перетащить компонент ScrollBar с панели Components (Компоненты) и отпустить его справа от текстового поля, полоса прокрутки автоматически присоединится к текстовому полю.

Параметр Target TextField компонента ScrollBar указывает имя экземпляра объекта TextField, к которому присоединяется полоса прокрутки.

Экземплярю полосы прокрутки имя присваивать не нужно.

ИСПОЛЬЗОВАНИЕ ПАНЕЛИ LIBRARY С КОМПОНЕНТАМИ

При добавлении компонента в файл на панель Library (Библиотека) добавляется папка Flash UI Components (Компоненты пользовательского интерфейса Flash), как показано на рис. 22.7.

Панель Library имеет следующие элементы.

- Видеоклип компонента, представленный в виде пиктограммы типа компонента.
- Папка Component Skins (Оболочки библиотеки) с подпапкой Global Skins (Глобальные оболочки), содержащей графические элементы, применимые ко всем компонентам, а также подпапкой Skins (Оболочки) для отдельных типов компонентов.
- Папка Core Assets (Основные средства), содержащая средства для опытных разработчиков, в том числе прикладной программный интерфейс Data Provider (Источник данных) и иерархию классов, используемую компонентами.

В папке Skins содержатся графические символы, которые используются для отображения типа компонента в файле. В папке Global Skins (Глобальные оболочки) содержится информация, общая для всех компонентов.

Все оболочки можно отредактировать обычным образом. Очевидно, что редактирование оболочек влияет на внешний вид компонентов, имеющих в документе.

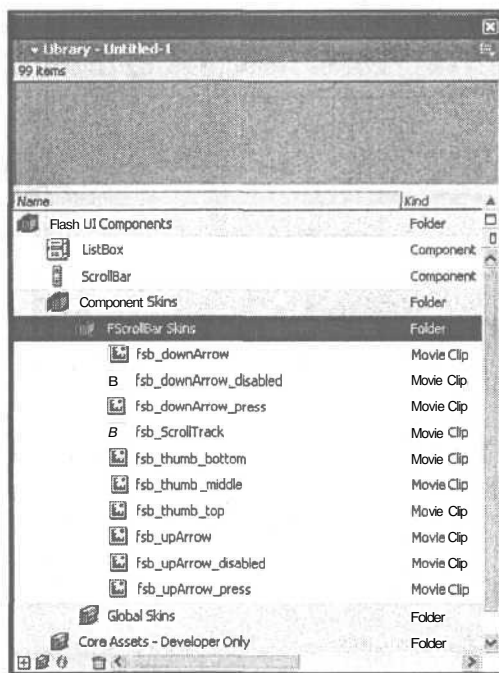


Рис. 22.7. При использовании компонента на панели Library отображаются данные о нем

РЕДАКТИРОВАНИЕ ОБОЛОЧЕК

Редактировать оболочки для изменения внешнего вида компонентов в рабочем поле можно точно так же, как и при работе с любой другой графикой. Можно также создать новые графические изображения и разбить их на элементы оболочки.

Если вы предпочитаете создать новое графическое изображение и разбить его на части, то его элементы необходимо зарегистрировать как компоненты. Это делается путем редактирования кода в первом кадре слоя **Read Me** каждой оболочки библиотеки.

При выполнении редактирования вы изменяете все экземпляры компонентов, в которых используются отредактированные оболочки. Изменять оболочки отдельных экземпляров компонентов нельзя.

СОЗДАНИЕ НОВЫХ КОМПОНЕНТОВ

До сих пор мы рассматривали встроенные компоненты Flash MX. Вы, вероятно, обратили внимание, что все они являются полями форм, которые можно редактировать в соответствии с потребностями форм. Однако компоненты могут быть всевозможными — программируемыми кнопками, полями форм самых разных размеров и контуров, которыми можно управлять, или элементами любых типов, которые могут использоваться неоднократно, с разными параметрами. Чтобы увидеть эти компоненты в действии, просмотрите кнопки компонентов в библиотеке Flash.

В этом подразделе мы рассмотрим создание прямоугольника с программируемыми опциями изменения ширины и высоты. Хотя при создании настоящего компонента этого, возможно, и не стоит делать, для иллюстрации процесса мы создадим прямоугольник, имеющий такие изменяющиеся свойства: ширину, высоту, окраску и еще один параметр, который позволяет управлять применением окраски.

НАЧАЛО СОЗДАНИЯ СОБСТВЕННОГО КОМПОНЕНТА

Каждый компонент начинается одним и тем же образом: с видеоклипа. После создания видеоклипа производится добавление нужного кода **ActionScript** для класса, свойств и методов прямоугольника. Затем устанавливается связь видеоклипа с определением класса.

Совет

Подробная информация о видеоклипах приведена в главе 17 "Демонстрация мощи видеоклипов".

Начнем с создания видеоклипа и присвоим ему имя **Rectangle Component** (Компонент прямоугольника). В пределах видеоклипа нарисует прямоугольник. Залить его можно любым цветом. С помощью панели инспектора свойств зададим ширину и высоту прямоугольника: 15x30 пикселей.

Откроем панель **Align** (Выравнивание) и выровняем прямоугольник по центру горизонтально и по нижнему краю вертикально. Проверьте, чтобы опция **Align To** (Выровнять по) была установлена в значение **To Stage** (По рабочему полю). Поскольку мы хотим иметь возможность масштабировать прямоугольник, то необходимо знать, что основание прямоугольника будет оставаться статичным. На рис. 22.8 показан правильно выровненный прямоугольник нужных размеров.

ОПРЕДЕЛЕНИЕ КЛАССА ПРЯМОУГОЛЬНИКА

Процедура группирования данных об объекте называется созданием класса. К процедуре создания класса относится и определение свойств объекта. В данном случае мы определяем свойства прямоугольника.

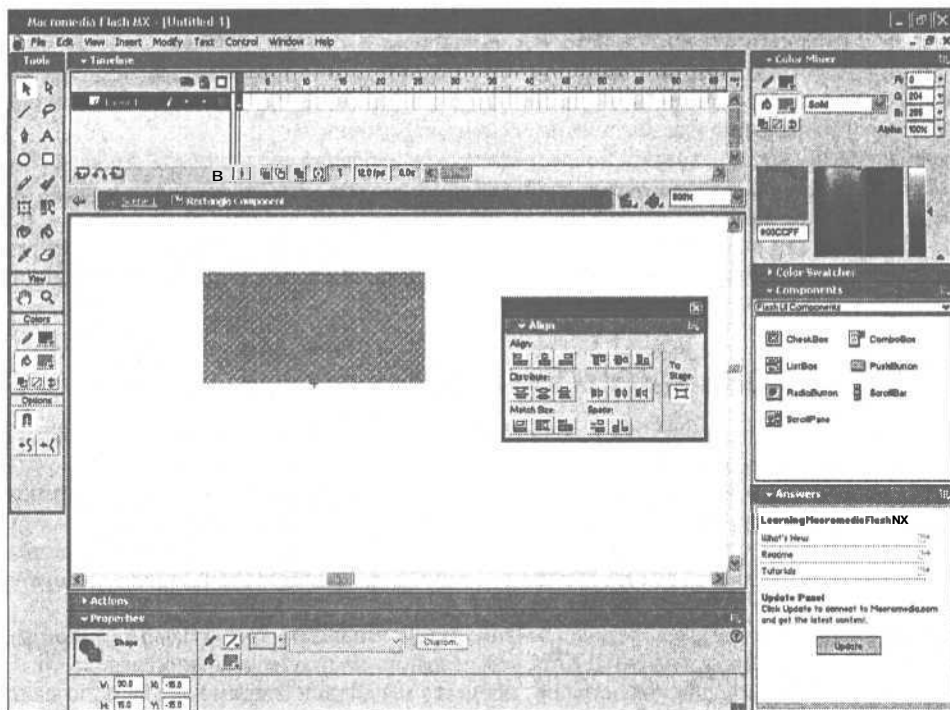


Рис. 22.8. Уверенность в правильном выравнивании при создании компонента означает, что при изменении размеров прямоугольника его основание останется правильно позиционированным

Совет

Подробная информация о классах приведена в главе 15 "Объединение предложений в функции".

Следующий шаг заключается в создании нового слоя видеоклипа и присвоении ему имени **Actions**. Откройте панель **Actions** в режиме **Expert**, а затем добавьте следующий код:

```
#initclip
function RectangleClass() {
    this.cobj = new Color(this);
    this.update();
}

// Разрешить RectangleClass наследовать свойства класса MovieClip
RectangleClass.prototype = new MovieClip();

// Обновить начертание прямоугольника с текущим основанием и значения-
ми высоты
RectangleClass.prototype.update = function () {
    if (this.applyTint) {
        this.cobj.setRGB(this.rcolor);
    }
    this._xscale = 30 * this.rbase;
    this._yscale = 15 * this.rheight;
}

// Связать класс с идентификатором связи для данного видеоклипа
Object.registerClass("Frectangle", RectangleClass);
#endinitclip
```

Код определяет класс компонента. Прежде чем продолжить, рассмотрим отдельные части этого кода, которые можно перенастроить и использовать для других компонентов.

ПРЕДЛОЖЕНИЯ #INITCLIP и #ENDINITCLIP



Новые предложения Flash MX, `tinitclip` и `ttendinitclip`, используются исключительно при создании и определении компонентов. При создании компонента вы определяете конструктор класса и методы, а затем регистрируете класс.

Предложения `#initclip` и `#endinitclip` размечают код определения компонента. Код внутри предложений при воспроизведении фильма выполняется только один раз. Кроме того, он всегда выполняется перед любым другим кодом во временной шкале фильма. Это большой шаг вперед по сравнению с ограничениями, существовавшими во Flash 5. Раньше приходилось ждать полной загрузки ключевого кадра, и только после этого становились доступными определенные методы; теперь это происходит сразу же.

ОПРЕДЕЛЕНИЕ КОНСТРУКТОРА

Показанная здесь функция `RectangleClass()` определяет конструктор компонента. Когда компонент создан во Flash, вызывается конструктор и выполняются действия.

```
function RectangleClass()
```

Совет

Подробная информация о конструкторах приведена в главе 15 "Объединение предложений в функции".

СОЗДАНИЕ НАСЛЕДОВАНИЯ

Сразу же после конструктора должна быть указана приведенная ниже строка кода; она должна стоять перед определением любых методов класса.

```
RectangleClass.prototype = new MovieClip();
```

При создании команды о присоединении определения класса к видеоклипу Flash автоматически присваивает тип видеоклипа типу (в данном случае — класса), определенному вами. Проблема с реализацией данной функциональности заключается в том, что фактически вы хотите, чтобы видеоклип функционировал как видеоклип.

Чтобы обойти эту проблему, используется приведенный в этом подразделе код, который указывает, что новый класс должен наследовать методы и свойства видеоклипов.

ОБНОВЛЕНИЕ ПРЯМОУГОЛЬНИКА

В приведенном ниже методе используются текущие значения ширины и высоты прямоугольника. Метод выясняет, хотите ли вы применить окраску, и, если это так, он выполняется и применяет ее. Умножаемые значения заданы равными 30 и 15, что соответствует размерам нарисованного прямоугольника. Данный метод гарантирует, что масштабирование будет выполняться корректно:

```
RectangleClass.prototype.update = function () {  
    if (this.applyTint) {  
        this.cobj.setRGB(this.rcolor);  
    }  
    this._xscale = 30 * this.rbase;  
    this._yscale = 15 * this.rheight;  
}
```

РЕГИСТРАЦИЯ КЛАССА

Последняя команда, **вероятно**, является наиболее важной. Она связывает класс с клипом в идентификаторе связи:

```
Object.registerClass("Fractangle", RectangleClass);
```

ИСПОЛЬЗОВАНИЕ МЕТОДОВ СЧИТЫВАНИЯ И ЗАПИСИ

Теперь, глядя на созданный ранее код, пришло время двигаться дальше и добавить к нему другие методы и свойства, необходимые для компонента Rectangle (Прямоугольник). Стандартные **операции** подразделяются на два типа: задание и получение. Операции получения возвращают значения свойства, а операции задания перерисовывают прямоугольник при изменении значений свойств.

На панель Actions (Действия) следует добавить приведенный ниже код. Его необходимо разместить после предложения наследования, но до окончания блока `#endinitclip`. В этом коде нет ничего такого, что не рассматривалось бы раньше, поэтому разбивать его на части и анализировать каждую из них мы не будем. На этом этапе код уже говорит сам за себя.

```
RectangleClass.prototype.setBase = function (b) {  
    this.rbase = b;  
    this.update();  
}  
  
RectangleClass.prototype.getBase = function () {  
    return (this.rbase);  
}  
  
RectangleClass.prototype.setHeight = function (h) {  
    this.rheight = h;  
    this.update();  
}  
  
RectangleClass.prototype.getHeight = function () {  
    return (this.rheight);  
}  
  
RectangleClass.prototype.setTintColor = function (c) {  
    this.rcolor = c;  
    this.update();  
}  
  
RectangleClass.prototype.turnOnTint = function () {  
    this.applyTint = true;  
    this.update();  
}
```

ОПРЕДЕЛЕНИЕ СВОЙСТВ КОМПОНЕНТОВ

Может возникнуть вопрос, почему до сих пор еще ничего не сделано для преобразования видеоклипа в компонент. Причина этого очень проста. Она заключается в том, что существует логический порядок создания компонентов и поэтому сначала добавляется сценарий и подобные элементы. Теперь, когда мы имеем прямоугольник, можно определить его параметры.

Откройте панель Library (Библиотека) и выделите фильм компонента Rectangle. Откройте контекстное меню и выберите из него пункт Component Definition (Определение компонента), в результате чего откроется одноименное диалоговое окно, изображенное на рис. 22.9.

Для добавления параметров воспользуйтесь кнопками с изображениями знаков "плюс" и "минус", расположенными в верхней части диалогового окна. По умолчанию добавляемые

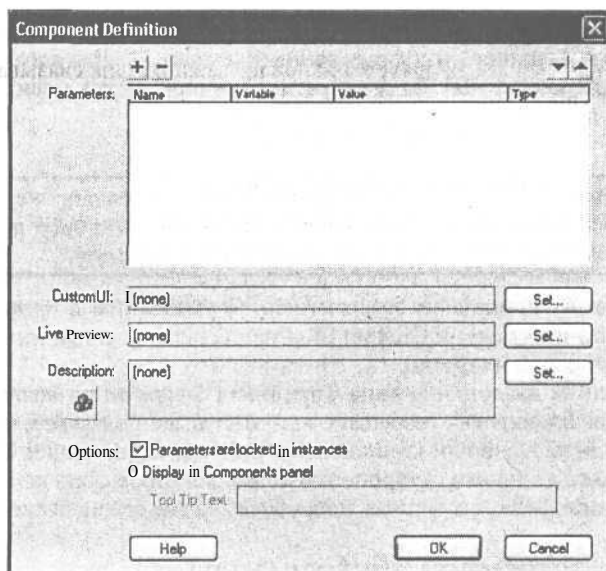


Рис. 22. Р. 5 диалоговом окне *Component Definition* можно задавать параметры любого созданного компонента

свойства относятся к типу Default (По умолчанию) и значению **default value** (Значение по умолчанию). Во Flash можно добавить параметры следующих типов:

- Default (По умолчанию);
- Array (Массив);
- Object (Объект);
- List (Список);
- String (Строка);
- Number (Число);
- Boolean (Булева величина);
- Font Name (Имя шрифта);
- Color (Цвет).

Окошко в верхней части диалогового окна разбито на четыре столбца. В столбец Name (Имя) добавляется текст, который будет отображаться на панели инспектора свойств. В столбце Variable (Переменная) указывается имя, которое будет использоваться в сценариях компонента. Имя переменной должно соответствовать стандартным правилам наименования переменных (без пробелов и т.п.). В столбце Value (Значение) указывается начальное значение свойства, а в столбце Type (Тип) — тип параметра, соответствующий приведенному выше списку.

Для созданного компонента Rectangle следует добавить свойства, указанные в табл. 22.1.

ТАБЛИЦА 22.1. СВОЙСТВА КОМПОНЕНТА RECTANGLE

Имя	ПЕРЕМЕННАЯ	ЗНАЧЕНИЕ	Тип
Base Length (Длина основания)	rbase	30	Number (Число)
Height (Высота)	rheight	15	Number (Число)
Tint Color (Цвет окраски)	rcolor	#000000	Color (Цвет)
Apply Tint (Применить окраску)	applyTint	false	Boolean (Булева величина)

К компоненту можно добавить текстовое описание, которое позволит остальным пользователям понять, какую функцию выполняет данный компонент. Для добавления текстового описания щелкните на кнопке Set (Задать), расположенной рядом с текстовым полем Description (Описание), а затем введите нужное описание.

XML-описания

В качестве альтернативы текстовому описанию, чтобы добавить на панель Actions пользовательское действие, можно создать XML-документ. XML-файл будет доступным, если он размещен в папке `Configuration/ActionsPanel/CustomActions`.

Оставим пиктограмму компонента назначенной по умолчанию и проигнорируем опции Live Preview (Текущий просмотр) и Custom UI (Специализированный интерфейс пользователя). Эти опции будут описаны кратко.

Предпоследняя опция диалогового окна Component Definition позволяет управлять блокировкой параметров. Блокировка разрешает или запрещает пользователям добавлять или удалять параметры при размещении компонента в рабочем поле. Опция Display in Components Panel (Отобразить на панели Components) позволяет отображать компонент на панели Components. Установите флажок в ее поле. Подробнее данные опции будут описаны ниже.

ДОБАВЛЕНИЕ СОБСТВЕННОЙ ПИКТОГРАММЫ

Одной из опций, которая упоминалась в предыдущем подразделе, является возможность создания собственной пиктограммы для компонента. Если она будет создана, то именно она и будет отображаться на панелях Library (Библиотека), Movie Explorer (Проводник фильма) или Components (Компоненты) (если вы укажете отображать там компоненты).

Для добавления пользовательской пиктограммы сначала необходимо создать ее графическое изображение либо в Fireworks, либо в другом приложении для создания графики. Можно даже создать графический файл во Flash и экспортировать его в формате PNG.

Чтобы пиктограмма отображалась корректно, ее размеры не должны превышать 20x20 пикселей.

После создания и сохранения файла его необходимо импортировать в библиотеку. Чтобы быть уверенным в том, что Flash распознает этот файл как пользовательскую пиктограмму, в библиотеке сначала необходимо создать новую папку. Назовите эту папку **FCustomIcons** и импортируйте в нее графическое изображение.

Закройте и снова откройте панель Library. Теперь рядом с компонентами должна появиться новая пиктограмма, как показано на рис. 22.10.

ЗАДАНИЕ ИДЕНТИФИКАТОРА СВЯЗИ

Следующий шаг заключается в закрытии видеоклипа и возврату к основной сцене фильма. Для задания идентификатора связи щелкните правой кнопкой мыши на новом компоненте на панели Library и из контекстного меню выберите пункт Properties. При этом откроется окно Symbol Properties (Свойства символа). Щелкните на кнопке Advanced (Дополнительно), чтобы открыть расширенное диалоговое окно.

Установите флажок в поле опции Export for ActionScript (Экспортировать для ActionScript), а все остальные поля оставьте пустыми. Затем в поле Identifier (Идентификатор) введите ID, ранее созданный для прямоугольника. В данном случае введите Frectangle, как показано на рис. 22.11.

ТЕСТИРОВАНИЕ

В основном разработка компонента завершена, код ActionScript добавлен. Теперь пришло время узнать, что произойдет при использовании нового компонента.



Рис. 22.10. При создании пользовательской пиктограммы Flash отобразит ее на панелях Library, Movie Explorer и Components

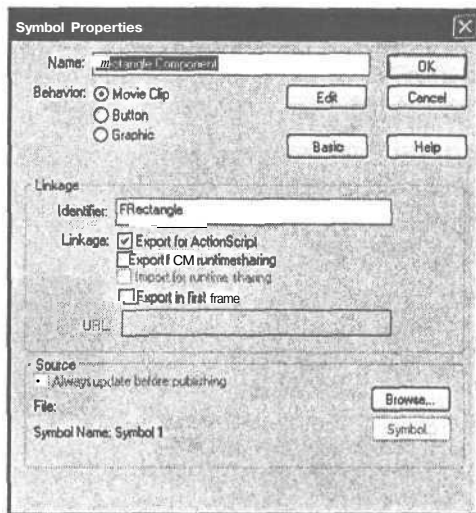


Рис. 22.11. Идентификатор связи добавляют в диалоговом окне Symbol Properties. Необходимо совпадение с созданным ранее идентификатором, в данном случае **FRectangle**

Перетащите экземпляр компонента из библиотеки в рабочее поле. При тестировании фильма на этом этапе будет показан прямоугольник в том виде, в каком он был нарисован.

Для редактирования свойств компонента можно воспользоваться панелью инспектора свойств. Чтобы отредактировать отображенные значения, щелкните в соответствующем столбце и измените числа. В данном случае были изменены параметры ширины и высоты компонента. Обратите внимание на то, что изменения отображаются только при тестировании фильма. На этапе разработки их не видно. На рис. 22.12 изображен измененный прямоугольник. Кроме того, можно поэкспериментировать с параметрами окраски, изменяя значение **false** на **true** и выбирая цвет окраски.

УСТАНОВКА КОМПОНЕНТА НА ПАНЕЛИ COMPONENTS

Чтобы установить компонент на панели Components, начните с добавления компонента, его пиктограммы и любых других символов, использующихся с данным компонентом, в новую папку панели Library. Этот шаг гарантирует, что все элементы будут храниться вместе. На рис. 22.13 изображена вновь созданная папка, в которой хранятся видеоклип, папка пользовательских пиктограмм и пиктограмма компонента.

Для установки компонента необходимо разместить файл **.fla** в нужном месте. После этого при запуске панели Flash загрузит в нее данный компонент.

Место размещения файла **.fla** зависит от операционной системы и используемых настроек. Данные размещаются в папке **Configuration**, отдельной для каждого пользователя и расположенной в месте хранения операционной системой информации о приложениях.

На заметку

Если папка **Configuration\Components** не видна, проверьте, чтобы для всех скрытых файлов и папок было выбрано значение **visible** (видимые).

Некоторые примеры приведены в табл. 22.2.

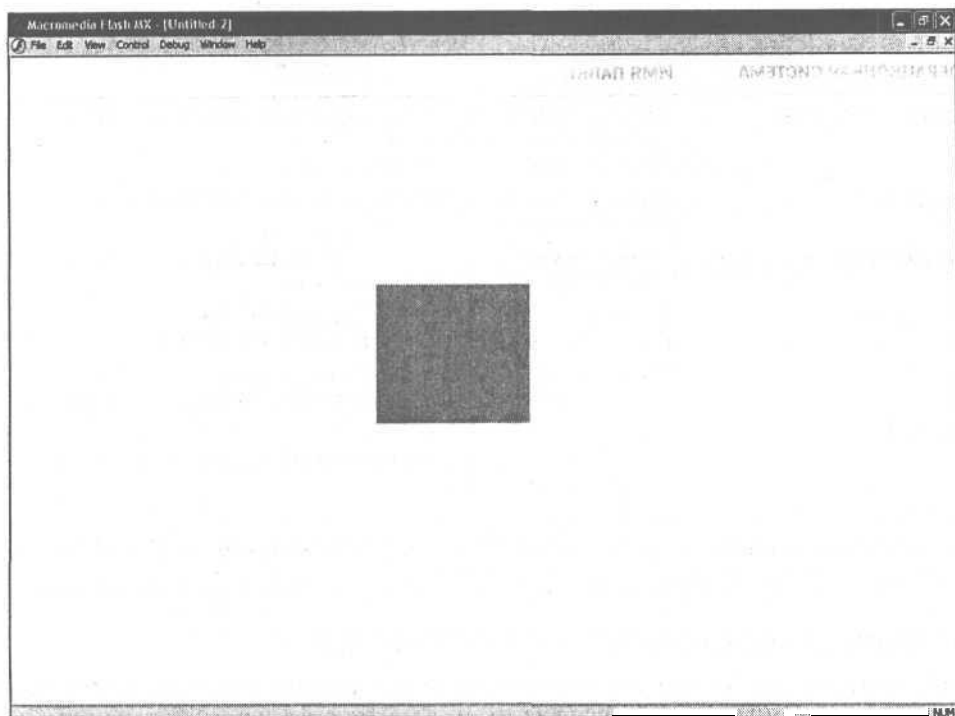


Рис. 22.12. При тестировании фильма видно, как изменились свойства компонента

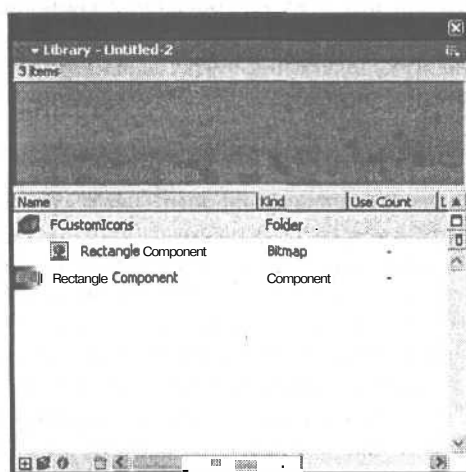


Рис. 22.13. Поместив все элементы компонента в одну папку, вы облегчите жизнь и себе, и другим пользователям данного компонента

ТАБЛИЦА 22.2. МЕСТА УСТАНОВКИ КОМПОНЕНТОВ

ОПЕРАЦИОННАЯ СИСТЕМА	Имя ПАПКИ
Windows 2000 и XP	C:\Documents and Settings\User\Application Data\Macromedia\Flash MX\Configuration\Components\
Windows XP	C:\Program Files\Macromedia\Flash MX\First Run\Components\
Windows 98 и ME	C:\Windows\Application Data\Macromedia\Flash MX\Configuration\Components\
Windows NT	C:\WinNT\Profiles\User\Application Data\Macromedia\Flash MX\Configuration\
Mac OS X	Hard Drive:Users:Library:Application Support:Macromedia:FlashMX:Configuration:Components
Mac OS 9.1	Hard Drive:Users:User:Documents:Macromedia:FlashMX:First Run:Components:
Mac OS 9.x (многопользовательская)	Hard Drive:Users:User:Documents:Macromedia:FlashMX:Configuration:Components:

СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

До настоящего момента мы занимались созданием компонента, рассматривали свойства и вставляли код там, где было необходимо. Одно из самых лучших качеств Flash MX заключается в возможности управления вопросами удобства и простоты использования элементов программы. Создание пользовательского интерфейса является одним из примеров реализации этой возможности.

Пользовательский интерфейс представляет собой файл Flash-фильма, который позволяет пользователям вводить значения свойств, такие как, например, ширина и высота, без обращения к панели инспектора свойств. Созданный интерфейс можно запустить с панели инспектора свойств или с панели Component Parameters (Параметры компонентов).

Фильм интерфейса должен иметь возможность получения значений, задаваемых пользователем. В данном случае такими значениями являются `rbase`, `rheight`, `rcolor` и `ApplyTint`.

Ключевым элементом, который необходимо иметь в пользовательском интерфейсе, является экземпляр создаваемого видеоклипа, именуемый `xch`. Передача значений компоненту осуществляется путем записи пользовательским интерфейсом значений в свойства экземпляра `xsp`, например `xch.tbase = 75;`.

Интерфейс должен позволять пользователям вводить нужные им значения так, чтобы при тестировании или публикации компонента значения `xch` копировались в компонент.

Для создания пользовательского интерфейса выполните следующие действия.

1. Создайте новый Flash-фильм и задайте его размеры равными 70х400 пикселей. Таковы ориентировочные размеры панели инспектора свойств.
2. Вставьте в фильм пустой видеоклип и назовите его `Change`.
3. Поместите экземпляр клипа в рабочее поле и назовите его `xch`.
4. Создайте в фильме текстовые поля для параметров ширины, высоты, красного, зеленого, синего цветов и для значений 0-255.
5. Определите используемые текстовые поля как предназначенные для извлечения значений свойств `rbase` и `rheight`. Создайте поля для ввода данных основания и высоты,

затем разместите их над соответствующим полем статического текста и задайте имена экземпляров соответственно как **bs** и **ht**.

6. Задайте текстовые поля ввода данных, предназначенные для извлечения **цвета** окраски. Создайте **еще** три текстовых поля для ввода данных, присвойте им имена **rc**, **gc**, и **bc** (соответственно для красного, зеленого и синего цветов) и разместите их над соответствующим полем статического текста.
7. Чтобы пользователь мог предварительно просмотреть цвет, создайте элемент мозаичного изображения, который указывает текущий параметр окраски. Для создания квадрата воспользуйтесь инструментом рисования. Преобразуйте этот квадрат в видеоклип и присвойте экземпляру имя **tintSq**.
8. Чтобы указать, хочет ли пользователь применить цвет окраски, можно воспользоваться предварительно определенным Flash-компонентом **fCheckBox**. Откройте панель Components и перетащите компонент флагового поля в нужное место рабочего поля. В окне **Properties** (Свойства) задайте текст **Apply Tint?** (Применить окраску?), а значение по умолчанию для него установите равным **false**. Для свойства **Change Handler** (Обработчик изменения) введите **tintApply**.

Последним шагом будет написание кода специализированного пользовательского интерфейса. В кадр 1 нового слоя введите следующий код:

```
// Задать значения свойств по умолчанию
xch.rbase = 30;
xch.rheight = 15;
xch.rcolor = 0x000000;
xch.applyTint = false;
// Задать исходные текстовые поля для основания и высоты
bs.text = 30;
ht.text = 15;
// Задать исходный мозаичный элемент окраски и текстовые поля цветов
tsc = new Color(tintSq);
rc.text = gc.text = bc.text = "0";
setTintSq(0, 0, 0);

// Задав значения красного, зеленого и синего цветов, создайте
// полное
// значение цвета, задайте мозаичный элемент окраски и верните
// полное значение цвета.
function setTintSq(r, g, b) {
    var col = r << 16 | g << 8 | b;
    tsc.setRGB(col);
    return col;
}

// Задав целое число c, привяжите его к значениям в диапазоне от 0
// до 255
function boundColor (c) {
    if (isNaN(c))
        return 0;
    if (c < 0) {
        c = 0;
    } else if (c > 255) {
        c = 255;
    }
    return c;
}
```

```
// При изменении текстовых полей цветов измените цвет окраски
// и задайте соответствующее свойство в экземпляре xch
rc.onChanged = function () {
    xch.tcolor = setTintSq(boundColor(this.text), ParseInt(gc.text),
    ParseInt(bc.text));
}
gc.onChanged = function () {
    xch.tcolor = setTintSq(ParseInt(rc.text),
    boundColor(this.text), ParseInt(bc.text));
}
bc.onChanged = function () {
    xch.tcolor = setTintSq(ParseInt(rc.text),
    ParseInt(bc.text), boundColor(this.text));
}

// Флаговое поле (c) вызывает эту функцию, когда пользователь ус-
танавливает
// или снимает флажок. Соответственно задайте свойство applyTint
экземпляра xch
function tintApply (c) {
    xch.applyTint = c.getValue();
}

RectangleClass.prototype.getArea = function ()
{
    return (0.5 * this.tbase * this.theight);
}
```

9. Сохраните фильм на жестком диске и протестируйте его. При этом будет создан файл `.swf`, расположенный там же, где и сохраненный фильм.

На заметку

Если вы планируете сохранить свой компонент как фильм формата Flash 5, то связь с ним необходимо установить как с внешним, а не как с внедренным файлом.

10. Объедините специализированный пользовательский интерфейс с фильмом Rectangle. Для этого вернитесь к фильму Rectangle, откройте панель Library (Библиотека) и выберите компонент Rectangle. Далее откройте диалоговое окно Component Definition (Определение компонента) и щелкните на кнопке Set (Задать), расположенной справа от поля опции Custom UI (Специализированный пользовательский интерфейс).
11. В группе опций Type (Тип) установите переключатель в поле Custom UI with `.swf` file embedded in `.fla` file (Специализированный пользовательский интерфейс с файлом `.swf`, внедренным в файл `.fla`). Другая опция, Custom UI in external `.swf` file (Специализированный пользовательский интерфейс во внешнем файле `.swf`), позволяет хранить специализированный пользовательский интерфейс вне прямоугольника (это хорошо для тестирования, но не для реализации). При выборе данной опции необходимо не только ввести абсолютный путь к файлу (это означает, что в случае использования компонента на другой платформе путь должен быть откорректирован), но и распределить оба файла.
12. В группе опций Display (Отображение) установите переключатель в поле Display in Property Inspector (Отобразить в инспекторе свойств).
13. Щелкните на кнопке Browse, расположенной под полем Custom UI `.swf` file (Файл `.swf` специализированного пользовательского интерфейса), и укажите местоположе-

ние файла `.swf` либо щелкните на кнопке Update (Обновить), если местоположение данного файла уже указано и вы хотите перезагрузить пользовательский интерфейс.

- Щелкните на кнопке ОК. Теперь при выделении прямоугольника в рабочем поле и открытом окне Properties вы должны увидеть свой пользовательский интерфейс.

На заметку

Если специализированный пользовательский интерфейс сразу не виден, попробуйте закрыть и перезагрузить Flash. Прежде чем переходить к предварительному просмотру текущих данных, убедитесь, что специализированный пользовательский интерфейс виден.

Теперь при редактировании свойств компонента вы должны увидеть отображенный пользовательский интерфейс во всей красе и за работой.

СОЗДАНИЕ ФИЛЬМА ТЕКУЩЕГО ПРОСМОТРА

Фильм **текущего** просмотра позволяет пользователю увидеть, как будет выглядеть компонент при публикации. Как мы уже заметили, изменения, вносимые посредством пользовательского интерфейса, во время разработки не отображаются. Использование фильма текущего просмотра позволяет пользователю вносить изменения в параметры и сразу же видеть их.

Фильм текущего просмотра предназначен для просмотра только основных характеристик компонента, таких как цвет и размер, что очень подходит для рассматриваемого в данной главе компонента прямоугольника. Однако данный режим не очень подходит для предварительного просмотра анимации, поскольку фильм текущего просмотра выполняется со скоростью всего 1 кадр в секунду.

В отличие от специализированного пользовательского интерфейса, устанавливающего значения в пределах объекта `xch` и передающего их в компонент, фильм текущего просмотра считывает значения прямо из объекта. Текущий просмотр реализуется путем активизации функции, вызываемой посредством определенного вами события `onUpdate`.

Flash активизирует функцию `onUpdate` при внесении изменений, а пользователю сообщается, когда предварительный просмотр требует обновления.

Перед детальным анализом и созданием текущего просмотра необходимо на секунду остановиться и подумать об отличиях между компонентом и фильмом текущего просмотра.

В большинстве случаев таких отличий будет совсем немного, и при использовании экземпляра фактического компонента все будет работать наилучшим образом. Однако при применении к прямоугольнику цвета окраски вы не сможете его перемещать (можете только изменять цвет). Поэтому было бы очень кстати, если бы пользователи могли видеть прямоугольник как с примененной окраской, так и без нее.

Для достижения этой цели в режиме текущего просмотра создайте два прямоугольника. Исходный прямоугольник будет отображаться без применения окраски, а второй прямоугольник — с окраской. Затем укажите, какой из них следует отображать, исходя из выбора пользователя. Для создания фильма предварительного просмотра выполните следующие действия.

- Создайте новый Flash-фильм с теми же размерами, что и у прямоугольника, — 15x30 пикселей.
- Добавьте в файл два слоя. Один из них назовите **Tinted** (Окрашенный), а другой — **Plain** (Обычный).
- В слое **Plain** создайте прямоугольник, имеющий тот же цвет, размеры и т.п., что и компонент.
- Преобразуйте прямоугольник в видеоклип с именем **plain rectangle** и присвойте его экземпляру имя **rp** (Обычный прямоугольник).

5. В слой Tinted поместите копию экземпляра **rp** и переименуйте ее, присвоив имя **rtint**.
6. Создайте слой **Actions** (Действия), в котором будет содержаться ранее упоминавшаяся функция **onUpdate**. Эта функция будет считывать значения свойств и соответственно корректировать размеры прямоугольника. Код сообщит функции **onUpdate**, следует ли отображать окрашенный или обычный прямоугольник.
7. В кадр 1 слоя Actions добавьте **следующий** код:

```
// Данная программа вызывается, когда свойства компонента были изменены.
// Перерисуйте прямоугольник с новыми значениями. Если применена окраска,
// используйте прямоугольник rtint. Если окраску не нужно применять,
// используйте прямоугольник rp.
function onUpdate() {
    var ac = _root.xch.applyTint, b = _root.xch.rbase, h =
    _root.xch.rheight;

    tc._xscale = 30 * b;
    tc._yscale = 15 * h;
    tctint._xscale = rp._xscale;
    tctint._yscale = rp._yscale;

    if (ac) {
        rc_color.setRGB(_root.xch.tcolor);
    }
    rtint._visible = ac;
    rp._visible = !ac;
}
// Есть две копии прямоугольника: rp — для предварительного просмотра
// обычного
// неокрашенного прямоугольника
// и rtint — для предварительного просмотра окрашенного прямоугольника.
// По умолчанию отображается только обычный прямоугольник.
rtint._visible = false;
// Применить цвет Color для окраски.
rp_color = new Color(rtint);
```

8. Убедитесь, что текущий просмотр разрешен. В нижней части меню Control (Управление) должен быть помечен галочкой пункт Enable Live Preview (Разрешить текущий просмотр). Если галочки нет, добавьте ее.
9. Сохраните файл и протестируйте его, в результате чего будет создан файл **.swf**, необходимый для связи с компонентом.
10. Откройте диалоговое окно Component Definition (Определение компонента) для компонента Rectangle и щелкните на кнопке Set (Задать), расположенной рядом с полем Live Preview (Текущий просмотр).
11. Установите переключатель в поле опции Live Preview with **.swf** file embedded in **.fla** file (Текущий просмотр с файлом **.swf**, внедренным в файл **.fla**). При выборе данной опции файл предварительного просмотра будет связан с компонентом. Дважды щелкните на кнопке ОК, чтобы закрыть диалоговое окно Component Definition. Когда вы вернетесь к рабочему полю, выполняться будет не компонент, а фильм предварительного просмотра.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Я не вижу своего компонента на панели Components. Почему?

На панели *Components* по умолчанию показан ряд компонентов пользовательского интерфейса Flash. Рядом с названием стоит значок открытия выпадающего меню. Щелкните на нем и **посмотрите**, появится ли файл, в котором вы сохранили прямоугольник. Если да, то выберите его, и тогда появятся компонент и выбранная для него пиктограмма. Если файл есть, а компонента нет, то, вероятно, вы забыли установить флажок в поле опции **Display in Components Panel** диалогового окна *Component Definition*. Если в раскрывшемся меню не указано имя файла, это значит, что вы не поместили файл в нужную папку.

Мне нужно отредактировать часть кода. Могу ли я вносить изменения прямо из каталога?

Если возникнет необходимость отредактировать код, то лучше всего удалить существующий файл из каталога *First Run* и области хранения данных, внести изменения в его отдельной копии, а затем вставить его обратно в каталог *First Run*.

ИСПОЛЬЗОВАНИЕ ОБУЧАЮЩИХ ВЗАИМОДЕЙСТВИЙ

В ЭТОЙ ГЛАВЕ...

Взаимосвязи и шаблоны	559
Использование перетаскивания	567
Заполнение бланка	569
Активные объекты	570
Активные области	570
Множественный выбор	571
Истина или ложь	571
Обратная связь, отслеживание знаний и навигация	572
Возможные проблемы	574
Flash за работой: использование системы LMS	574

ВЗАИМОСВЯЗИ и ШАБЛОНЫ

В настоящее время огромная область разработок посвящена созданию интерактивных обучающих программ. Зная об этом, компания Macromedia поставляет с пакетом Flash MX ряд встроенных обучающих взаимодействий, позволяющих любому пользователю создавать комплексные интерактивные обучающие программы в изолированном приложении или в приложении, которое пересылает данные на серверную систему управления обучением *LMS* (*Learning Management System*). Можно работать с отдельным обучающим взаимодействием или с тем, в котором система LMS используется для отслеживания совокупных результатов. Имейте в виду, что, прежде чем приступить к планированию и созданию обучающих проектов, нуждающихся в

отслеживании, необходимо приобрести серверную систему LMS, соответствующую стандарту **AICC** (Aviation Industry CBT Committee — Комитет профессиональной компьютерной подготовки в авиапромышленности) или **SCORM** (Shareable Content Object Reference Model — объектная эталонная модель совместно используемого содержимого).

Кроме того, ваша система должна соответствовать определенным ограничениям совместимости браузеров для отслеживания **обучающих** взаимодействий. Пользователи Windows должны иметь браузеры Internet Explorer 4.0 или Netscape 4.0 либо последующих версий. На компьютерах Macintosh в браузерах Internet Explorer отслеживание вообще невозможно, поэтому пользователи должны установить браузеры Netscape версии 4.5 или выше.

ИСПОЛЬЗОВАНИЕ ОБУЧАЮЩИХ ВЗАИМОДЕЙСТВИЙ

Обучающие взаимодействия Flash MX можно использовать одним из двух способов. Предоставляемые с программой шаблоны контрольных вопросников полностью доступны для редактирования с точки зрения разработки, но они имеют встроенный код **ActionScript**, предназначенный для отслеживания набранных очков и передачи их в систему LMS, совместимую со стандартом SCORM или AICC.

Кроме того, можно использовать изолированные взаимодействия, которые можно редактировать в соответствии со своими потребностями и дизайном. Такие взаимодействия можно редактировать с точки зрения графической перспективы.

Результаты, полученные при использовании изолированных взаимодействий, можно обработать и передать только в систему **AICC LMS**. Для использования отслеживания, совместимого со SCORM, необходимо использовать шаблон контрольного вопросника.

В этом месте вы можете запаниковать только от самой идеи использования шаблона. Дело в том, что в большинстве случаев пользователей смущает, что при использовании шаблона существует привязка к определенному стилю и макету, и все будут знать о том, что вы пользуетесь шаблоном. Об этом не стоит беспокоиться, поскольку вы будете иметь полный контроль над графическим содержимым шаблона, а также сможете полностью регулировать задаваемые вопросы. При этом в шаблоне имеется полнофункциональный вопросник со встроенной системой навигации, и разработку можно будет сразу же протестировать.

Во Flash MX имеются три обучающих шаблона: **quiz_style1**, **quiz_style2** и **quiz_style3**. Доступ к ним можно получить с помощью команды **File⇒New from Template** (**Файл⇒Создать** на основе шаблона). Содержимое всех шаблонов одинаково и состоит из следующих составных элементов:

- вводная страница;
- страница навигации;
- страница результатов;
- обучающее взаимодействие;
- код ActionScript для отслеживания результатов.

Процедура создания вопросника во Flash очень проста. Достаточно выполнить следующие действия.

1. Выполните команду **File⇒New from Template**, в результате чего откроется диалоговое окно New Document (Создать документ), показанное на рис. 23.1.
2. Из списка Category (Категория) выберите пункт Quiz (Вопросник). При этом в правой части диалогового окна будут отображены пункты трех шаблонов **quiz_style**.
3. Взгляните на три пункта шаблонов, расположенных в списке Category Items (Элементы категории). Перемещая курсор от одного шаблона к другому, в окне предварительного просмотра вы сможете увидеть стиль каждого из них (рис. 23.2).

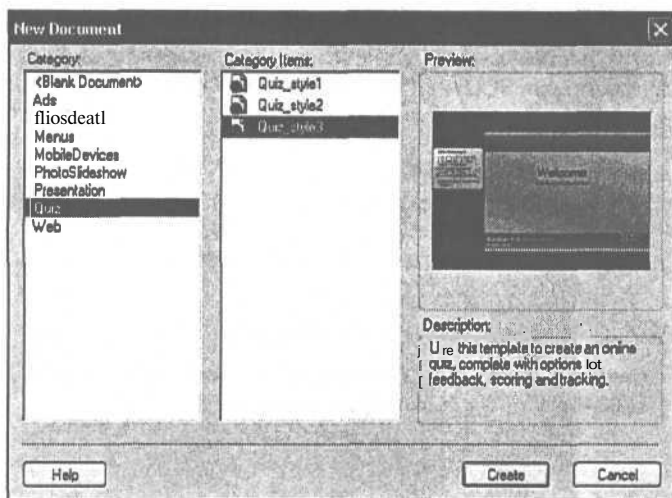


Рис. 23.1. В диалоговом окне *New Document* можно выбрать один из трех шаблонов *quiz_style*, с которым вы хотите работать

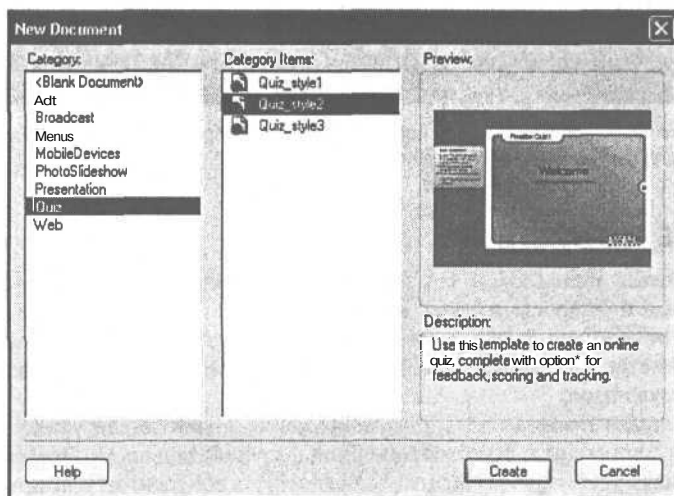


Рис. 23.2. Прежде чем выбрать шаблон вопросника, каждый из них можно предварительно просмотреть, щелкнув на пункте шаблона

4. Выберите шаблон, с которым хотите работать, а затем щелкните на кнопке *Create* (Создать). Flash создаст полный рабочий фильм со всеми необходимыми компонентами вопросника.
5. Сохраните файл.

На заметку

При первом сохранении файла Flash автоматически предложит сохранить его копию, поскольку файл был создан на основе шаблона.

После создания файла в рабочем поле появятся вопросник и взаимодействия. На рис. 23.3 показано, что все слои вопросника располагаются на временной шкале.



Рис. 23.3. После создания файла в рабочем поле появятся вопросник и взаимодействия. Все слои сразу же будут видны на временной шкале

ДОБАВЛЕНИЕ ПАРАМЕТРОВ

Вопросник нельзя использовать без добавления параметров, управляющих тем, как пользователи будут видеть вопросы, а также внешним видом и порядком отображения вопросов. Вопросы могут появляться в определенной последовательности или случайным образом. Кроме того, параметры управляют использованием системы LMS и тем, увидят ли пользователи страницу результатов.

При выборе опции **последовательного** отображения пользователи увидят вопросы, появляющиеся друг за другом в виде обычной линейной последовательности. Отображение вопросов в случайном порядке удобно, когда вы хотите представить вопросник пользователям несколько раз, но при этом изменить порядок вопросов. Если порядок отображения вопросов изменяется, то пользователям сложнее будет записать ответы и использовать их в дальнейшем.

Фильм содержит панель Instructions (Инструкции), расположенную в рабочем поле. Она используется для добавления и управления параметрами вопросника. Перед заданием параметров необходимо щелкнуть на панели Instructions и выбрать компонент Quiz (Вопросник).

Совет

Панель Instructions предоставляется только для вашего удобства — пользователи ее никогда не видят.

Для задания параметров вопросника необходимо заполнить панель Component Parameters (Параметры компонента). Сначала откройте ее из меню Window и по необходимости измените размеры.

Заполните панели в соответствии со своими потребностями. Приведенная ниже информация поможет составить оптимальный вопросник.

- Чтобы отображать вопросы в случайной последовательности, установите флажок в поле опции **Randomize** (Расположить в случайном порядке).
- В текстовом поле **Questions to Ask** (Задать вопросов) можно указать число вопросов, задаваемых пользователям.
 - При задании в этом поле значения 0 пользователи всегда будут видеть все вопросы.
 - Если вопросник содержит, к примеру, 12 вопросов, а в текстовом поле вы укажете меньшее число, то пользователи будут видеть только указанное количество вопросов.
 - Если в текстовом поле указано число, превышающее число имеющихся вопросов, пользователи увидят все вопросы, однако дважды они отображаться не будут.
- Если для использования вопросника пользователи должны **зарегистрироваться**, а вы не используете систему LMS, совместимую с AICC, то Flash перенаправит пользователей по указанному на этой панели адресу URL для регистрации.
- При использовании системы LMS введите название деятельности и ID системы LMS. Если вы не используете LMS, эти поля можно проигнорировать.
- Чтобы в конце вопросника пользователи могли увидеть свои результаты, установите флажок в поле опции **Show Results Page** (Показать страницу результатов).

Заполненная панель должна выглядеть приблизительно так, как показано на рис. 23.4.

ДОБАВЛЕНИЕ ВЗАИМОДЕЙСТВИЙ

Созданная на основе шаблонов временная шкала имеет восемь ключевых кадров. К ним относятся экран приветствия (ключевой кадр 1), экран результатов (ключевой кадр 8) и шесть ключевых кадров вопросника. Шаблон вопросника можно увидеть в любой момент, протестировав фильм.

Процедура помещения вопросов в вопросник очень проста. В любой момент вы добавляете ключевые кадры со взаимодействиями или удаляете существующие. Единственное прави-

The image shows a software interface window titled 'Component Parameters' with a sub-tab 'Quiz'. Inside the window, there are several configuration options for a quiz:

- Randomize**: A checkbox that is checked.
- Questions to Ask**: A text input field containing the number '6'.
- Login File URL**: A text input field containing the URL 'http://www.loncoley.co.uk/login'.
- Activity ID**: An empty text input field.
- Activity Name**: An empty text input field.
- Show Results Page**: A checkbox that is checked.

Рис. 23.4. Панель **Component Parameters** необходимо заполнить до составления вопросов

ло, которое следует запомнить, состоит в следующем: первый и последний ключевые кадры должны содержать соответственно экран приветствия и страницу результатов, вопросы (взаимодействия) необходимо размещать в слое Interactions (Взаимодействия) на временной шкале в нужном порядке.

Текст на странице приветствия и странице результатов можно редактировать в соответствии с собственным стилем, цветовой гаммой и т.п. Важно только помнить, что на этих страницах не могут содержаться обучающие взаимодействия и что при редактировании текста необходимо оставить какие-то инструкции, чтобы пользователи, например, знали, что для перехода к следующему вопросу им необходимо щелкнуть на кнопке **Next** (Далее). Динамические текстовые поля на странице **результатов** изменять не нужно, поскольку это приведет либо к неправильному отображению вопросов, либо вообще к невозможности их отображения.

Взаимодействия на других экранах можно корректировать или удалять, исходя из своих потребностей и дизайна узла. Начнем с самого простого.

УДАЛЕНИЕ ВЗАИМОДЕЙСТВИЯ

Процедура удаления взаимодействия очень проста, но нужно быть внимательным и удалить взаимодействие из всех необходимых слоев. В слое Interactions выберите ключевой кадр с ненужным взаимодействием. Затем можно выполнить одно из двух действий. Можно удалить содержимое, нажав клавишу <Delete> или <Backspace>, и тогда данный ключевой кадр можно будет использовать, добавив в него другое взаимодействие. Кроме того, можно полностью удалить кадр из временной шкалы, выполнив команду **Insert⇒Remove Frame** (Вставка⇒Удалить кадр). Обязательно проверьте, чтобы все слои на временной шкале оканчивались в одной точке.

КОНФИГУРИРОВАНИЕ ВЗАИМОДЕЙСТВИЯ

В шаблоне вопросника содержится по одному из шести типов обучающих взаимодействий. Они хранятся в видеоклипах, расположенных на панели Library (Библиотека). Каждый видеоклип выполняет роль контейнера элементов, составляющих взаимодействие. Перед редактированием взаимодействия в соответствии со своими потребностями его необходимо разбить на части.

Чтобы сконфигурировать взаимодействие, выполните следующее.

1. В слое Interactions щелкните на ключевом кадре, содержащем компонент, подлежащий конфигурированию.
2. Выберите обучающее взаимодействие, а затем выполните команду **Modify⇒Break Apart** (Изменение⇒Разбить на части). Команда Break Apart используется для разбивки групп, экземпляров и растров на отдельные элементы. В данном случае команда позволяет конфигурировать обучающее взаимодействие.
3. Снимите выделение всех элементов, содержащихся в рабочем поле. Для этого выполните команду **Edit⇒Deselect All** (Правка⇒Снять все выделение) либо воспользуйтесь комбинацией клавиш <Ctrl+Shift+A>.
4. Щелкните на панели Instructions обучающего взаимодействия, расположенной слева от рабочего поля.
5. Откройте панель Component Parameters, выполнив команду **Window⇒Component Parameters**. Заполните панель Component Parameters (укажите имя и ID взаимодействия, если вы используете систему LMS). На рис. 23.5 показана панель, заданная по умолчанию для взаимодействия True/False.
6. Введите вопрос или инструкцию, которую должны выполнить пользователи. Затем щелкните на кнопке Options, чтобы открыть одноименный экран, в котором можно указать параметры Feedback (Обратная связь), Knowledge Track (Отслеживание знаний) и Navigation (Навигация), как показано на рис. 23.6.
7. Воспользуйтесь кнопкой Assets (Элементы), чтобы изменить элементы обучающего взаимодействия.

После настройки первого взаимодействия сохраните и протестируйте файл.

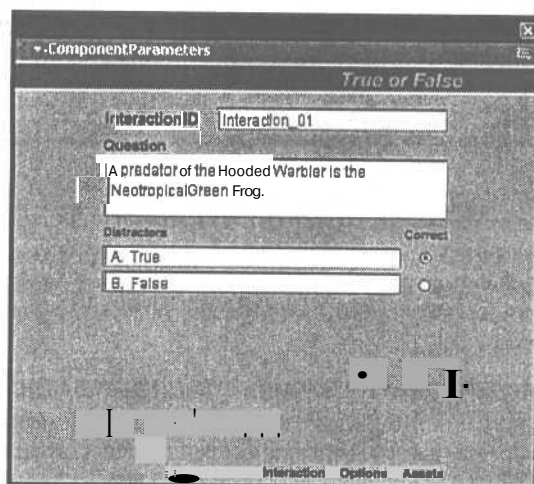


Рис. 23.5. Панель *Component Parameters* используется для добавления вопросов и указания возможных ответов

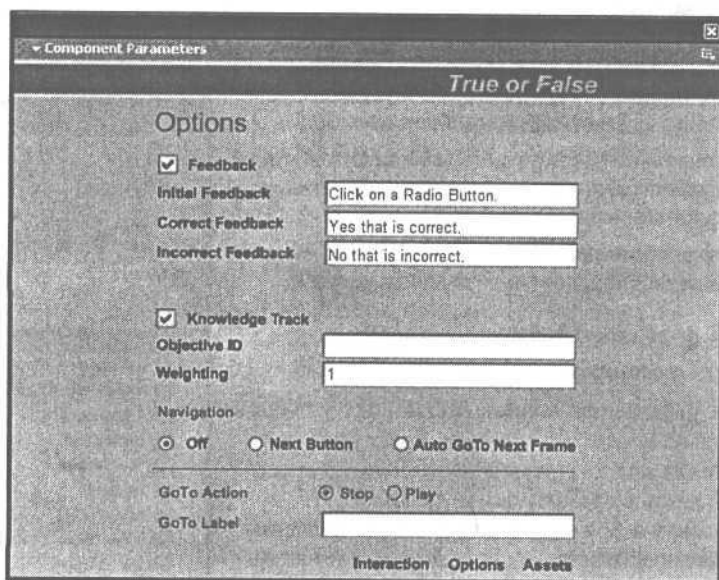


Рис. 23.6. Яд экране *Options* панели *Component Parameters* можно добавить и изменить обратную связь с пользователями, а также задать параметры отслеживания знаний и навигации

На заметку

В документах, созданных на основе шаблона вопросника, для каждого обучающего взаимодействия необходимо установить флажок в поле *Knowledge Track*, а переключатель группы *Navigation* — в положение *Off* (Выкл.).

Будьте осторожны при разбивке!

При разбивке обучающих взаимодействий соблюдайте осторожность. Разбить нужно элементы, а не фактический компонент. При разбивке компонента обучающего взаимодействия будет разбит код **ActionScript**, и вопросник перестанет корректно функционировать. Если это произошло, то лучше всего полностью удалить взаимодействие, вставить новое из библиотеки и начать все заново.

Шесть типов обучающих взаимодействий будут подробно описаны далее в этой главе.

ДОБАВЛЕНИЕ ВЗАИМОДЕЙСТВИЯ НА ВРЕМЕННУЮ ШКАЛУ

При использовании шаблона необходимо последовательно добавить взаимодействия на временную шкалу. При добавлении взаимодействий в уже созданный Flash-документ их можно добавить либо в один кадр временной шкалы, либо в последовательные кадры для последовательных вопросов.

ДОБАВЛЕНИЕ ВЗАИМОДЕЙСТВИЯ НА ВРЕМЕННУЮ ШКАЛУ ШАБЛОНА

Если вы хотите добавить взаимодействие на **временную** шкалу шаблона, выберите в слое Interactions **ключевой** кадр, в котором содержится вопрос, предшествующий тому, который вы хотите вставить. Воспользуйтесь меню **Insert** для добавления нового пустого ключевого кадра. Находясь в слое Interactions, выполните команду Insert Frame (Вставить кадр), а затем, выделив новый кадр, выполните команду Insert Blank Keyframe (Вставить пустой ключевой кадр), чтобы преобразовать новый кадр в ключевой. После этого выполните следующие действия.

Совет

Процедура конфигурирования взаимодействий была описана ранее в этой главе, в подразделе "Конфигурирование взаимодействия".

1. Откройте панель Library данного документа и выберите папку 2_Learning Interactions (рис. 23.7).
2. Выделите только что вставленный ключевой кадр и перетащите один из компонентов обучающего взаимодействия в рабочее поле.
3. Разместите компонент **там**, где он должен появиться, а затем сконфигурируйте взаимодействие.

ДОБАВЛЕНИЕ ВЗАИМОДЕЙСТВИЯ НА ВРЕМЕННУЮ ШКАЛУ СУЩЕСТВУЮЩЕГО ДОКУМЕНТА

Процедура добавления взаимодействия на временную шкалу существующего документа аналогична процедуре добавления взаимодействия на временную шкалу шаблона, описанной в предыдущем подразделе.

Откройте документ, в который вы хотите добавить обучающее взаимодействие, и, находясь в соответствующем слое, вставьте пустой ключевой кадр. Откройте панель библиотеки обучающих взаимодействий (рис. 23.8), выполнив команду **Window**⇒**Common Libraries**⇒**Learning Interactions** (**Окно**⇒**Общие библиотеки**⇒**Обучающие взаимодействия**). Затем выполните операции добавления и конфигурирования обучающего взаимодействия так, как это делалось для шаблона.

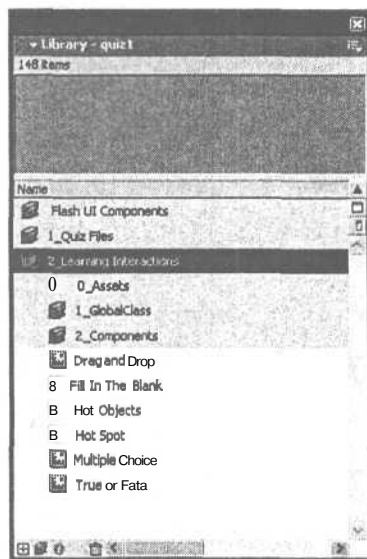


Рис. 23.7. В папке 2_Learning Interactions содержится обучающие взаимодействия, которые можно добавить в вопросник

ИСПОЛЬЗОВАНИЕ ПЕРЕТАСКИВАНИЯ

Из самого названия Drag and Drop (Перетаскивание) **следует**, что данное обучающее взаимодействие позволяет пользователям перетаскивать элементы по экрану, а затем отпускать их в каком-то месте. Эта возможность широко используется для реализации совпадения контуров или для создания "ажурных" взаимодействий, где результат является очень важным.

Обучающее взаимодействие Drag and Drop может поддерживать максимум восемь объектов и восемь целевых элементов. В целях тестирования объекты можно отпускать в любом месте экрана. Они могут иметь несколько целевых элементов, например объект 1 и объект 3 могут иметь один целевой элемент 7.

Целевые элементы не должны иметь совпадающих объектов. Таким образом, реализуется необходимый для тестирования выбор правильного ответа из нескольких предложенных.

КОНФИГУРИРОВАНИЕ ОБУЧАЮЩЕГО ВЗАИМОДЕЙСТВИЯ DRAG AND DROP

Как и при работе со всеми обучающими взаимодействиями, первое, что необходимо сделать после размещения взаимодействия в рабочем поле, — это разбить его на части и открыть панель Component Parameters. Далее следует сконфигурировать взаимодействие, выполнив следующие действия.

1. Внесите в столбец Drag Object Name (Имя объекта перетаскивания) имена экземпляров объектов перетаскивания.
2. Имейте в виду, что каждый объект **перетаскивания** должен иметь уникальное имя. При каждом добавлении объекта перетаскивания в обучающее взаимодействие необходимо возвращаться к панели и присваивать ему имя.
3. В столбце Matches Target Name (Имя совпадающего целевого элемента) перечислите имена экземпляров целевых элементов, как показано на рис. 23.9. Как и объекты перетаскивания, каждый целевой элемент должен иметь уникальное имя.
4. Если вы хотите, чтобы объекты вернулись в начальные положения в случае, если их перетаскивают не на целевые элементы, установите флажок в поле опции Snap to Start (Привязать к начальному положению).

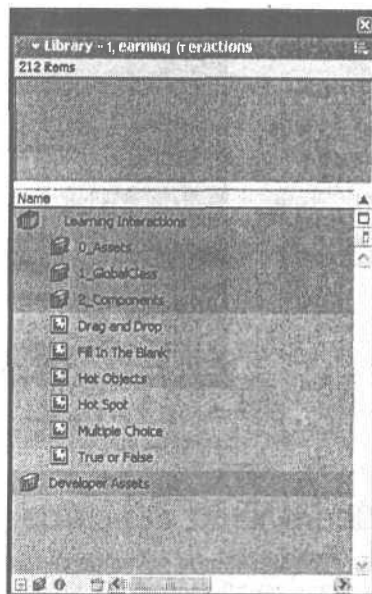


Рис. 23.8. На панели Library содержатся взаимодействия, которые можно добавить в любой документ, а не только в шаблоны

Соответствующие целевые элементы

Каждому объекту перетаскивания, указанному в столбце Drag Object Name, должен быть поставлен в соответствие экземпляр целевого элемента, имя которого указывается в столбце Matches Target Name.

Однако можно иметь экземпляр целевого элемента, которому не поставлен в соответствие экземпляр объекта перетаскивания. Таким образом, можно иметь целевые элементы, к которым пользователи смогут привязывать объекты, но которые не будут считаться правильными соответствиями.

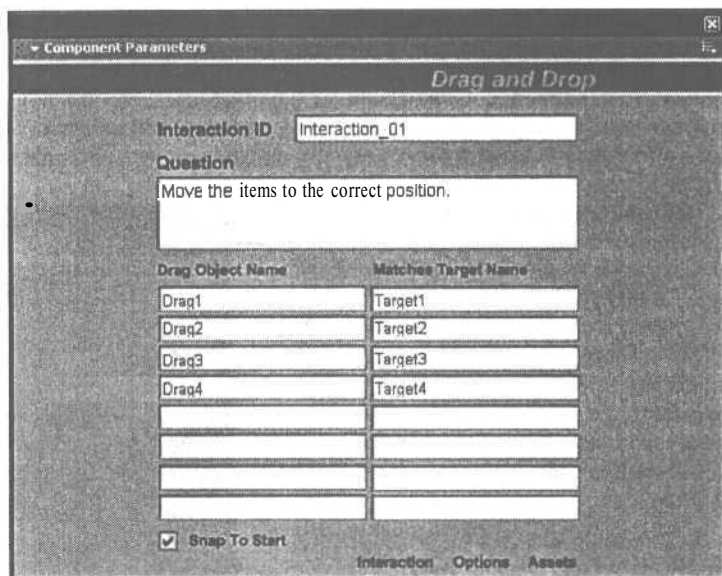


Рис. 23.9. Все экземпляры должны иметь уникальные имена. Это относится как к объектам перетаскивания, так и к целевым элементам

ДОБАВЛЕНИЕ ОБЪЕКТОВ ПЕРЕТАСКИВАНИЯ и ЦЕЛЕВЫХ ЭЛЕМЕНТОВ

По умолчанию обучающее взаимодействие Drag and Drop имеет шесть объектов и шесть соответствующих целевых элементов. Flash MX позволяет добавлять и удалять объекты и элементы, доводя их максимальное число до восьми, а минимальное — до одного.

В качестве упражнения создайте символ видеоклипа, содержащий графическое изображение объекта перетаскивания. Если вопросник связан с автомобилями и в данный момент во взаимодействии имеется пять автомобилей, то необходимо создать графическое изображение шестого, седьмого и восьмого автомобилей (столько, сколько требуется). Далее выполните следующие действия.

1. Поместите новое графическое изображение в библиотеку. Выбрав обучающее взаимодействие Drag and Drop, перетащите новый символ или символы с панели Library в рабочее поле.
2. На панели инспектора свойств задайте имя экземпляра каждого нового символа.
3. На панели Component Parameters добавьте имя экземпляра.
4. Повторите процесс присвоения имени для каждого нового добавляемого экземпляра.

Это все, что нужно сделать. Остальную работу компонент сделает самостоятельно в процессе исполнения. Необходимо только сохранить и протестировать файл, чтобы убедиться, что не возникает неожиданных проблем.

УДАЛЕНИЕ ОБЪЕКТОВ ПЕРЕТАСКИВАНИЯ

Экземпляр объекта перетаскивания можно удалить, выделив его в рабочем поле и применив команду удаления. Помните о том, что необходимо также удалить имя экземпляра удаленного объекта с панели Component Parameters. Если вы забудете сделать это и зарегистрируете какие-либо "отвлечения", фактически не находящиеся в рабочем поле, Flash не предупредит вас об этом до тех пор, пока фильм не будет протестирован. Только на этом этапе будет отображено сообщение об ошибке результата отслеживания.

ЗАПОЛНЕНИЕ БЛАНКА

Вероятно, вы уже пришли к заключению, что обучающие взаимодействия во Flash MX названы очень логично. Взаимодействие Fill in the Blank (Заполнение бланка) позволяет пользователям заполнить бланк, чего и следовало ожидать.

В обучающем взаимодействии Fill in the Blank имеется текстовое поле, в котором указан вопрос, задаваемый пользователям, текстовое поле, в которое пользователи могут ввести свой ответ, кнопка управления, позволяющая пользователям увидеть, правильно ли они ответили на вопрос, и текстовое поле обратной связи, в котором отображается информация, инициируемая после щелчка на кнопке управления.

КОНФИГУРИРОВАНИЕ ВЗАИМОДЕЙСТВИЯ FILL IN THE BLANK

Сконфигурировать взаимодействие Fill in the Blank очень легко. Сначала нужно открыть панель Component Parameters точно так же, как это делалось при работе с другими обучающими взаимодействиями. На панели Component Parameters может быть размещено до трех правильных ответов на поставленный вопрос (рис. 23.10).

Существуют такие альтернативы задания вопросов.

- Введите на панели правильный ответ (или ответы) и установите флажок в поле опции Correct (Правильный).
- Укажите, что правильными ответами являются не те, которые были введены. Для этого на панели введите неправильные ответы, снимите флажок с поля опции Correct и установите его в поле Other Responses (Другие ответы).

Укажите, чувствительны ли требуемые ответы к регистру символов. Для этого установите или снимите флажок с поля опции Case Sensitive (Чувствительный к регистру). Затем установите флажок в поле Exact Match (Точное совпадение), если пользователи должны вводить ответ точно в таком же виде, как его ввели вы. Если флажок в поле опции Exact Match не ус-

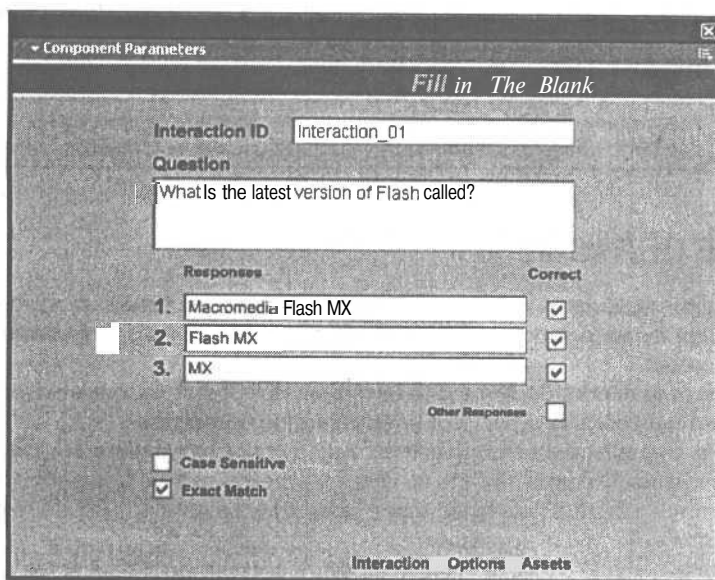


Рис. 23.10. Введите на панели максимум три приемлемых ответа либо укажите только ответы, являющиеся неправильными

тановлен, то взаимодействие будет распознавать ответ по правильно введенному слову. Если верным ответом на вопрос будет считаться просто **Flash**, а пользователь вводит **Flash MX**, то ответ будет считаться правильным.

АКТИВНЫЕ ОБЪЕКТЫ

Название **Hot Object** (Активный объект) для обучающего взаимодействия может показаться несколько странным. Это название менее очевидно, чем другие. Обучающее взаимодействие **Hot Object** позволяет вам задать вопрос, а пользователям — выбрать один или несколько правильных ответов из ряда графических изображений. Например, если тема вашего вопросника связана с автомобилями, то пользователям можно показать от одного до восьми изображений автомобилей и задать вопрос, на который может быть несколько правильных ответов. К примеру, можно спросить, какие автомобили изготовлены концерном **Ford** или какие модели уже сняты с производства.

КОНФИГУРИРОВАНИЕ ОБУЧАЮЩЕГО ВЗАИМОДЕЙСТВИЯ НОТ ОБЪЕКТ

Если вы хотите сконфигурировать обучающее взаимодействие **Hot Object**, начните с разбивки его на части и открытия панели **Component Parameters**. Заполнение панели для взаимодействия **Hot Object** теперь покажется очень простым. При выполнении этой задачи необходимо следовать той же логике, что и в предыдущих случаях. Начните с задания взаимодействия и его **ID**, затем введите вопрос, на который должны будут отвечать пользователи, и установите флажок в поле опции **Correct** напротив каждого правильного ответа.

ДОБАВЛЕНИЕ И УДАЛЕНИЕ ВАРИАНТОВ АКТИВНЫХ ОБЪЕКТОВ

Варианты активных объектов добавляют и удаляют точно так же, как это делалось с объектами перетаскивания (взаимодействие **Drag and Drop**). Необходимо создать дополнительные графические изображения в символах видеоклипов и перед размещением в рабочем поле добавить их на панель **Library**. После этого можете обратиться к инструкциям подраздела о взаимодействии **Drag and Drop**.

Совет

Инструкции по работе с объектами перетаскивания приведены выше, в подразделе "Конфигурирование обучающего взаимодействия **Drag and Drop**".

АКТИВНЫЕ ОБЛАСТИ

Обучающие взаимодействия **Hot Spot** (Активная область) заданы так, что для указания правильного ответа на вопрос пользователю необходимо щелкнуть в определенной области (или областях) экрана.

Единственное отличие **Hot Spot** от взаимодействия **Hot Object** заключается в том, что первое является некоторой областью, а второе — фактическим объектом.

Как и в случае с остальными взаимодействиями, для конфигурирования взаимодействия **Hot Spot** его сначала необходимо разбить на части и открыть панель **Component Parameters**. На панели пометьте каждый возможный ответ как правильный или неправильный, причем правильных ответов может быть несколько.

Опция **Up State Alpha** (Прозрачность в ненажатом состоянии) позволяет задать прозрачность области до того, как пользователь щелкнет на ней мышью. Полной прозрачности соответствует значение 0.

Кроме того, можно задать прозрачность области в нажатом состоянии (Down State Area). Данная опция управляет прозрачностью области после того, как пользователь щелкнет на ней мышью. Это поможет, к примеру, затенить выбранные ответы, подчеркивая, что на них уже щелкали.

Добавлять и удалять объекты Hot Spot можно точно так же, как и при работе с другими обучающими взаимодействиями.

МНОЖЕСТВЕННЫЙ ВЫБОР

Взаимодействия Multiple Choice (Множественный выбор) позволяют пользователям выбрать **правильный** ответ или ответы из вариантов, предложенных на экране. Можно указать, имеет ли каждый вопрос только один или несколько правильных ответов. К каждому вопросу имеется шесть возможных ответов (однако предлагать пользователям именно такое количество вариантов совсем необязательно). Для взаимодействия Multiple Choice они нумеруются не по порядку, а буквами от A до F (в английском алфавите).

Как всегда, для конфигурирования взаимодействия следует воспользоваться панелью Component Parameters. На панели введите возможные ответы, а затем вопрос, задаваемый пользователям. Пометьте соответствующие ответы как правильные или неправильные.

ДОБАВЛЕНИЕ ВАРИАНТОВ МНОЖЕСТВЕННОГО ВЫБОРА

По умолчанию предлагается шесть вариантов множественного выбора, о чем уже упоминалось в предыдущем подразделе. Однако число возможных вариантов можно увеличить до восьми или, наоборот, уменьшить, удалив ненужные.

Для добавления вариантов множественного выбора выполните следующие действия.

1. На временной шкале выберите кадр, содержащий взаимодействие Multiple Choice. Затем выполните команду **Window⇒Library** и откройте папку Flash UI Components (Компоненты пользовательского интерфейса Flash), расположенную на панели Library.
2. Перетащите компонент Check Box (Флаговое поле) в рабочее поле и на панели инспектора свойств присвойте имя экземпляру.
3. Откройте панель Component Parameters и добавьте имя только что созданного экземпляра.

ИСТИНА или ложь

Обучающее взаимодействие True or False (Истина или ложь) позволяет пользователям в качестве ответа на вопрос **выбирать** варианты типа "истина" или "ложь". Взаимодействие True or False состоит из текстового поля, в котором содержится задаваемый пользователям вопрос, двух компонентов Radio Buttons (Переключатели) для возможных ответов, кнопки управления и текстового поля обратной связи. Кнопка управления и текстовое поле обратной связи показывают пользователям, правильно ли они ответили на вопрос, и дают инструктирующий текст. Параметры обратной связи будут рассмотрены далее в этой главе.

КОНФИГУРИРОВАНИЕ ОБУЧАЮЩЕГО ВЗАИМОДЕЙСТВИЯ TRUE OR FALSE

Для конфигурирования обучающего взаимодействия True or False используется панель Component Parameters, на которой указываются ID взаимодействия и задаваемый вопрос. Помните, что для обозначения правильного ответа необходимо выбрать один из переключателей.

Текст взаимодействия True or False можно изменить на более соответствующий вашему узлу и стилю. Для этого необходимо отредактировать текст в поле Distractors (Варианты). Например, в качестве ответов можно использовать варианты "Да" и "Нет".

ОБРАТНАЯ СВЯЗЬ, ОТСЛЕЖИВАНИЕ ЗНАНИЙ И НАВИГАЦИЯ

До сих пор мы рассматривали различные типы обучающих взаимодействий, предлагаемые во Flash MX. Вы убедились, что их использование представляет собой обычную адаптацию функциональных возможностей для решения своих задач. Данная глава посвящена обучающим взаимодействиям, и в ней не рассматриваются специфические вопросы по настройке графических элементов. Я уверен, что на данной стадии знакомства с Flash вы уже достаточно знаете об этом, и повторяться не стоит. Однако к концу главы пришло время рассказать, какие части взаимодействий можно настраивать, а также рассмотреть те части, в которых содержатся сценарии. В данном разделе мы расскажем о том, что видят пользователи, рассмотрим отображаемые сообщения и встроенные средства навигации.

Совет

Подробная информация о графике во Flash представлена в главе 3 "Рисование во Flash".

Управление параметрами обратной связи, отслеживания знаний и навигацией осуществляется с уже знакомой панели Component Parameters (Параметры компонента). Доступ к этим параметрам для каждого взаимодействия можно получить после щелчка на кнопке Options (Параметры) панели Component Parameters.

ЗАДАНИЕ ПАРАМЕТРОВ ОБРАТНОЙ СВЯЗИ

Параметры обратной связи (Feedback) регулируют то, что увидят пользователи в виде отклика на данный ими ответ на поставленный вопрос. После каждого вопроса пользователи могут получать сообщение, говорящее им о правильности данного ответа. Можно также установить число попыток ответа на каждый из вопросов. Параметры обратной связи можно задать для каждого вопроса отдельно. Если установить флажок в поле опции Feedback, то пользователи будут получать сообщение после ответа на каждый поставленный вопрос. Кроме того, все взаимодействия, кроме множественного выбора, имеют параметр, позволяющий указать число возможных попыток ответа на вопрос.

Текст, указанный в поле Initial Feedback (Исходная обратная связь), пользователи увидят до того, как ответят на вопрос. Это удобный способ представления инструктирующего текста типа "выберите переключатель" или "щелкните для выбора правильного ответа".

В текстовые поля Correct Feedback (Обратная связь при правильном ответе) и Incorrect Feedback (Обратная связь при неправильном ответе) вводится сообщение, которое пользователи увидят в том случае, если соответственно правильно или неправильно ответят на поставленный вопрос. Чтобы воодушевить пользователей, используйте обе эти опции. Даже если был дан неправильный ответ, то ободряющие слова сообщения воодушевят пользователя на продолжение теста и помогут ему не отчаиваться. Когда вы учитесь чему-то новому, нет ничего хуже, когда вам просто указывают на неправильный ответ. Сообщение, введенное в поле Additional Tries (Дополнительные попытки), будет отображаться при каждой следующей попытке ответа на вопрос.

ЗАДАНИЕ ПАРАМЕТРА ОТСЛЕЖИВАНИЯ ЗНАНИЙ

Параметр отслеживания знаний Knowledge Track работает как в AICC-, так и в SCORM-совместимых системах LMS. Данный параметр представляет собой автоматическую функцию отслеживания данных, позволяющую передать данные об успеваемости обучающегося в LMS или другую систему отслеживания.

Чтобы иметь возможность использовать параметр Knowledge Track, обучающее взаимодействие необходимо внедрить на HTML-страницу, содержащую требуемый код JavaScript. Эта страница генерируется на основе настроек, заданных для публикации.

Совет

Подробная информация о публикации и оптимизации фильмов представлена в главе 30 "Оптимизация, публикация и экспортирование фильмов".

Все данные, охватываемые и отслеживаемые посредством параметра Knowledge Track, основываются на стандарте AICC.

На панели Component Parameters для выбранного взаимодействия можно установить значения элементов данных. Для задания параметров Knowledge Track выполните следующие действия.

1. В рабочем поле выберите панель Instructions, а затем откройте панель Component Parameters.
2. Щелкните на кнопке Options и установите флажок в поле опции Knowledge Track.
3. В поле Objective ID (Идентификатор цели) укажите цель взаимодействия. Этот идентификатор связан с целью, устанавливаемой в пределах системы LMS. Если поле опции оставить незаполненным, отслеживание все равно будет выполняться.
4. Укажите весовой коэффициент взаимодействия. Теперь все опции заполнены (рис. 23.11).

На заметку

Весовой коэффициент обозначает важность вопроса. Если все вопросы имеют одинаковую сложность, установите все значения весовых коэффициентов, равными 1; в противном случае задайте значения, исходя из того, что 1 означает самый легкий вопрос, а 3 — самый сложный.

ОСНОВНАЯ СТРУКТУРА СЦЕНАРИЕВ и КОМПОНЕНТОВ ОБУЧАЮЩИХ ВЗАИМОДЕЙСТВИЙ

Обучающие взаимодействия являются основой всего интерактивного процесса. Взаимодействия сами собирают параметры пользователей и создают массив SessionArray и функции обработчиков событий взаимодействий.

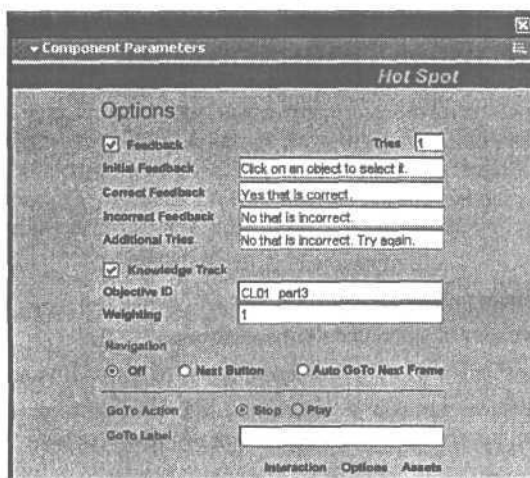


Рис. 23.11. На панели Component Parameters указаны все данные, необходимые для отслеживания информации и определения ее весового коэффициента

Чтобы глубже понять принципы работы этих функций, лучше всего рассмотреть сценарии панели Library.

Глобальный класс **Lttoolbox** обрабатывает хранимые данные и данные форматирования взаимодействий. Каждый компонент взаимодействия также имеет собственный сценарий инициализации функций обработчиков событий, которые активизируются посредством элементов взаимодействий.

Если вы хотите проанализировать или отредактировать сценарий, воспользуйтесь для его открытия панелью Library. Во всех сценариях имеются соответствующие комментарии Macromedia, что облегчает процесс анализа и редактирования кода. Большинство участков сценариев встроено в функции.

Сценарий, открытый на панели Library, можно редактировать так же, как и любой код ActionScript.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Я добавил новое взаимодействие, но теперь все работает неправильно. Какую проверку нужно выполнить ?

Проверьте, чтобы новый кадр, добавленный для взаимодействия, был ключевым. Добавление обычного кадра приводит к возникновению проблем.

Помогите! Я разбил компонент обучающего взаимодействия на части. Что теперь делать ?

Если разбить компонент обучающего взаимодействия на части, то встроенные сценарии взаимодействия будут потеряны или повреждены. В этом случае следует полностью удалить взаимодействие и заменить его новой копией из библиотеки.

Почему некоторые параметры для разных взаимодействий отличаются ?

Несмотря на то что большинство взаимодействий выполняется одинаково, вполне естественно, что они немного отличаются друг от друга. Например, для взаимодействия True and False нельзя разрешить делать несколько попыток ответов.

FLASH ЗА РАБОТОЙ: ИСПОЛЬЗОВАНИЕ СИСТЕМЫ LMS

Если вы почти или вообще ничего не знаете о том, что представляет собой система управления обучением LMS и как она функционирует, то приведенная ниже информация поможет разобраться с тем, что происходит при работе с интерактивным обучающим взаимодействием.

При работе с вопросником, совместимым с LMS, происходят указанные ниже события. Некоторые из них видны обучающемуся, а другие происходят "за кулисами".

1. Система LMS запускается, и пользователь регистрируется.
2. Пользователь выбирает обучающее действие.
3. Пользователь запускает обучающее взаимодействие.
4. Пользователь прорабатывает вопросы контрольного вопросника.
5. Flash-файл отправляет данные в систему LMS посредством файлов отслеживания HTML и JavaScript; пользователь не видит, как это происходит.

Независимо от типа LMS, пользователь приобретает фактически одни и те же навыки.

В зависимости от того, пересылаются ли данные вопросника в систему отслеживания AICC или SCORM, подготовительные мероприятия могут слегка отличаться друг от друга.

ПОДГОТОВКА ДАННЫХ ПРИ РАБОТЕ С СИСТЕМОЙ, СОВМЕСТИМОЙ С AICC

Отправление отслеженных данных в систему LMS, совместимую с AICC, требует возможности отслеживания вопросника и последующей публикации фильма с помощью шаблона отслеживания Flash W/AICC. Создаваемые во Flash HTML- и SWF-файлы необходимо поместить в один и тот же каталог на Web-сервере. Кроме того, необходимо отредактировать файл набора фреймов, созданный для того, чтобы указать имя вопросника.

Для подготовки к работе с системой, совместимой с AICC, выполните следующие действия.

1. Во Flash откройте фильм, содержащий вопросник. Выполните команду **File⇒Publish Settings** (**Файл⇒Параметры публикации**), в результате чего откроется одноименное диалоговое окно. Откройте вкладку **Formats** (**Форматы**) и убедитесь, что установлены флажки в полях **Flash** и **HTML**.
2. Откройте вкладку **HTML** и из раскрывающегося меню **Template** (**Шаблон**) выберите пункт **Flash w/AICC Tracking**, как показано на рис. 23.12.
3. Щелкните на кнопке **Publish** (**Публиковать**), а затем закройте диалоговое окно, щелкнув на кнопке **OK**. Flash создаст SWF- и HTML-файлы. Поместите оба этих файла в один и тот же каталог на Web-сервере.
4. С помощью приложения управления файлами выясните местоположение папки программы **Flash MX** и подпапки **First Run\HTML\Learning Extensions**.
5. Откройте последнюю папку и скопируйте два HTML-файла и подпапку **Scripts** в тот же серверный каталог, что был указан в шаге 3.
6. С помощью текстового редактора или приложения **Dreamweaver** откройте HTML-файл набора фреймов, скопированный на Web-сервер.
7. Измените строку с кодом `<frame src="Untitled-1.htm" name="content" frame border="0">` так, чтобы файл `Untitled-1.htm` ссылался на HTML-файл, созданный при публикации во Flash.
8. Запустите систему LMS, обращающуюся к странице `frameset.htm`.

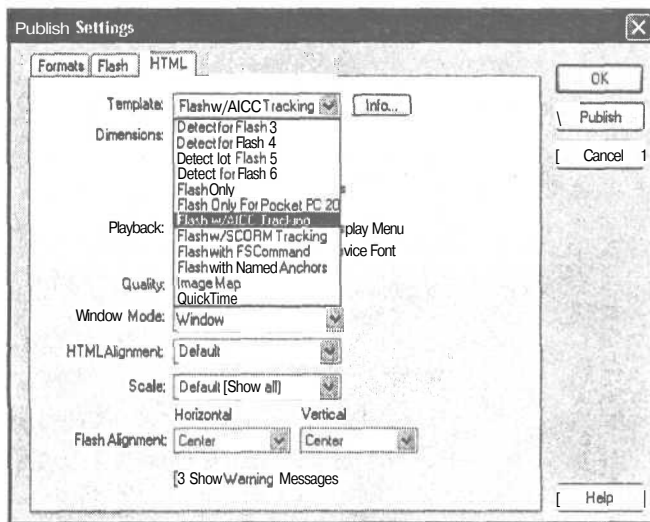


Рис. 23.12. Чтобы убедиться в том, что создаются правильные файлы, во вкладке **HTML** из раскрывающегося меню **Template** выберите пункт **Flash w/AICC Tracking**

ПОДГОТОВКА ДАННЫХ ПРИ РАБОТЕ С СИСТЕМОЙ, СОВМЕСТИМОЙ СО SCORM

Основное отличие между подготовкой к работе с системой, совместимой с AICC или со SCORM, заключается в том, какой шаблон используется во время публикации. В данном случае не требуется редактирование внутренних файлов. Для подготовки к работе с системой, совместимой со **SCORM**, выполните следующие действия.

1. Во Flash откройте фильм, содержащий вопросник. Выполните команду **File⇒Publish Settings (Файл⇒Параметры публикации)**, в результате чего откроется одноименное диалоговое окно.
2. Откройте вкладку **Formats (Форматы)** и убедитесь, что установлены флажки в полях **Flash** и **HTML**.
3. Откройте вкладку **HTML** и из раскрывающегося меню **Template (Шаблон)** выберите пункт **Flash w/SCORM Tracking**.
4. Щелкните на кнопке **Publish (Публиковать)**, а затем закройте диалоговое окно, щелкнув на кнопке **OK**. Flash создаст **SWF**- и **HTML**-файлы. Поместите оба этих файла в один и тот же каталог на Web-сервере.
5. Запустите систему **LMS** и добавьте ссылки, требующиеся для **HTML**-файла, созданного во Flash.

Чтобы файл работал корректно, необходимо убедиться в том, что система **LMS** запустила набор фреймов отслеживания **SCORM**.

КОЛЛЕКТИВНОЕ ИСПОЛЬЗОВАНИЕ ACTIONSCRIPT

В ЭТОЙ ГЛАВЕ...

Библиотеки коллективного использования	577
Удаленный доступ во Flash	580
Включение внешнего кода <code>ActionScript</code>	583
Возможные проблемы	584
Flash за работой: функция <code>LocalConnection</code>	585

БИБЛИОТЕКИ КОЛЛЕКТИВНОГО ИСПОЛЬЗОВАНИЯ

Библиотеки коллективного использования позволяют использовать видеоклипы, графику, звуки и кнопки в пределах нескольких фильмов. В библиотеке коллективного использования могут содержаться различные кнопки и звуки, последовательно использующиеся во многих файлах фильмов. Звук или графику можно задать так, чтобы они добавлялись ко всем клиентским узлам или видеоклипам, всегда использующимся в работе.

При работе с крупными узлами, содержащими несколько фильмов, преимущество использования библиотек коллективного использования заключается в возможности уменьшения времени загрузки.

После загрузки фильма, содержащего элементы коллективного использования, эти элементы не нужно загружать снова. В последующих файлах содержатся только ссылки на них, и при этом не используется мультимедиа-информация, включенная в фильм несколько раз.

Основное преимущество работы с элементами библиотек коллективного использования заключается в том, что они помогают сэкономить время, затрачиваемое на работу над узлом или управление проектом.

Библиотеки коллективного использования полезны каждому пользователю, работающему во Flash. Область их применения простирается от коллективного использования логотипа, отображаемого на всех узлах, до создания большой центральной библиотеки, позволяющей выполнять обновление узлов из одного места.

Элементы коллективных библиотек позволяют по вашему усмотрению повторно использовать элементы исходного фильма во многих других целевых фильмах. Этой цели можно достичь (как и в большинстве случаев при работе во Flash MX) двумя способами: либо с помощью сеансовых коллективных элементов, либо коллективных элементов, созданных на этапе разработки.

СЕАНСОВЫЕ ЭЛЕМЕНТЫ КОЛЛЕКТИВНОГО ИСПОЛЬЗОВАНИЯ

При работе с сеансовыми элементами коллективного использования их в качестве символов помещают в библиотеку одного фильма. Этот фильм называют *исходным*. Затем из целевого фильма устанавливается связь с элементами коллективного использования исходного. В результате элементы будут загружаться при воспроизведении фильма в браузере пользователя. Чтобы эта методика выполнялась, исходный фильм необходимо выгрузить или опубликовать в определенном каталоге. Если исходный фильм не выгружен, то элементы коллективного использования найдены не будут и, следовательно, не будут загружены при воспроизведении фильма.

Чтобы успешно воспользоваться сеансовыми коллективными элементами, необходимо убедиться в следующей последовательности операций. Авторы фильма, содержащего элементы коллективного использования, должны определить коллективный элемент в своем фильме. Для этого элемента необходимо задать строку и URL, указывающий местоположение файла.

Для определения коллективных **свойств** элемента в исходном файле используется диалоговое окно Symbol Properties (Свойства символа).

Процедура задания исходного фильма заключается в выполнении следующих действий.

1. Откройте или создайте фильм, который будет содержать элементы коллективного использования. При работе с новым фильмом создайте или добавьте символы, которые подлежат коллективному использованию.
2. Выберите символ, подлежащий коллективному использованию, а затем откройте панель Library (Библиотека), выполнив команду **Window ⇒ Library (Окно ⇒ Библиотека)**.
3. Щелкните на символе правой кнопкой мыши и из открывшегося меню выберите пункт Properties (Свойства). В результате откроется диалоговое окно Symbol Properties (Свойства символа), показанное на рис. 24.1. Если внешний вид диалогового окна отличается от изображенного на рисунке, щелкните для его расширения на кнопке Advanced (Дополнительно).
4. Чтобы сделать элемент доступным для связывания с целевым фильмом, установите флажок в поле опции Export for runtime sharing (Экспортировать для сеансового коллективного использования).
5. В поле Identifier (Идентификатор) введите имя элемента. Flash будет использовать это имя для идентификации элемента при установлении связи с целевым фильмом.
6. Введите адрес URL, в котором будет записан SWF-файл, содержащий элемент коллективного использования, и щелкните на кнопке OK.

На заметку

Чтобы элементы коллективного использования были доступны в целевых фильмах, необходимо поместить SWF-файл в URL, указанный в п. 6.

Теперь, когда данные коллективного использования исходного фильма расположены в нужном месте, очень важно, чтобы точно такие же данные были включены в целевой фильм. Если данные будут хоть немного отличаться, то элемент не будет корректно найден или будет недоступным для коллективного использования.

Для добавления данных снова используется диалоговое окно Symbol Properties. Для успешного установления связи выполните следующие действия.

1. В целевом фильме откройте панель Library.
2. На панели Library выберите видеоклип, кнопку или графический символ, а затем из меню параметров панели (кнопка в правом верхнем углу) выберите пункт Properties. Если в открывшемся диалоговом окне Symbol Properties опции дополнительных свойств не видны, щелкните на кнопке Advanced.
3. В группе опций Linkage (Связь) установите флажок в поле Import for runtime sharing (Импортировать для сеансового коллективного использования) для установления связи с элементом исходного фильма.

4. Введите тот же самый идентификатор, что и в исходном фильме. Идентификаторы должны быть совершенно одинаковыми.

5. Введите адрес URL, в котором будет записан SWF-файл, содержащий элемент коллективного использования. Ранее уже говорилось о том, что эта информация должна быть точно такой же, как и в исходном фильме.

6. Для автоматического обновления элемента при обнаружении новой версии по указанному местонахождению исходного фильма установите флажок в поле опции Always update before publishing (Всегда обновлять перед публикацией) и щелкните на кнопке OK.

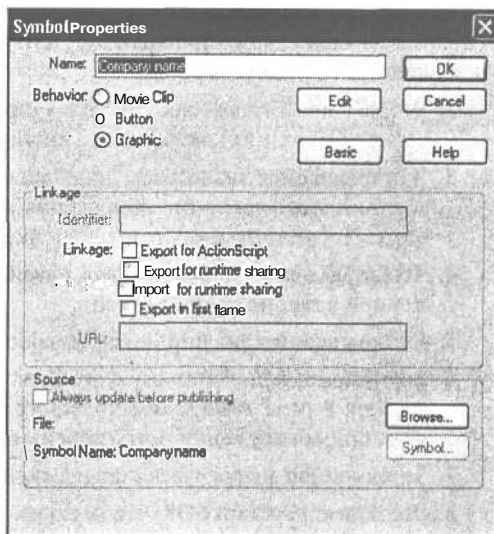


Рис. 24.1. В диалоговом окне *Symbol Properties* можно задать опции сеансового коллективного использования

На заметку

Чтобы отключить связь в целевом фильме, на панели Library выберите связанный символ, откройте диалоговое окно Symbol Properties и снимите флажок с поля опции Import for runtime sharing.

ЭЛЕМЕНТЫ КОЛЛЕКТИВНОГО ИСПОЛЬЗОВАНИЯ, СОЗДАННЫЕ НА ЭТАПЕ РАЗРАБОТКИ

Элементы коллективного использования, созданные на этапе разработки, используются для замены (или обновления) символов в новом фильме на символы из коллективного источника. Символы можно обновлять в любой момент во время разработки. При использовании коллективных символов, созданных на этапе разработки, содержимое символа заменяется (или обновляется) на содержимое символа, взятого из коллективного источника. В этом случае заданные имя и свойства символа сохраняются, а изменяется только его содержимое. При коллективном использовании элементов, созданных на этапе разработки, любые элементы, использующиеся в коллективном символе, также импортируются в целевой фильм.

Эта возможность удобна в случае необходимости использования обновленного изображения логотипа или обновления связи в пределах видеоклипа или компонента. Вместо того чтобы искать каждый экземпляр, можно просто обновить содержимое одного символа на содержимое элемента коллективного использования.

Из приведенных ниже инструкций видно, что процедура обновления или замены символов очень проста.

1. Откройте целевой фильм, выделите видеоклип, кнопку или графический символ, а затем из меню параметров панели Library выберите пункт Properties. Откроется диалоговое окно Symbol Properties.
2. Чтобы в диалоговом окне Symbol Properties выбрать новый файл FLA, в группе опций Source (Источник) щелкните на кнопке Browse (Обзор).
3. В открывшемся диалоговом окне найдите файл FLA, содержащий символ, который вы хотите использовать для обновления или замены символа, выделенного на панели Library. После этого щелкните на кнопке Open (Открыть).
4. Чтобы выделить новый символ в файле FLA, щелкните на кнопке Symbol, расположенной в группе опций Source.
5. В открывшемся диалоговом окне найдите нужный символ и щелкните на кнопке Open.
6. Вернувшись к диалоговому окну Symbol Properties, в группе опций Source установите флажок в поле Always update before publishing, чтобы автоматически обновить элемент при обнаружении новой версии по указанному местонахождению источника.
7. Заполненное диалоговое окно должно выглядеть так, как показано на рис. 24.2.
8. Щелкните на кнопке ОК, чтобы закрыть диалоговое окно Symbol Properties.

УДАЛЕННЫЙ ДОСТУП во FLASH

Удаленный доступ во Flash — это еще одна новая функция Flash MX, позволяющая соединяться с удаленными службами, такими как страницы ColdFusion. Эта функция позволяет разработчикам поддерживать связь с удаленными серверными объектами.

В части V рассказывается об использовании динамических данных для соединения с сервером. Эта функция позволяет хранить в базе данных информацию, используемую при создании покупательских корзинок, досок объявлений и т.п.

Функция удаленного доступа Flash поддерживает объектно-ориентированный доступ (например, объекты Java), а также обработку документов для обмена данными с удаленными службами, такими как JavaBean и Java-классы.

Чтобы воспользоваться функцией удаленного доступа Flash, необходимо установить программы Flash Remoting Components и Jrun4 и убедиться в том, что в системе установлены приложения Flash MX и Rash 6 Player.

Из использования функции удаленного доступа Rash можно извлечь много полезных моментов. Описание некоторых из них приведено в табл. 24.1, а полный перечень имеется на Web-узле Macromedia по адресу <http://www.macromedia.com>



Рис. 24.2. В заполненном диалоговом окне Symbol Properties должны быть указаны имя исходного файла FLA и символ

ТАБЛИЦА 24.1. ПРЕИМУЩЕСТВА УДАЛЕННЫХ СЛУЖБ FLASH

ФУНКЦИЯ	ПРЕИМУЩЕСТВО
Наличие одного прикладного программного интерфейса для вызова удаленных служб и XML-документов из приложений Macromedia Flash	Упрощает процесс разработки, требующийся для доступа к удаленным службам из Flash-приложений. Во Flash-приложениях отображает прикладные программные интерфейсы удаленных служб в виде простых прикладных программных интерфейсов ActionScript, делая доступной взаимосвязь разработчиков пользовательского интерфейса и разработчиков серверных приложений
Наличие новых прикладных программных интерфейсов ActionScript для обработки набора записей и связывания данных	Упрощает использование наборов записей в ActionScript, позволяя использовать Flash-приложения в качестве пользовательских интерфейсов стандартных реляционных баз данных
Поддержка широкого диапазона удаленных служб, включая EJB, Java-классы, JavaBeans и MBeans	Позволяет использовать интерфейс Flash в качестве пользовательского интерфейса приложений бизнес-логики
Реализация высокой скорости передачи данных между Flash и удаленными объектами	Увеличение производительности приложения путем минимизации количества данных, необходимых для передачи

JAVA-КЛАССЫ и УДАЛЕННЫЙ ДОСТУП во FLASH

Чтобы файлы Java-приложения стали доступными для функции удаленного доступа Flash, необходимо либо добавить новые каталоги в приложение JMV Classpath на сервере, использующем JMC, либо использовать каталог **Server-Inf**. При использовании каталога **Server-Inf** для Java-классов (в том числе JavaBeans) в нем должны содержаться классы, для которых указан путь `jrun_root/servers/jrun_server/SERVER-INF`.

Использование правильной структуры означает, что классы будут доступны на всем сервере.

При создании JavaBean необходимо внедрить **Serializable**, а при создании Java-объекта этого делать не нужно.

Если вы хотите разрешить пользователям соединяться непосредственно с Java-классами, необходимо задать требуемые разрешения для доступа к пакету.

Предполагая, что имеется пакет `com.myfiles` и что вы хотите разрешить доступ к классам, содержащимся в этом пакете, необходимо отредактировать файл `lib/jrun.policy`.

Приведенный ниже код делает этот файл доступным для любого пользователя, поэтому перед внесением подобных изменений будьте осторожны с его содержимым. Поскольку разрешения можно задать только для уровня пакета с Java, необходимо задавать разрешения для любого пакета, который вы хотите сделать общедоступным.

```
// PERMISSIONS GRANTED TO EVERYONE
```

```
grant {  
    // users should narrow or expand on this as they see fit  
    // to grant wide-open security access to all code, uncomment this line  
    // permission java.security.AllPermission;  
    // to grant clients access to classes in the jrun packages, uncomment  
    // this line
```



```
//permission java.lang.RuntimePermission
"accessClassInPackage.jrun";
// добавить эту строку
permission java.lang.RuntimePermission
"accessClassInPackage.com.myfiles";
//snipped
}
```

КЛАССЫ ACTIONSCRIPT и УДАЛЕННЫЙ ДОСТУП во FLASH

Классы **ActionScript** для удаленного доступа во Flash должны быть включены в фильм на уровне кадра. Реализацию основной функциональности обеспечивает класс **NetServices.as**, включенный в загрузку Flash Remoting Components.

При удаленном доступе во Flash используются два других класса **ActionScript**. К ним относятся **NetDebug.as**, который используется исключительно в среде разработки и, как следует из его имени, является файлом отладки, а также **DataGlue.as**, который помогает форматировать результаты, использующиеся в объекте набора данных. Приведенные ниже три строки кода реализуют включение требуемых файлов **ActionScript**.

```
#include "NetServices.as"
#include "NetDebug.as"
#include "DataGlue.as"
```

ВЫПОЛНЕНИЕ СЛУЖЕБНЫХ ОБРАЩЕНИЙ к СЕРВЕРУ из FLASH-ФИЛЬМА

Перед выполнением служебного обращения к серверу приложения сначала необходимо создать в фильме служебный объект. Эта задача реализуется посредством функций **createGatewayConnection** и **getService** класса **NetServices**, предназначенных для создания объекта соединения.

Следующий код демонстрирует создание служебного объекта с именем **FirstRemote**:

```
#include "NetServices.as"

if (inited == null)
{
    // выполнить этот код только один раз
    inited = true;

    // задать URL перехода по умолчанию (используется только при авторской
    // разработке)
    NetServices.setDefaultGatewayUrl("http://localhost:8100/flashservices/
gateway");

    // соединиться со шлюзом
    gateway_conn = NetServices.createGatewayConnection();

    // получить обращение к службе
    FirstRemote = gateway_conn.getService("FirstRemote", this);
}
```

После создания объекта соединения можно выполнять служебные обращения к серверу приложения, как показано в приведенном ниже примере.

```
FirstRemote.remoteServiceMethodName(param1, "param2");
```

Обратите внимание на то, что URL представлен не в обычном формате ссылки. В данном случае URL указывает местонахождение и порт Web-сервера, а также отображение шлюзового сервлета для удаленного Flash-сервера.

Для обработки результатов исполнения служебной функции можно либо создать специальный обработчик результатов, либо воспользоваться групповым обработчиком.

Совет

Подробная информация об обработчиках событий представлена в главе 12 "Управление переменными, данными и типами данных".

ВКЛЮЧЕНИЕ ВНЕШНЕГО КОДА ACTIONSCRIPT

Несмотря на то что ввод сценариев прямо на панели Actions (Действия) фильма считается совершенно обычной процедурой, сценарий можно сохранить в виде внешнего файла и в случае необходимости использовать повторно.

Для создания и редактирования кода **ActionScript** воспользуйтесь обычным текстовым редактором, а затем вставьте код на панель Actions либо воспользуйтесь действием Include (Включить), чтобы добавить внешний сценарий при прогоне фильма.

Кроме того, можно создать код ActionScript обычным способом (с помощью панели Actions), а затем сохранить его в виде текстового файла для последующего использования или редактирования.

Совет

Основы создания сценариев ActionScript изложены в главе 11 "Знакомство с ActionScript".

Код ActionScript можно экспортировать в текстовый файл в два этапа. На панели Actions откройте выпадающее меню, как показано на рис. 24.3, и выберите из него пункт Export as File (Экспортировать как файл). Укажите место, в котором вы хотите хранить сохраненный файл, и щелкните на кнопке Save (Сохранить).

После сохранения файла для внесения изменений в ActionScript можно использовать любой текстовый редактор.

После сохранения текстового файла его можно повторно импортировать в тот же самый файл FLA или в какой-то другой. Это дает превосходную возможность предоставления кода ActionScript в коллективное использование. Убедитесь в том, что вы работаете в режиме **Expert**, а затем из того же выпадающего меню выберите пункт Import From File (Импортировать из файла), найдите нужный текстовый файл и щелкните на кнопке Open.

Вместо того чтобы вставлять код ActionScript на панель Actions, можно с помощью действия Include включить текстовый файл при экспортировании фильма.

Убедитесь, что в окошке сценария точка вставки кода расположена в нужном месте (там, где вы хотите вставить сценарий). Дважды щелкните на действии Include в категории Miscellaneous Actions (Разные действия) и в поле Path (Путь) введите путь к внешнему файлу.

На заметку

Вводимый путь должен быть указан относительно файла FLA. При экспортировании фильма действие Include будет заменено на содержимое внешнего файла.

Несмотря на то что данная глава посвящена в основном коллективному использованию, здесь уместно также поговорить о внешних файлах в целом. Используя Flash MX, можно загружать и отправлять данные во внешние источники. К таким источникам относятся внешние сценарии, расположенные на сервере, а также текстовые и, конечно же, XML-файлы.

Фильмы можно настраивать так, чтобы изображения и звуковые файлы загружались при их воспроизведении. Эта возможность позволяет изменять и редактировать внешний файл без необходимости изменения исходного SWF-файла. Если только не будет поврежден код при внесении изменений, фильм будет воспроизводиться превосходно.

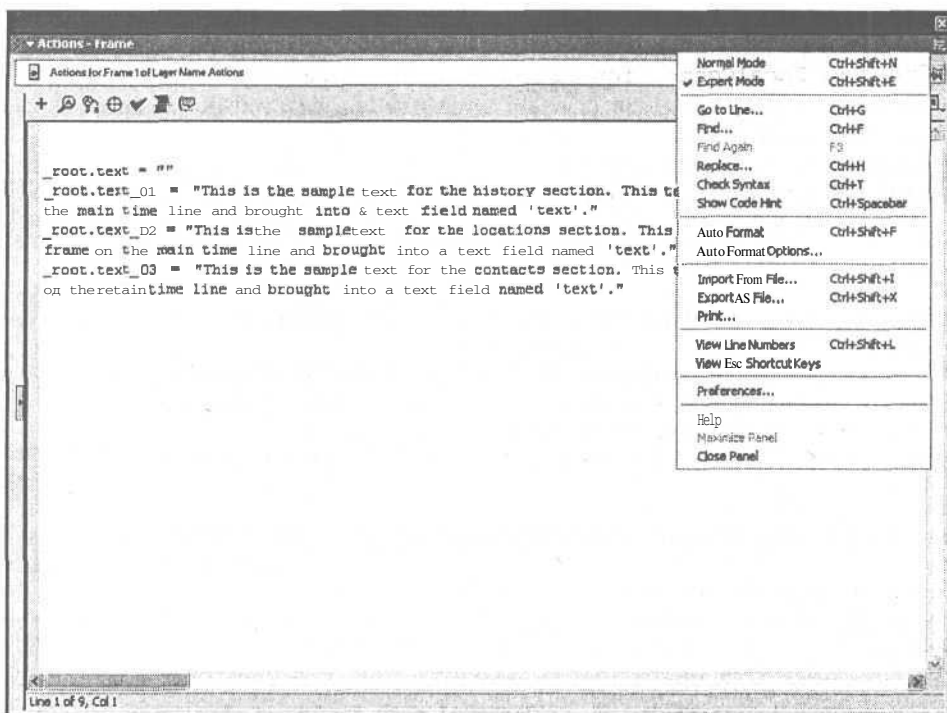


Рис. 24.3. В выпадающем меню панели Actions имеется много полезных функций, в том числе и Export as File

Расширить Flash очень просто. Для этого необходимо использовать методы fsCommand и Flash Player. Действие fsCommand можно передавать в функцию JavaScript HTML-файла, который открывается в новом окне.

Кроме того, компоненты Flash можно использовать неоднократно. Они являются единственным и самым важным средством коллективного и повторного использования элементов во Flash MX.

Совет

Подробная информация о компонентах приведена в главе 22 "Компоненты".

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Я поместил файл в библиотеку коллективного использования, но от там не появился. Что нужно сделать?

Эта проблема возникает чаще всего. Следует проверить, чтобы данные связи в исходном и целевом файле были совершенно одинаковыми.

При использовании функции удаленного доступа во Flash нужно ли объявлять, что в методах Java перемещены исключения?

Нет, совсем не нужно. Обычно исключения перемещают в случае необходимости трансляции прикладных исключений Flash-клиенту, но поскольку сам шлюз не обрабатывает выделенные исключения, то и объявлять их не нужно.

Каковы два обработчика объекта локального соединения?

Обработчик `LocalConnection.allowDomain` активизируется в случае, когда принимающий объект локальной связи получает запрос об активизации метода отправляющего объекта локальной связи.

Обработчик `LocalConnection.onStatus` активизируется после того, как отправляющий объект локальной связи попытается отправить команду принимающему объекту локальной связи.

Почему у меня открывается диалоговое окно `Resolve Library Conflict` (Разрешение конфликта библиотеки) ?

Это диалоговое окно открывается при попытке импортирования или копирования элемента библиотеки в фильм, уже имеющий элемент с тем же именем. Для разрешения конфликта можно выбрать опцию `Don't Replace Existing Items` (Не заменять существующие элементы), чтобы сохранить существующие элементы в целевом фильме. Либо же можно выбрать опцию `Replace Existing Items` (Заменять существующие элементы), чтобы заменить существующие элементы и экземпляры на новые элементы с тем же именем.

FLASH ЗА РАБОТОЙ: ФУНКЦИЯ LOCALCONNECTION

Читая эту книгу, вы обнаружили, что Flash MX имеет много новых, исключительно мощных и полезных функций. Одной из них является класс `LocalConnection`, который обладает методами, позволяющими отправлять данные из одного фильма в другой. Во Flash MX это можно делать без использования JavaScript или действия `FsCommand`.

Класс `LocalConnection` позволяет отправлять строки и целые объекты (в том числе и свойства) из одного фильма в другой, причем выполнять эти операции можно даже в различных браузерах.

СОЗДАНИЕ ЛОКАЛЬНОЙ СВЯЗИ

Класс `LocalConnection` имеет четыре метода: `send()`, `connect()`, `close()` и `domain()`. Сейчас мы рассмотрим только методы `send()` и `connect()`, требующиеся для передачи данных между двумя фильмами в одном домене.

Для начала создайте фильм для получения сообщений.

1. Создайте во Flash новый фильм и присвойте файлу имя **receiving fla**.
2. В файле создайте динамическое текстовое поле. Его экземпляру присвойте имя **MessSent**.
3. В кадр 1 панели Actions (Действия) поместите приведенный ниже код принимающего фильма.

```
receiving_lc = new LocalConnection();
receiving_lc.methodToExecute = function (param) {
    MessSent.text = param;
}
receiving_lc.connect("lc_test");
```

4. Сначала этот код с помощью имени переменной создает объект `LocalConnection` и правильный конструктор класса `LocalConnection`.
5. Затем выполняется определение метода, вызываемого при получении принимающим фильмом данных из отправляющего фильма. В данном случае экземпляр `MessSent` содержит строку, переданную отправляющим фильмом.

6. Воспользуйтесь методом `connect` О для установления соединения с локальным объектом, имеющим имя `lc_test`.
7. Сохраните свою работу.

Теперь создайте фильм для отправления данных.

1. Создайте во Flash новый фильм и присвойте файлу имя **sending fla**.
2. В файле создайте текстовое поле. Его экземпляру присвойте имя **MessSending**.
3. В фильм поместите кнопку и присвойте ее экземпляру имя **sendButton**.
4. Как и в случае с принимающим файлом, на панели Actions в кадр 1 поместите приведенный ниже код.

```
sendButton.onRelease = function() {  
    sending_lc = new LocalConnection();  
    sending_lc.send("lc_test", "methodToExecute", userMessage.text);  
    delete sending_lc;  
};
```

5. После сохранения работы вставьте оба фильма на одну HTML-страницу и протестируйте их в браузере с установленным модулем Flash 6 Player.
6. В первое текстовое поле введите сообщение и щелкните на расположенной под ним кнопке. В результате текст должен появиться во втором текстовом поле, как показано нарис. 24.4.

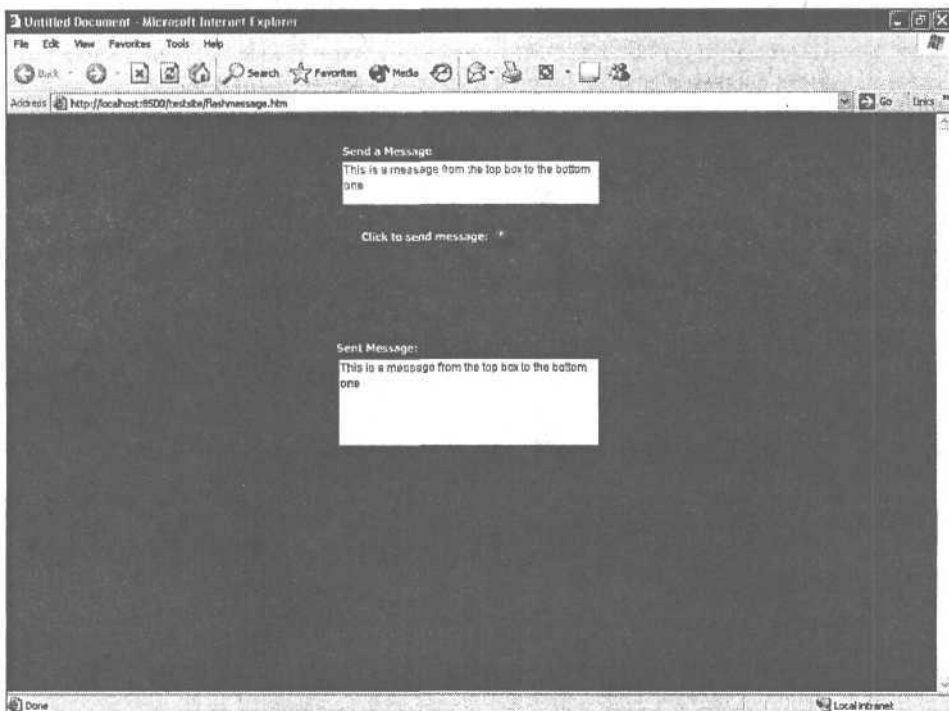


Рис. 24.4. Процедура отправления сообщений между фильмами одного домена очень проста: достаточно щелкнуть на кнопке, и нужный текст появится в поле

Как видно из приведенного примера, процедура установления данного соединения очень проста. Хотя текстовые поля располагаются на одной странице, все происходило бы точно так же, даже если бы они располагались на разных страницах. Данная процедура будет выполняться в любом случае, если фильмы находятся в пределах одного домена.

ОТПРАВЛЕНИЕ СООБЩЕНИЙ МЕЖДУ ДОМЕНАМИ

Сообщения можно отправлять и из одного домена в другой, однако установление такого соединения более сложно, и при его задании необходимо учитывать различные моменты, связанные с безопасностью.

Основное отличие междоменной взаимосвязи заключается в необходимости определения в обоих фильмах параметра `connectionName`. Это делается для того, чтобы во Flash не произошло добавление текущего субдомена. Данная задача выполняется путем добавления символа подчеркивания в начале параметра `connectionName` в обоих фильмах:

```
receiving_lc = new LocalConnection();
receiving_lc.aMethod = function(param) {
    trace(param);
}
receiving_lc.allowDomain = function(senderDomain) {
    return (senderDomain == "allowedDomain.com");
}
receiving_lc.connect("_lc_test");
```

Объект `LocalConnection` в отправляющем фильме из другого домена может затем использовать метод `send()` для отправления сообщений в целевой фильм:

```
sending_lc = new LocalConnection();
sending_lc.send("myDomain:_lc_test", "aMethod", "Это строковый параметр");
delete sending_lc;
```


ТЕСТИРОВАНИЕ, ОТЛАДКА и УСТРАНЕНИЕ ПРОБЛЕМ

В ЭТОЙ ГЛАВЕ...

Как избежать наиболее распространенных проблем	589
Использование отладчика	594

КАК ИЗБЕЖАТЬ НАИБОЛЕЕ РАСПРОСТРАНЕННЫХ ПРОБЛЕМ

Общая концепция локализации неполадок является в большей степени предметом здравого смысла, чем глубоких технических познаний. Ведь до сих пор очень многие пользователи сталкиваются с проблемами, которых можно было избежать, обладая достаточным опытом. Всегда считалось, что профилактика гораздо лучше, чем лечение. На практике хорошее планирование и тестирование сцен и фильмов на ранних этапах работы всегда делает процесс локализации неполадок менее проблематичным и более легким по сравнению с проведением аналогичных операций в конце проекта.

При работе над проектом всегда применимы некоторые стандартные правила, независимо от того, создаете ли вы небольшой Flash-фильм или разрабатываете огромный проект. В данной главе мы рассмотрим эти правила и расскажем о том, как решить те или иные проблемы в случае их возникновения.

В большинстве случаев проблемы, возникающие при работе над фильмами, можно свести к одной из трех категорий: фильм не воспроизводится вообще, фильм не воспроизводится в системе или не функционирует программное обеспечение.

Первая категория проблем, как правило, вызвана ошибками, допущенными пользователем. Проблемы с файлами могут быть вызваны изменением переменных, неправильным при-

своением имен переменным в экземплярах или некорректным назначением действий. Тестирование и сохранение файла в процессе работы над ним поможет быстро идентифицировать и устранить ошибки такого рода.

Причину возникновения проблем, связанных с системой, выявить несколько сложнее. Если определенный файл вызывает отказ системы, попробуйте протестировать его в другой системе. Посмотрите, какие новые приложения вы установили в системе, и подумайте, не могут ли они быть причиной проблем. Проверьте и параметры дисплея, поскольку их изменение также может привести к возникновению проблем.

Неработающее или некорректно работающее программное обеспечение может свидетельствовать о наличии проблем конфигурации системы. Устанавливая новую или обновленную версию программы, уверены ли вы в том, что система отвечает соответствующим для этого требованиям? Опять же, проверьте, возникает ли подобная проблема на другой машине. Если это так, то проблема заключается в самом программном обеспечении, а если нет, то, вероятнее всего, причиной возникновения проблемы является ваша система.

Убедитесь, что в проекте отведено достаточно времени для тестирования и любой требующейся отладки. Во-первых, некорректно воспроизводимым фильмом никто не сможет воспользоваться, а во-вторых, такой продукт наверняка не улучшит вашу репутацию как дизайнера или разработчика. Стремление побыстрее выполнить проект в ущерб планированию приведет к возникновению ошибок, а недостаточное время, выделенное на тестирование, почти неизбежно приведет к неудовольствию заказчика и отрицательно скажется на дальнейшем бизнесе.

При планировании любых Flash-проектов, независимо от того, насколько они велики и сколько в них задействовано людей, обязательно необходимо выделить достаточно времени для тестирования. Процедура тестирования и решения проблем на каждом этапе разработки позволяет контролировать все нюансы, которые, не будучи разрешенными, на последующих стадиях разработки могут превратиться в огромную проблему, решить которую будет уже гораздо сложнее.

Работая над несколькими фильмами, которые должны будут воспроизводиться все **вместе**, всегда проверяйте каждый из них и только потом устанавливайте связь между ними. Опять же, эта процедура поможет сузить, идентифицировать и выявить источник возникновения тех или иных проблем на ранних стадиях работы над проектом.

При наличии в фильме нескольких сцен протестируйте каждую из них отдельно и при обнаружении проблемы постарайтесь локализовать место ее возникновения и выяснить причину отказа.

При переходе от старой версии Flash к версии Flash MX проверьте, что вы используете правильный синтаксис кода и выражений.

Совет

Чтобы узнать, как следует корректно использовать текущую версию ActionScript, обратитесь к главе 11 "Знакомство с ActionScript".

Регулярно сохраняйте свои проекты под разными именами и в разных местах. Сохранив корректно воспроизводящуюся предыдущую версию фильма, вы всегда сможете вернуться к ней и найти точное место, в котором начинается сбой, либо выяснить, внесение каких изменений в сценарий без видимых причин привело к некорректному воспроизведению фильма.

Необходимо отметить, что Flash-файлы никогда не сбоят и не перестают неожиданно работать без соответствующего "вклада" со стороны пользователя. Не может быть такого, чтобы когда-то в процессе разработки кто-то (может быть, даже вы сами) не внес в фильм каких-то изменений или дополнений. Уверенность в возможности найти ошибку значительно облегчает процедуру ее устранения и требует значительно меньших затрат.

Наличие тщательно разработанного плана работ позволит в нужный момент вернуться к одной из предшествующих фаз разработки, не затрачивая на это лишних усилий. Каким бы утопическим не показался этот тезис, примите его во внимание. Хорошее планирование на ранних этапах разработки поможет выполнить в проект в целом гораздо более успешно.

МНОГОПЛАТФОРМЕННОЕ ТЕСТИРОВАНИЕ

Если вы не разрабатываете фильм, предназначенный для строго определенной аудитории, очень велики шансы, что созданный вами продукт увидят пользователи, работающие на различных типах машин с самыми разными платформами. По этой причине очень важно быть уверенным в том, что на каждой из них фильм будет воспроизводиться корректно.

Может показаться, что сейчас мы делаем шаг назад, однако обеспечение доступности фильма для надлежащей аудитории является неотъемлемой частью планирования проекта. Если вы хотите, чтобы ваш проект был солидным, то процедура планирования должна быть выполнена столь же тщательно и кропотливо, как и процедура самой разработки.

После выяснения минимальных системных требований к проекту фильм необходимо все время тестировать исходя из этих требований.

Лучше всего выполнять тестирование на нескольких машинах, но, к сожалению, это не всегда возможно. Далеко не каждый разработчик имеет доступ к большому количеству различного качества машин с разными операционными системами. Тем не менее нельзя предполагать, что фильм, превосходно воспроизводящийся на компьютере с операционной системой Windows 2000, будет столь же безупречным на компьютере Macintosh со старой операционной системой.

Проблемы, связанные с операционной системой, не отразятся на ваших навыках дизайнера и разработчика, но они важны с точки зрения разрешения.

Зная о недостатках системы и планируя способы их обхода в случае невозможности создания универсального решения, вы сможете сделать свой проект доступным для использования максимально широкой целевой аудиторией. Если же вы завершите разработку великолепного фильма, а потом обнаружите, что его могут просматривать только 10% предполагаемых пользователей, то фильм наверняка придется практически полностью переделывать, а это явно неприемлемая стратегия даже для самых неопытных Flash-дизайнеров и разработчиков.

Говоря простыми словами, чем тщательнее спланирован проект, тем лучший результат будет достигнут.

ЭФФЕКТИВНЫЕ ПРОЦЕДУРЫ ТЕСТИРОВАНИЯ

Руководствуясь здравым смыслом и предпринимая разумные меры предосторожности для того, чтобы затрачивать минимум усилий на выявление и устранение неполадок, наилучших результатов тестирования можно добиться, следуя приведенным ниже советам. Их можно использовать в качестве основных правил эффективного тестирования.

- Для тестирования собственных творений всегда пользуйтесь командой **Control⇨Test Movie** (**Управление⇨Тестировать фильм**). В отличие от остальных процедур, имеющих ограниченные возможности, данный путь лучше всего позволяет смоделировать законченный фильм. Все ссылки, звуки и элементы навигации должны быть в полном порядке. Использование данной команды является оптимальным способом быстрого и легкого выявления проблем.
- При обнаружении ошибок попытайтесь выделить проблемную область фильма в отдельный файл, а потом посмотрите, продолжает ли проблема существовать. Этот тест поможет понять, связана ли проблема с определенной областью фильма или с какими-то другими частями проекта.
- Протестируйте весь фильм на другом компьютере. Этот тест подскажет, связана ли имеющаяся проблема с используемым аппаратным обеспечением, и даже поможет выявить, связана ли она с версией Flash, установленной на данном компьютере.
- Протестируйте файл на сервере. Если проблемы обнаружатся на одном сервере, попробуйте протестировать фильм на другом. Этот тест подскажет, связана ли проблема с исходным сервером или с серверами вообще.

- При наличии ошибок, связанных с серверами, проверьте наличие на проблемном сервере соответствующих типов **MIME**, связанных с расширением **.swf**. Подробная информация по этому вопросу представлена на Web-узле Macromedia по адресу <http://www.macromedia.com>.
- Если в проблемный фильм или проект включен код **ActionScript**, то прежде чем впадать в панику по поводу устранения ошибки, проверьте правильность синтаксиса, присвоения имен и использования знаков пунктуации. Все эти ошибки вызывают кажущиеся очень значительными проблемы, решение которых на самом деле требует минимального количества усилий.

ОПТИМИЗАЦИЯ

Несмотря на то что оптимизация файлов не относится напрямую к решению проблем, данную процедуру, во избежание возникновения ошибок, все же желательно выполнять. Как уже было сказано, профилактика всегда лучше, чем лечение. Определенные методики оптимизации основываются исключительно на здравом смысле, но, тем не менее, помогают сэкономить время и избежать последующей головной боли.

Во Flash не требуется выполнять все операции самостоятельно, поскольку встроенные функции публикации автоматически удалят дублируемые контуры и добавят их в конечный файл всего один раз. Подробная информация о сжатии файлов в целях уменьшения размера и времени загрузки приведена в главе 30 "Оптимизация, публикация и экспортирование фильмов".

Совет

Об оптимизации фильмов также рассказывается в главе 30 "Оптимизация, публикация и экспортирование фильмов".

ТЕСТИРОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ЗАГРУЗКИ

Тестирование производительности загрузки фильма позволяет убедиться, что фильм можно будет просматривать надлежащим образом. Нет ничего хуже, когда обнаруживается, что фильм, который вы так тщательно планировали и скрупулезно создавали, воспроизводится с паузами, когда плеер не может достичь заданной частоты смены кадров.

Можете воспользоваться подпрограммой **Bandwidth profiler** (Профайлер полосы пропускания), с помощью которой создается графический отчет о загрузке, чтобы увидеть, какое количество данных пересылалось в каждом кадре. Такой отчет составляется на основе заданной скорости соединения. Фильм следует протестировать на как можно большем количестве разных машин и с различными скоростями соединения. Если это невозможно, Flash попытается смоделировать подобные условия.

Для тестирования производительности загрузки выполните следующие действия.

1. Из меню **Control** (Управление) выберите либо пункт **Test Movie** (Тестировать фильм), либо **Test Scene** (Тестировать сцену).
2. Из меню **Debug** (Отладка) выберите скорость соединения, при которой вы хотите выполнить тестирование. Чтобы задать свои собственные параметры, откройте диалоговое окно **Custom Modem Settings** (Настройка параметров модема), выполнив команду **Debug**⇒**Customize** (Отладка⇒Настройка).
3. Из меню **View** (Вид) выберите пункт **Bandwidth** (Полоса пропускания), в результате чего будет создано графическое представление настроек.

Изменить способ отображения данных профайлером можно путем изменения настроек меню **View**, в котором содержатся следующие опции.

- **Show Streaming** (Показать поток). Переключение воспроизведения между имитацией Web-соединения и его отсутствием. Изменение данной настройки в процессе воспроизведения приводит к повторному запуску фильма.
- **Streaming Graph** (Потоковая диаграмма). Показывает, какой из кадров приостановит воспроизведение. Блоки кадров показаны разными цветами. Обычно первый кадр больше остальных, поскольку в нем содержится больше информации. Щелчок на любом кадре диаграммы приведет к отображению данных кадра на панели в левой части окна.
- **Frame by Frame Graph** (Покадровая диаграмма). Показывает любые кадры, которые приводят к приостановке воспроизведения. Красная линия, подобная той, что на рис. 25.1 расположена на отметке 100 В, является направляющей. Все кадры выше этой линии вызовут приостановку фильма, а остальные кадры будут загружаться нормально.

Во Flash можно создать текстовый файл с покадровой разбивкой данных конечного файла фильма. Для этого из меню **File** (Файл) выберите пункт **Publish Settings** (Параметры публикации), а затем установите флажок в поле опции **Generate Size Report** (Генерировать отчет о размере). В результате Flash создаст текстовый файл с тем же именем, что и файл **.fla**. Например, файлу **movie1.fla** будет соответствовать файл **movie1report.txt**.

ПОИСК И УСТРАНЕНИЕ НЕПОЛАДОВ В КОДЕ ACTIONSCRIPT

Как и любой другой язык сценариев, **ActionScript** время от времени дает сбои. Чем осторожней вы будете обращаться со сценарием, тем большего количества проблем удастся избежать. Не важно, насколько опытным пользователем **ActionScript** вы являетесь, — определенные действия вызваны исключительно здравым смыслом и помогут избежать ловушек. Приведенные ниже советы помогут работать логически и своевременно устранить проблемы.

- Сопровождайте код **ActionScript** подробным комментарием о том, как он должен работать. Комментарии помогут другим разработчикам не запутаться в вашем тщательно разработанном коде, а при работе над большими проектами помогут запомнить, почему код создан именно таким образом.

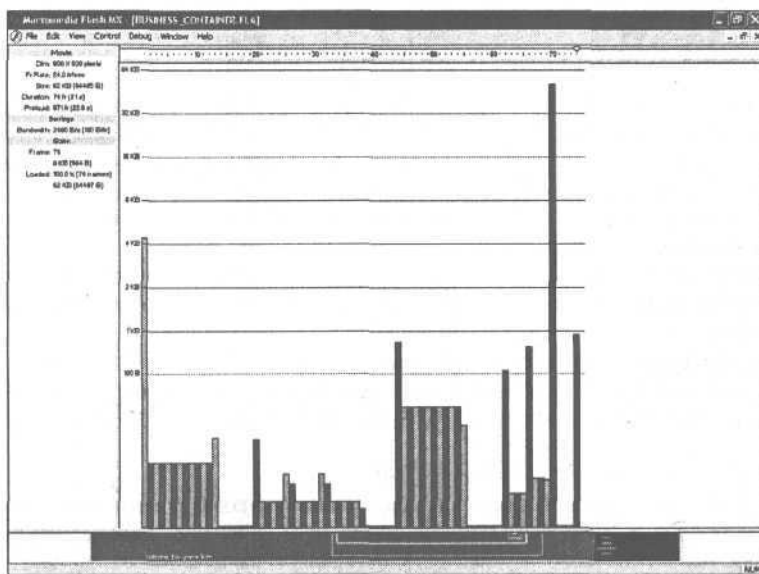


Рис. 25.1. Профайлерполосы пропускания позволяет локализовать проблемную область в файлах фильма

- Присваивайте элементам кода имена, несущие логическую нагрузку. При наличии 15 или 20 различных кнопок присвоение имени `button` для какой-то из них будет совершенно бессмысленным. Имя `SmallDepressedButton`, по крайней мере, описывает этот элемент в целом. Каждое из слов, составляющих имя элемента, начинайте с прописной буквы. Это делает имена более доходчивыми.
- Убедитесь в том, что присвоенные имена несут смысловую нагрузку, особенно это касается переменных. Опять-таки, полезная информация, содержащаяся в имени переменной, поможет при выявлении неполадок. Так, переменную, содержащую информацию об электронной почте, гораздо лучше назвать `EmailForm01`, чем просто `email`.

Помните, что для просмотра всех элементов фильма и кода ActionScript предназначена панель Movie Explorer (Проводник фильма).

Совет

Подробно об устранении неполадок в коде ActionScript рассказывается в главе 11 "Знакомство с ActionScript".

ИСПОЛЬЗОВАНИЕ ОТЛАДЧИКА

Итак, вы предприняли все разумные меры для того, чтобы фильм воспроизводился корректно, но все равно что-то не так. Пришло время вооружиться инструментами Flash, предназначенными для выявления и устранения проблем, причины возникновения которых выходят за рамки обычного здравого смысла.

Встроенная во Flash MX подпрограмма Debugger (Отладчик) имеет гораздо лучшие характеристики по сравнению с отладчиком, существовавшим в версии Flash 5. Следовательно, данная подпрограмма является гораздо более мощным инструментом.

Моей любимой характеристикой отладчика является возможность его работы как с локальными, так и с серверными файлами. При открытии панели Debugger (посредством команды **Control⇨Test Movie**) не только автоматически запускается сама подпрограмма, но и загружается текущий фильм в режиме тестирования. На рис. 25.2 показан экран отладки.

КОНТРОЛЬНЫЕ ТОЧКИ И ПРОХОЖДЕНИЕ ПО НИМ

Контрольные точки управляют фильмом в процессе его воспроизведения во Flash-плеере. Они позволяют протестировать сценарии, вызывающие проблемы, внести необходимые исправления и снова протестировать фильм. Пусть, к примеру, код содержит ряд предложений, и вы знаете, что в нем есть ошибка, но не уверены, где именно. Добавив контрольные точки перед каждым предложением кода, можно пройти по очереди по всем ним и выяснить, где возникает ошибка.

Прохождение по коду от одной контрольной точки к другой гарантирует, что вы сможете устранить ошибку на одном участке кода, а затем перейти к следующей потенциально проблемной области.

Задать контрольные точки можно как в самом файле `.fla` с целью их постоянного использования, так и в пределах отладчика. Если для задания контрольных точек используется отладчик, то они действительны только в течение текущего сеанса, а затем удаляются.

ЗАДАНИЕ И УДАЛЕНИЕ КОНТРОЛЬНЫХ ТОЧЕК

Чтобы использовать отладчик для задания или удаления контрольных точек, в окошке сценариев на панели Actions (Действия) найдите строку кода, в которую необходимо вставить контрольную точку. Вероятнее всего, вставка будет производиться в начале блока предложений, например предложений `if-else`, которые не работают надлежащим образом. Затем откройте меню Debug Options (Параметры отладки) и выберите соответствующий параметр меню.

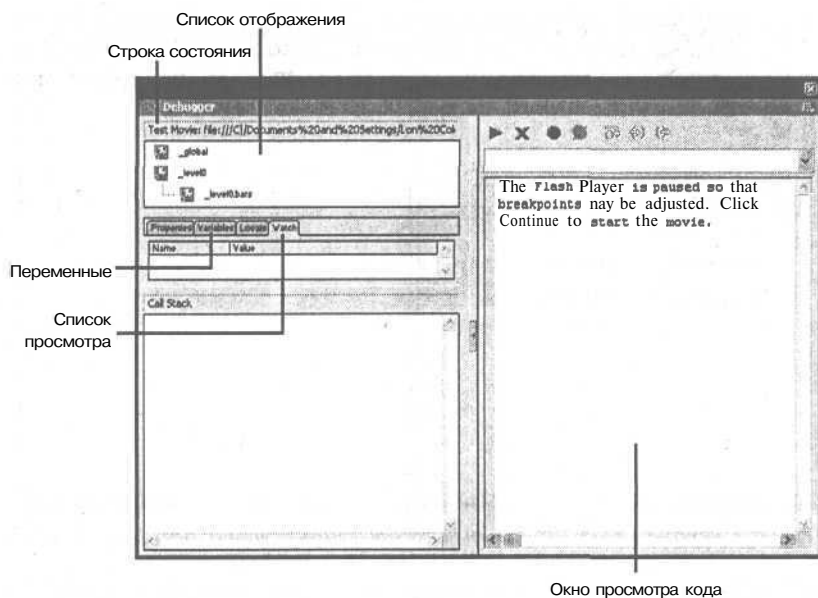


Рис. 25.2. Отладчик позволяет поэтапно пройти по коду фильма с целью поиска и устранения ошибок

Для задания или удаления контрольных точек на панели Debugger выполните следующие действия.

1. Найдите строку кода, в которой следует добавить либо удалить контрольную точку.
2. Щелкните на кнопке Toggle Breakpoint (Включить контрольную точку) или Remove All Breakpoints (Удалить все контрольные точки), которые расположены над окном просмотра кода.
3. В качестве альтернативного способа можно открыть меню параметров и выбрать из него подходящий пункт.
4. Когда плеер остановится в контрольной точке, воспользуйтесь управляющими кнопками (рис. 25.3) для прохождения по коду.

ПОШАГОВОЕ ПРОХОЖДЕНИЕ по КОДУ

Процесс перемещения по коду от одной контрольной точки к другой называется *пошаговым прохождением*. В начале сеанса отладки Flash-плеер автоматически приостанавливается, чтобы вы могли управлять сеансом. Если контрольные точки заданы посредством панели Actions, щелкните на кнопке Play (Воспроизведение). Тогда воспроизведение фильма будет выполняться до достижения первой контрольной точки.

Если контрольные точки заданы в среде отладки, то для перемещения к первой из них следует воспользоваться управляющими кнопками.

В процессе перемещения по коду содержимое, отображаемое на панели Debugger, изменяется. Список просмотра, переменные, локальные элементы и свойства изменяются в соответствии с отображаемыми текущими данными кода. В окошке просмотра кода желтой стрелкой помечается место остановки.

Панель, расположенная над окошком просмотра кода (см. рис. 25.3), позволяет управлять процедурой отладки.

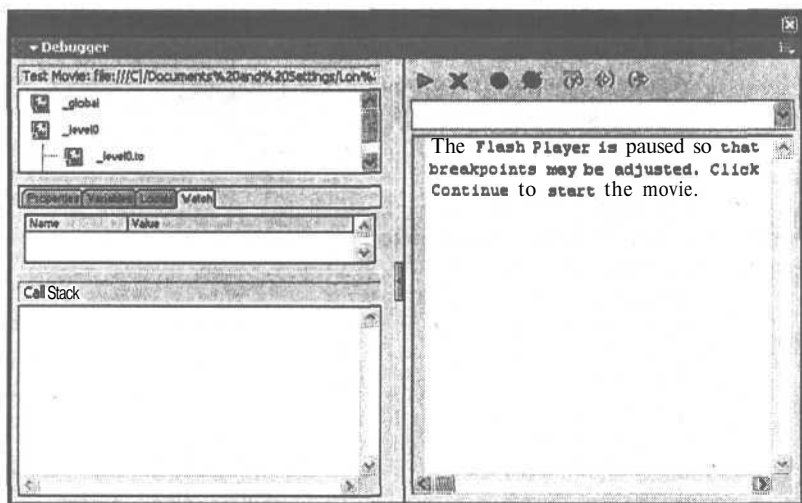


Рис. 25.3. Используйте управляющие кнопки для отладки фильма на панели **Debugger**

- **Step In (Шаг в).** Эта кнопка применима только к функциям, заданным пользователем. После щелчка на ней отладчик перемещается или делает шаг внутрь функции.
- **Step Out (Шаг из).** Эта кнопка функционирует только в случае, если уже сделан шаг в функцию.
- **Step Over (Шаг через).** С помощью этой кнопки отладчик перемещается вперед на одну строку кода независимо от функций.
- **Continue (Продолжить).** После щелчка на данной кнопке фильм продолжает воспроизводиться с его текущего места и до следующей контрольной точки.
- **Stop Debugging (Прекратить отладку).** После щелчка на этой кнопке фильм продолжает воспроизводиться во Flash-плеере, но отладчик не функционирует.

Окно OUTPUT

Окно Output (Вывод данных), которое открывается после выполнения команды **Window⇒Output**, было разработано специально для того, чтобы помочь решить проблемы, обнаруженные в фильме. Синтаксические ошибки, например, отображаются в окне Output автоматически. Воспользовавшись командами **List Objects** (Перечислить объекты) и **List Variables** (Перечислить переменные), можно расширить диапазон отображаемой информации, что даст большие возможности поиска и устранения проблем.

Использование команды **List Objects** является самым простым способом поиска целевых путей и имен экземпляров. При работе в текстовом режиме в окне вывода автоматически отображаются любые синтаксические ошибки в коде, которые можно сразу же исправить и затем продолжить тестирование. Команду **Trace Action** (Отследить действие) можно использовать для передачи определенной информации в окно вывода во время фактического воспроизведения фильма. Данная команда будет подробнее рассмотрена далее в этой главе.

Работать с информацией, отображаемой в окне Output, можно с помощью меню параметров, расположенного в правом верхнем углу окна.

Элементы данного меню соответствуют выполнению следующих команд.

- **Copy (Копировать).** Содержимое окна помещается в буфер обмена.
- **Clear (Сбросить).** Производится сброс содержимого окна Output.

- **Save to File** (Сохранить в файл). Содержимое окна Output сохраняется в виде текстового файла с целью **последующего** выявления и устранения ошибок.
- **Print** (Печать). Выполняет печать содержимого окна Output.
- **Find** (Найти). Осуществляет поиск строки текста.
- **Find Again** (Найти далее). Повторяется предыдущий поиск.

РАБОТА С ПЕРЕМЕННЫМИ В ОКНЕ OUTPUT

При тестировании фильма можно воспользоваться командой List Variables (Перечислить переменные), в результате выполнения которой в окне Output отображается список текущих переменных. Основное отличие использования этой команды от отладчика заключается в том, что список без повторного применения данной команды обновляться не будет. Каждый раз, когда необходимо отобразить обновленные данные, следует воспользоваться командой List Variables.

На рис. 25.4 показан список переменных фильма. Обратите внимание на то, что вверху списка перечислены глобальные переменные.

В списке переменных показаны также свойства, созданные с помощью метода `Object.addProperty` и активизируемые посредством метода `set` или `get`. Они отображаются рядом с остальными свойствами объекта, а отличить их можно по наличию строки получения/задания.

ИСПОЛЬЗОВАНИЕ ДЕЙСТВИЯ ОТСЛЕЖИВАНИЯ

Использование действия отслеживания в сценарии позволяет передавать информацию из тестируемого фильма в окно Output. Чаще всего действие отслеживания выполняется для того, чтобы проверить, был ли успешным вызов функции. Наиболее распространенная при-



Рис. 25.4. При выполнении команды List Variables в окне вывода отображается список переменных фильма с указанием свойств каждой из них

чина ненадлежащего воспроизведения Flash-фильма заключается в том, что указан неправильный путь к видеоклипу, переменной, объекту или функции. Даже такая кажущаяся простой ошибка может привести к тому, что переменные будут иметь неправильные значения, а обращение к функциям будет выполняться некорректно или не выполняться вообще. Протестировать этот аспект позволяет действие отслеживания.

Изъясняясь терминами сценариев, действие отслеживания похоже на оператор `alert` в JavaScript. Это действие можно использовать для передачи определенной информации или примечаний о программе с целью их отображения. После этого кадр воспроизводится в тестовом режиме.

Для применения действия отслеживания в качестве параметров необходимо использовать выражения. В окне Output будет отображено значение выражения.

СРЕДСТВА ДИСТАНЦИОННОЙ ОТЛАДКИ

Хотя отладку фильма можно выполнить в любом из трех типов Flash-плеера (автономном, подключаемом модуле или Active X), подготовку к этой операции необходимо провести при публикации файла, но не позже.

Если в диалоговом окне Publish Settings (Параметры публикации) не установить опцию отладки, то дистанционно активизировать экран отладки будет невозможно.

Для задания свойств удаленной отладки выполните следующие действия.

1. Выполните команду **File**⇒**Publish Settings** (**Файл**⇒**Параметры публикации**), в результате чего откроется одноименное диалоговое окно (рис. 25.5).
2. В диалоговом окне Publish Settings выберите вкладку Flash и установите флажок в поле опции Debugging Permitted (Отладка разрешена).
3. В поле Password (Пароль) введите пароль. Этот пароль гарантирует, что доступ к экрану отладки сможете получить только вы или те люди, с которыми вы сотрудничаете.

Теперь все готово к публикации фильма на сервере.

4. Выгрузите вновь созданный файл `.swd` в то же самое место на сервере, что и файл `.swf`.

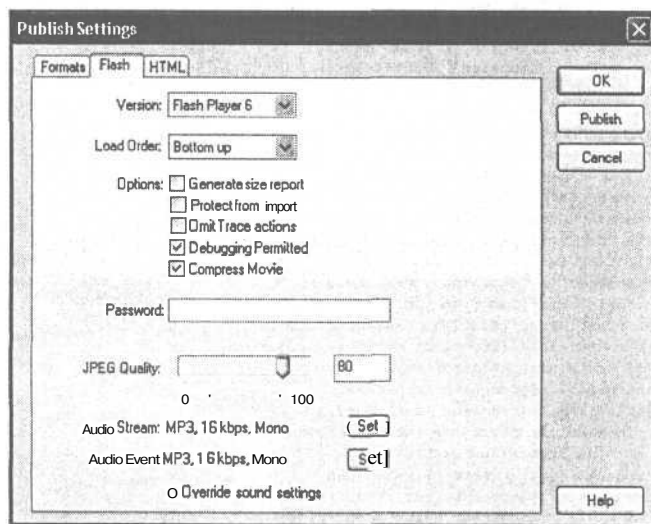


Рис. 25.5. Для включения дистанционной отладки необходимо сначала установить флажок в поле опции разрешения отладки диалогового окна Publish Settings

5. Во Flash выполните команду **Window⇨Debugger**, а затем из меню параметров панели выберите пункт **Enable Remote Debugging** (Разрешить дистанционную отладку).

На заметку

После выполнения описанных выше действий Flash создаст новый файл с тем же именем, что и файл `.swf`. Этот файл имеет расширение `.swd` и является очень важным для дистанционной отладки.

Теперь отладку фильма можно выполнять дистанционно.

АКТИВИЗАЦИЯ ОТЛАДЧИКА УДАЛЕННЫМ ОБРАЗОМ

Предполагая, что файл `.swd` правильно выгружен в то же самое место на сервере, что и файл `.swf`, процедура удаленной отладки будет работать превосходно. Если файл `.swd` не найден, то функция пошагового прохождения кода не будет выполняться, даже несмотря на то, что будет открыто окно отладчика.

Хотя вы работаете с удаленным файлом, сначала необходимо открыть программу Flash MX.

1. В окне браузера (или Flash-плеера) откройте удаленно расположенный `.swf`-файл. Обратите внимание на то, что открывается файл `.swf`, а не `.fla`.
2. Откроется диалоговое окно **Remote Debug** (Дистанционная отладка). Если оно не открывается, это значит, что не найден файл `.swd`. В этом случае необходимо открыть контекстное меню фильма и выбрать из него пункт **Debugger** (Отладчик).
3. В диалоговом окне **Remote Debug** выберите соответствующую опцию в зависимости от того, установлены ли средства разработки Flash и плеер отладчика на одной машине. Если указанные средства установлены на разных машинах, введите IP-адрес другой машины.

Щелкните на кнопке **ОК** и подождите, пока будет установлено соединение с файлом `.swf`. Когда будет отображено соответствующее приглашение, введите пароль, который был задан ранее (см. выше подраздел "Средства дистанционной отладки").

В окне отладки должен быть виден список отображения фильма.

ДО СИХ ПОР ОСТАЛИСЬ ПРОБЛЕМЫ?

В этой главе были рассмотрены встроенные средства тестирования и отладки Flash-файлов. Если описанные здесь средства и советы по устранению проблем, рассматриваемые в конце каждой из глав, вам не помогли, воспользуйтесь нижеприведенными советами.

Начните с посещения узла Macromedia (www.macromedia.com). Центр поддержки (Flash Support Center) является превосходным источником не только полезных советов и подсказок, но также технических заметок, связанных с определенными проблемами. Кроме того, можно присоединиться к форумам, посвященным Flash, задать вопросы и найти ответы, касающиеся проблем, которые вы не смогли решить самостоятельно.

Форумы и другие типы поддержки расположены в соответствующей части Web-узла Macromedia. Если вы хотите попасть прямо туда, обратитесь по адресу <http://www.macromedia.com/support/flash>.

Кроме того, на Web-узле Macromedia приведены ссылки на другие узлы, предлагающие помощь и советы опытных Flash-дизайнеров и разработчиков. Независимо от уровня вашей квалификации, есть вероятность того, что проблема, с которой вы столкнулись, уже была кем-то решена.

Одним из самых приятных моментов работы с Flash и другими продуктами от Macromedia является то, что люди, использующие их, готовы поделиться информацией и помочь решить возникающие проблемы.

РЕАЛИЗАЦИЯ ВНЕШНИХ СВЯЗЕЙ С ПОМОЩЬЮ FLASH

В ЭТОЙ ЧАСТИ...

Глава 26. Локальная связь

Глава 27. Взаимосвязь с сервером

Глава 28. XML-данные

ЛОКАЛЬНАЯ СВЯЗЬ

В ЭТОЙ ГЛАВЕ...

Управление браузером	603
Вызов функций JavaScript	605
Управление проектором	614
Взаимосвязь с программой Director	615
Хранение информации локально	620
Взаимосвязь Flash-фильмов	622
Возможные проблемы	622

УПРАВЛЕНИЕ БРОУЗЕРОМ

Flash-фильмы могут воспроизводиться в одном из нескольких окружений, наиболее распространенным из которых является Flash-плеер в Web-браузере. Кроме того, Flash-фильм может выполняться в виде автономного проектора или внутри фильма Macromedia Director. Во всех перечисленных случаях фильм может разными способами взаимодействовать с окружением. В данной главе мы рассмотрим каждое окружение и способы взаимосвязи фильма с ним.

Flash-фильмы могут функционировать подобно HTML-дескриптору `<A HREF>` при просмотре в Web-браузере, являясь элементом навигации браузера и изменяя отображаемую страницу. Кроме того, они могут влиять на страницу, отображаемую в других окнах и фреймах браузера.

ИСПОЛЬЗОВАНИЕ КОМАНДЫ GETURL

Основным инструментом управления браузером является команда `getURL`. Эта команда отправляет сообщение браузеру, указывая ему на необходимость загрузки новой страницы с заданного URL. Исполнение команды `getURL` подобно щелчку на ссылке в дескрипторе `<A HREF>` браузера.

В приведенном ниже сценарии кнопки команды `getURL` используются для загрузки страницы, имеющей имя `newpage.html`.

```
on (release) {  
    getURL("newpage.html");  
}
```

URL — это унифицированный указатель информационного ресурса, который является единственным необходимым параметром команды `getURL`.

Существуют два типа URL: относительные и абсолютные. Относительный URL может быть локальным файлом, как, например, `newpage.html` в предыдущем примере. Он также может задавать относительный путь к файлу или каталогу.

Примером относительного URL, указывающим файл в каталоге, является `myfolder/newpage.html`. В данном случае указан файл `newpage.html`, находящийся в папке `myfolder`.

Абсолютный URL представляет собой полный путь к файлу или каталогу на сервере. Примером абсолютного URL является `http://www.mysite.com/newpage.html`.

Команду `getURL` с относительным URL можно использовать для загрузки страницы на том же сервере, на котором находится текущая Web-страница. Кроме того, относительный URL можно использовать для загрузки страницы, когда она находится на локальном жестком диске или когда фильм выполняется как проектор.

На заметку

Помните о том, что Flash-плеер не выполняет фактическую загрузку страницы. Он отправляет сообщение браузеру, который затем выполняет всю необходимую работу.

Абсолютный URL используется в команде `getURL` только при наличии соединения с Internet, поскольку в этом случае, как правило, производится обращение к узлу на Web-сервере, а не к странице на локальном компьютере.

Поскольку сила команды `getURL` заключена не во Flash, а в Web-браузере, то в качестве параметра команды можно использовать любой элемент, который браузер сможет распознать как адрес. Например, если вы хотите, чтобы браузер перечислил файлы, находящиеся на FTP-сервере, то можно использовать URL, подобный `ftp://ftp.myserver.com/`. Если данный браузер поддерживает FTP и искомый **FTP-узел существует**, то будут перечислены находящиеся на нем файлы.

ЦЕЛЕВЫЕ ОКНА и ФРЕЙМЫ

Вторым параметром команды `getURL` является целевой объект. Им может быть окно и фрейм браузера, а также специальная команда.

Во всех Web-браузерах можно открывать несколько **ОКОН**, в каждом из которых будет отображена страница со своим URL. Эти окна могут быть разных размеров и содержать различные панели инструментов (это зависит от того, как были открыты окна и какие настройки были выполнены пользователем). Чтобы отобразить несколько URL в одном окне браузера, его необходимо разделить на фреймы. Это делается на уровне HTML с помощью специальных HTML-дескрипторов. В каждом фрейме может содержаться страница со своим URL, изменение которой не повлияет на содержимое остальных фреймов. В большинстве браузеров в качестве альтернативы URL в адресную строку можно вводить специальные команды. Эти команды изменяют поведение браузера или позволяют получить информацию о нем.

Чтобы задать имя окна или фрейма в качестве параметра команды `getURL`, сначала необходимо присвоить имена фреймам и окнам браузера, а эта процедура требует применения определенного планирования при создании HTML-страниц.

Предположим, имеется страница, состоящая из двух фреймов. В левом фрейме располагается небольшая панель навигации, а в большом правом фрейме — содержимое страницы. Фрейму с содержимым можно присвоить имя `contentFrame`. В навигационном фрейме может содержаться Flash-фильм с кнопками навигации. Одна из этих кнопок может указывать фрейму `contentFrame` на необходимость перехода к новой странице, например, как показано ниже.

```
on (release) {  
    getURL("newpage.html", "contentFrame");  
}
```

Процедура указания целевого фрейма очень проста. Всего одна команда разрешает одному окну браузера, содержащему **Flash-фильм**, управлять страницей (URL), отображаемой в другом окне. Так, в основном окне браузера может содержаться фильм, управляющий меньшим всплывающим окном.

СОЗДАНИЕ НОВЫХ ОКОН БРОУЗЕРА

Если использовать команду `getURL` с несуществующим целевым объектом, то браузер создаст для отображения содержимого новое окно. Например, если целевым объектом является `newWindow`, а окна с таким именем не существует, то откроется новое окно, которому будет присвоено имя `newWindow`.

Когда такое окно существует, к нему можно обращаться по имени, т.е. окно `newWindow` в качестве целевого объекта можно использовать неоднократно. При первом обращении к `newWindow` браузер создаст это окно. При каждом последующем обращении будет изменяться только его содержимое. Таким образом, используя одно и то же имя, можно создать новое окно и впоследствии изменять его содержимое. HTML-содержимое можно показывать пользователю путем создания и повторного использования одного и того же окна.

ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ ЦЕЛЕВЫХ ОБЪЕКТОВ

Помимо задействования имен, можно воспользоваться одним из четырех специальных целевых объектов. Их имена начинаются с символа нижнего подчеркивания. Специальные целевые объекты перечислены ниже.

- `_self`. Эквивалентен отсутствию целевого параметра. Содержимое отображается в текущем окне или во фрейме.
- `_parent`. Указывает на фрейм, расположенный на один уровень выше относительно текущей страницы.
- `_top`. Указывает на окно, в котором располагается текущий кадр или страница.
- `_blank`. Создает новое окно. Использование данного целевого объекта является единственным способом гарантированного открытия нового окна независимо от имен текущих окон.

Чтобы после щелчка на кнопке открылось новое окно, воспользуйтесь следующим сценарием:

```
on (release) {  
    getURL("newpage.html", "_blank");  
}
```

Вызов ФУНКЦИЙ JAVASCRIPT

Несмотря на то что команда `getURL` позволяет загружать с помощью Flash-фильма новые страницы, практически полностью управлять браузером можно посредством команды `fscommand`. С ее помощью можно отправлять сообщения в сценарии JavaScript браузера. Кроме того, Flash-фильм также может получать сообщения из сценариев JavaScript, содержащихся на Web-странице.

ПРОБЛЕМЫ, СВЯЗАННЫЕ с JAVASCRIPT

Прежде чем рассмотреть взаимодействие Flash и JavaScript, необходимо выяснить, какими недостатками обладает JavaScript. Основным недостатком является тот факт, что один и тот же блок кода может не функционировать в некоторых вариациях браузеров.

Для реализации взаимосвязи Flash и JavaScript необходимо наличие программного блока, содействующего ей. Для браузеров Internet Explorer, работающих под управлением Windows, таким блоком программного обеспечения является **Active X**. Для старых версий Netscape Communicator таковым является модуль LiveConnect. Оба модуля являются частью соответствующих браузеров.

Хотя Active X является частью браузера Internet Explorer в Windows, он не является его частью на платформе Macintosh. LiveConnect является частью браузера Netscape версии 4.7 и более ранних, но не является частью Netscape 6 как для Macintosh, так и для Windows. Из-за этого существует несколько вариаций, в которых команда `fscommand` работать не будет. К ним относятся версии Netscape для Windows и большинство браузеров для Macintosh. Можно предположить, что любые другие браузеры, такие как iCab или Opera, также не поддерживают взаимодействие Flash и JavaScript. Только по этой причине большинство разработчиков избегают взаимодействия Flash и JavaScript.

На заметку

Если вы решили использовать команду `fscommand`, разместите предупреждение для пользователей Macintosh и Netscape, чтобы они знали, что в их браузерах фильм воспроизводиться корректно не будет.

ИЗМЕНЕНИЕ ДЕСКРИПТОРА OBJECT/EMBED

Чтобы Flash и JavaScript могли взаимодействовать, сначала для соответствующей **настройки** Flash-фильма необходимо изменить дескриптор OBJECT/EMBED на Web-странице. С этой целью лучше всего использовать диалоговое окно Publish Settings (Параметры публикации), открывающееся посредством одноименной команды меню File (Файл). В диалоговом окне следует открыть вкладку HTML и из раскрывающегося меню Template (Шаблон) выбрать опцию Flash with **FSCommand**. При выборе этой опции нужная информация автоматически размещается в дескрипторе OBJECT/EMBED.

Совет

Подробная информация о параметрах публикации приведена в главе 30 "Оптимизация, публикация и экспортирование фильмов".

Однако разработчики должны точно знать, какие данные следует добавлять в дескрипторы, поскольку им необходимо размещать Flash-фильм на собственных страницах, а не в шаблонах Macromedia.

И в дескриптор OBJECT, и в дескриптор EMBED необходимо включить параметр ID. Этот параметр представляет собой просто основную часть каждого дескриптора. Например, если речь идет о фильме `communicationtests fla`, то сжатый Flash-файл будет `communicationtests.swf`. Следовательно, параметр ID должен быть задан равным `communicationtests`. Ниже приведен полный пример дескриптора OBJECT/EMBED с параметрами ID.

```
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/
swflash.cab#version=6,0,0,0"
ID="communicationtests" WIDTH="550" HEIGHT="400" ALIGN="">
<PARAM NAME=movie VALUE="communicationtests.swf">
<PARAM NAME=quality VALUE=high>
<PARAM NAME=bgcolor VALUE=#FFFFFF>
```

```
<EMBED src="communicationtests.swf" quality=high bgcolor=#FFFFFF
WIDTH="550" HEIGHT="400" swLiveConnect=true
ID="communicationtests" NAME="communicationtests" ALIGN=""
TYPE="application/x-shockwave-flash"
PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer">
</EMBED>
</OBJECT>
```

Добавление параметров ID — не единственное изменение, вносимое в дескрипторы. В дескриптор EMBED необходимо еще добавить предложение `swLiveConnect=true`, которое также приведено в данном примере.

ДОБАВЛЕНИЕ JAVASCRIPT

Изменение дескриптора OBJECT/EMBED является только первым шагом. На страницу необходимо также поместить определенный базовый код JavaScript, чтобы она могла принимать сообщения из Flash-фильма. Такие блоки JavaScript служат в качестве "мостика" между Flash и JavaScript.

Все Flash-сообщения поступают в функцию JavaScript, которая называется `DoFSCCommand`. Однако точное имя этой функции зависит от имени фильма. Например, если фильм называется `communicationtests`, то функция JavaScript должна называться `communicationtests_DoFSCCommand`.

Большинство базовых сценариев подобно приведенному ниже.

```
<SCRIPT LANGUAGE=JavaScript>
function communicationtests_DoFSCCommand(command, args) {
    // Здесь указывается код
}
</SCRIPT>
```

Однако такой сценарий будет работать только в браузерах версий Netscape 3.x и 4.x. В браузерах Internet Explorer он работать не будет, поскольку они не содействуют взаимодействию Flash и JavaScript. Но Flash может взаимодействовать с VBScript — собственным языком сценариев Internet Explorer. В свою очередь, VBScript может взаимодействовать с JavaScript. Следовательно, необходимо настроить систему, в которой сообщение передавалось бы из Flash в VBScript, а из VBScript — в JavaScript.

Такая система сделает код более сложным. Сначала необходимо проверить, установлен ли в системе пользователя браузер Internet Explorer. Если это так, то в код необходимо добавить функцию VBScript, для чего используется команда `document.write`:

```
if (navigator.appName && navigator.appName.indexOf("Microsoft") != -1 &&
    navigator.userAgent.indexOf("Windows") != -1 &&
    navigator.userAgent.indexOf("Windows 3.1") == -1) {
    document.write('<SCRIPT LANGUAGE=VBScript> \n');
    document.write('on error resume next \n');
    document.write('Sub communicationtests_FSCCommand(ByVal command, ByVal
args)\n'),-
    document.write(' call communicationtests_DoFSCCommand(command,
args)\n');
    document.write('end sub\n');
    document.write('</SCRIPT> \n');
}
```

НАСТРОЙКА JAVASCRIPT для ВЗАИМОДЕЙСТВИЯ с FLASH

Теперь все готово для начала передачи сообщения из Flash в JavaScript. Но чтобы отправить сообщения обратно во Flash, необходимо знать местонахождение адресата. Мы уже ука-

зали ID фильма в дескрипторах OBJECT и EMBED, но, к сожалению, браузеры Netscape и IE трактуют их по-разному.

Браузеры Netscape считают Rash-фильм частью *документа*, поэтому адрес фильма необходимо указывать как `document.communicationtests`. Браузеры Internet Explorer используют только ID, а именно `communicationtests`.

Чтобы облегчить жизнь, для одного из этих обращений в используемом по умолчанию шаблоне HTML Flash задается новая переменная `communicationtestsObj`. Для этого применяется следующий код:

```
var InternetExplorer = navigator.appName.indexOf("Microsoft") != -1;
var communicationtestsObj =
    InternetExplorer ? communicationtests :
    document.communicationtests;
```

Таким образом, функция может возвращать данные в переменную `communicationtestsObj`, независимо от того, какой браузер установлен в системе пользователя. Использование такого синтаксиса JavaScript не является широко распространенной практикой, поэтому применяется и другой способ, который выполняет ту же самую роль, но является более удобным, особенно для пользователей, не имеющих большого опыта работы с JavaScript:

```
if (navigator.appName.indexOf("Microsoft") != -1) {
    var communicationtestsObj = communicationtests;
} else {
    var communicationtestsObj = document.communicationtests;
}
```

ПЕРЕДАЧА СООБЩЕНИЙ ИЗ FLASH В БРОУЗЕР

Для передачи сообщений из фильма в браузер необходимо воспользоваться командой `fscommand`. В качестве примера можно привести следующий сценарий кнопки:

```
on (release) {
    fscommand ("alert", вводимый_текст);
}
```

Функция `fscommand` передает в JavaScript два параметра, первым из которых является команда, а вторым — данные. В приведенном ранее примере функции JavaScript `communicationtests_DoFSCommand` можно увидеть, что эти параметры представлены в виде `command` и `args`.

Ни один из параметров не будет ничего означать для JavaScript до тех пор, пока вы не напишете какой-то код для их обработки. Например, приведенная ниже функция обрабатывает команду `alert` путем создания предупредительного окна JavaScript, содержимым которого является параметр `args`.

```
function communicationtests_DoFSCommand(command, args) {
    if (command == "alert") {
        window.alert(args);
    }
}
```

Естественно, в браузере Internet Explorer этот код будет работать только при наличии кода VBScript, описанного ранее в этой главе.

Пример такого кода приведен в демонстрационном фильме `communicationtests fla`. Весь необходимый код JavaScript содержится в HTML-файле `communicationtests.html`. Запустите фильм на HTML-странице в своем браузере. Заполните поле в верхней части

фильма, а затем щелкните на красной кнопке. Введенный текст будет отправлен в функцию JavaScript, которая затем отобразит пользователям предупредительное окно.

Например, если ввести текст "Hello **World!**", то вы должны увидеть диалоговое окно, которое будет выглядеть приблизительно так, как показано на рис. 26.1.

ПЕРЕДАЧА СООБЩЕНИЙ ИЗ БРОУЗЕРА ВО FLASH

С передачей сообщений из Flash в JavaScript связана только одна команда **ActionScript**, а для передачи сообщений из JavaScript во Flash предназначено множество команд, основной из которых является **SetVariable**. Приведенная ниже функция передает строку с именем браузера в переменную фильма **fromJavaScript**.

```
<SCRIPT LANGUAGE=JavaScript>
function sendToFlash(args){
    window.document.communicationtests.SetVariable("fromJavaScript",
        navigator.appName);
}
</SCRIPT>
```

Результатом выполнения этого сценария является то, что теперь переменная **fromJavaScript** фильма **communication-tests** содержит значение константы JavaScript **navigator.appName**. Для этого требуется, чтобы параметры ID дескрипторов OBJECT/EMBED фильма были установлены в значения **communicationtests**.

Результаты выполнения этого сценария можно увидеть в фильме **communicationtests fla**. На HTML-странице имеется простая кнопка, которая вызывает функцию JavaScript **sendToFlash**. Эта функция, в свою очередь, передает значение **navigator.appName** во Flash-фильм, где оно будет отображено в динамическом текстовом поле, связанном с переменной **fromJavaScript**.

SetVariable является одной из многих функций, которые можно вызвать в JavaScript и которые повлияют на Flash-фильм. Ниже приводится перечень остальных функций.

- **GetVariable(имя_переменной)**. Извлекает значение переменной на корневом уровне фильма.
- **GotoFrame(номер_кадра)**. Перемещает фильм в кадр с указанным номером. См. также **TGotoFrame**.
- **IsPlaying()**. Возвращает значение **true**, если фильм воспроизводится в данный момент.
- **LoadMovie(номер_слоя, URL_фильма)**. Загружает новый фильм в указанный слой.
- **Pan(x, y, mode)**. После того как фильм промасштабирован (См. **Zoom**), с помощью значений **x** и **y** этой команды можно выполнить панорамирование. Если аргумент **mode** установлен в значение 0, то **x** и **y** являются пикселями. Если аргумент **mode** установлен в значение 1, то **x** и **y** являются процентными величинами.
- **PercentLoaded()**. Возвращает значения от 0 до 100, сообщая JavaScript о том, какая часть фильма загрузилась.
- **Play()**. Аналогична команде **play** на корневом уровне Flash-фильма. См. также **TPlay**.
- **Rewind()**. Отправляет корневой уровень фильма обратно к кадру 1.

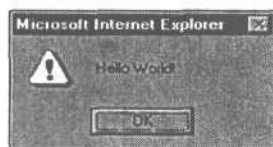


Рис. 26.1. Данное предупредительное диалоговое окно было создано путем взаимодействия Flash-фильма с браузером Internet Explorer версии 6.0 для Windows

- **SetVariable(имя_переменной, значение).** Устанавливает переменную корневого уровня фильма в значение.
- **SetZoomRect(left, right, top, bottom).** Альтернатива функции Zoom. Масштабирует фильм так, что его прямоугольный контур заполняет текущую область фильма.
- **StopPlay().** Аналогична команде stop на корневом уровне Flash-фильма. См. также TStopPlay.
- **TCallFrame(цель, номер_кадра).** Исполняет весь сценарий (кроме функций), существующий в указанном кадре. Может выполняться на корневом уровне либо в видеоклипе. См. также TCallLabel.
- **TCallLabel(цель, надпись_кадра).** Аналогична функции TCallFrame, но вместо номера кадра используется его имя.
- **TCurrentFrame(цель).** Возвращает номер текущего кадра видеоклипа.
- **TCurrentLabel(цель).** Возвращает имя текущего кадра видеоклипа.
- **TGetProperty(цель, символ_свойства).** Возвращает значение свойства видеоклипа. Это значение будет представлено в виде строки. См. список свойств в табл. 26.1, а также TGetPropertyAsNumber.
- **TGetPropertyAsNumber(цель, символ_свойства).** Аналогична функции TGetProperty, но возвращает числовое значение.
- **TGotoFrame(цель, номер_кадра).** Передает видеоклип в кадр с определенным номером. См. также TGotoLabel.
- **TGotoLabel(цель, надпись_кадра).** Передает видеоклип в кадр с определенным именем. См. также TGotoFrame.
- **TPlay(цель).** Аналогична команде play для видеоклипа. См. также Play.
- **TSetProperty(цель, символ_свойства, значение).** Задает свойство в видеоклипе. См. список свойств в табл. 26.1, а также TGetPropertyAsNumber.
- **TStopPlay(цель).** Аналогична команде stop в **ActionScript**, относящейся к видеоклипу. См. также StopPlay.
- **TotalFrames().** Возвращает общее количество кадров фильма.
- **Zoom(процентная_величина).** Масштабирует фильм на указанную величину. См. также SetZoomRect и Pan.

Некоторые из этих команд являются функциями, использующими параметр целевого объекта. Такие команды и функции не всегда работают на корневом уровне фильма. Однако параметр *цель* позволяет указать видеоклип фильма.

Значение "/" означает корневой уровень фильма. Если вы хотите указать видеоклип, имеющий имя **myMovieClip**, используйте параметр целевого объекта **"/myMovieClip"**. Если вы хотите указать видеоклип, имеющий имя **myOtherClip**, который находится внутри клипа **myMovieClip**, используйте параметр целевого объекта **"/myMovieClip"/myOtherClip"**. Таким образом, функции **myMovie.TPlay("/")** и **myMoviePlay** выполняют одну и ту же задачу.

И функция **TGetProperty**, и команда **TSetProperty** в качестве параметра принимают *символ_свойства*. В этом случае было бы неплохо использовать синтаксис **ActionScript**, подобный **_x** и **_alpha**, но, к сожалению, это невозможно. Вместо этого необходимо использовать специальный набор символов, отображающих свойства **ActionScript**. Их полный перечень приведен в табл. 26.1.

ТАБЛИЦА 26.1. ПОЛУЧЕНИЕ свойств ACTIONSCRIPT с помощью JAVASCRIPT

Свойство ACTIONSCRIPT	Символ JAVASCRIPT
<code>_x</code>	<code>X_POS</code>
<code>_y</code>	<code>Y_POS</code>
<code>_xscale</code>	<code>X_SCALE</code>
<code>_yscale</code>	<code>Y_SCALE</code>
<code>_currentFrame</code>	<code>CURRENT_FRAME</code>
<code>_totalFrames</code>	<code>TOTAL_FRAMES</code>
<code>_alpha</code>	<code>ALPHA</code>
<code>_visible</code>	<code>VISIBLE</code>
<code>_width</code>	<code>WIDTH</code>
<code>_height</code>	<code>HEIGHT</code>
<code>_rotation</code>	<code>ROTATE</code>
<code>_framesLoaded</code>	<code>FRAMES_LOADED</code>
<code>_name</code>	<code>NAME</code>
<code>_dropTarget</code>	<code>DROP_TARGET</code>
<code>_url</code>	<code>URL</code>

Помимо всех этих команд и функций, Flash-фильм автоматически инициирует два события. Событие `OnProgress` отправляет процентное значение загруженного фильма в JavaScript в процессе его загрузки. Подобным образом событие `OnReadyStateChange` принимает значения от 0 до 4, представляющие соответственно состояния загружающегося, инициализированного, загруженного, интерактивного и завершенного фильма.

КРУГОВЫЕ СООБЩЕНИЯ

Очень часто возникает необходимость запросить у браузера данные из Flash-фильма. Такая операция проходит в два этапа. Сначала фильм отправляет сообщение в JavaScript. Это сообщение затем инициирует JavaScript для передачи данных обратно в фильм.

Предположим, необходимо, чтобы браузер отправил в фильм свое имя. Начнем с применения команды `fscommand` в фильме. Эту операцию выполнит код, подобный приведенному ниже.

```
fscommand ("sendMeInfo", "appName");
```

Далее на Web-странице понадобятся все ранее упоминавшиеся стандартные параметры: ID, параметр `swLiveConnect`, функция JavaScript `myMovie_DoFSCommand` и "мостик" VBScript.

Приведенная ниже функция принимает сообщение из фильма и передает его обратно.

```
function myMovie_DoFSCommand(command, args) {
    var myMovieObj = InternetExplorer ? myMovie : document.myMovie;

    if (command == "sendMeInfo") {
        if (args == "appName") {
            myMovieObj.SetVariable("returnValue", navigator.appName);
        }
    }
}
```

Когда фильм передает на страницу команду `fscommand`, страница исполняет предыдущую функцию. Она проверяет переменные `command` и `args`. Если этими переменными являются `sendMeInfo` и `appName`, то свойство JavaScript `navigator.appName` передается в фильм, где оно появляется в виде значения переменной `returnValue`.

АЛЬТЕРНАТИВНЫЕ МЕТОДИКИ

Еще один способ взаимосвязи Flash и браузера основывается на том факте, что большинство браузеров воспринимают команды JavaScript, указанные в их адресной строке, расположенной сверху окна. Этим фактом можно воспользоваться для того, чтобы обойти упоминавшиеся ранее ограничения, которые имеют системы Macintosh и браузеры Netscape при реализации взаимосвязи с JavaScript.

Например, можно открыть браузер и ввести в адресной строке `javascript:window.alert('Hello!');`. При нажатии клавиши `<Return>` (Macintosh) или `<Enter>` (Windows) должен быть выполнен сценарий JavaScript, указанный после двоеточия. В результате будет получено предупредительное окно.

Когда Flash передает команду `getURL`, на самом деле выполняется передача строки в адресное поле браузера. Поэтому введя `javascript:` и какой-то сценарий JavaScript, вы сможете инициализировать его выполнение.

Вы можете выполнять команды JavaScript или даже вызывать функции JavaScript, которые были определены на странице. Эта методика применима к большинству браузеров, в том числе и к последним версиям Netscape для Windows и Internet Explorer для Macintosh. Вы сможете быстро протестировать любой браузер, введя в его адресную строку команду `javascript:.`

Несмотря на то что эта методика хороша для отправления сообщений из Flash в браузер, она не поможет, если необходимо отправить сообщения из браузера во Flash. Кроме того, она имеет и другие недостатки, например, частая замена содержимого текущего окна браузера результатами вызываемой функции JavaScript.

ЗАДАНИЕ ВОПРОСОВ с помощью JAVASCRIPT

JavaScript позволяет легко выполнять простые задачи, связанные с вводом данных. Например, с помощью всего одной команды можно быстро задать вопрос. Посредством команды JavaScript `confirm` вы получите небольшое диалоговое окно с кнопками **ОК** и **Cancel** (Отмена). После щелчка на кнопке **ОК** возвращается значение `true`.

Доступ к этой функции JavaScript можно получить с помощью команды `fscommand`. Например, эта команда используется в следующем сценарии кнопки:

```
on (release) {  
    fscommand("askYesOrNo", "Вы действительно хотите перейти к следующему  
        кадру?");  
}
```

Затем на Web-странице можно поместить приведенный ниже сценарий. Помните о включении всех других необходимых элементов, таких как `ID`, `swLiveConnect` и мостика VBScript. Приведенный ниже код подразумевает, что идентификатором фильма является `askquestion.swf`.

```
function askquestion_DoFSCommand(command, args) {  
    var askquestionObj = InternetExplorer ? askquestion :  
document.askquestion;  
    if (command == "askYesOrNo") {  
        if (confirm(args)) {  
            askquestionObj.play();  
        }  
    }  
}
```

На рис. 26.2 изображено созданное диалоговое окно подтверждения. В различных версиях браузера его внешний вид может слегка варьироваться.

Чтобы задать вопрос, представляющий собой короткую строку, можно также воспользоваться командой JavaScript prompt.

СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ОКОН

При взаимодействии Flash и JavaScript чаще всего необходимо открывать новые окна браузера. Хотя окно можно открыть и с помощью команды `getURL`, это не позволяет управлять размером и внешним видом окна. Для этого понадобится JavaScript.

Демонстрационный фильм `openwindow fla` позволяет протестировать открытие пользовательских окон. Вы сможете изменять буквально любой параметр окна, а для его открытия использовать кнопки, расположенные в нижней части экрана.

В коде **ActionScript** нет ничего особенного. Он объединяет в себе различные настройки окна в виде одной "функциональной" переменной. Для исполнения JavaScript необходимо указать URL, имя окна и перечень функций.

Код ActionScript отправляет эти значения в JavaScript в виде одной строки, содержащей URL, имя и функции, разделенные точкой с запятой. Поскольку между Flash и JavaScript может передаваться только один аргумент, три параметра объединяются в одну строку, как показано в приведенном ниже примере.

```
on (release) {  
    features = "left="+x;  
    features += ", screenX="+x;  
    features += ", top="+y;  
    features += ", screenY="+y;  
    features += ", width="+width;  
    features += ", height="+height;  
    features += ", toolbar="+toolbar;  
    features += ", resizable="+resizable;  
    features += ", directories="+directories;  
    features += ", status="+status;  
    features += ", location="+location;  
    features += ", menubar="+menubar;  
    features += ", scrollbars="+scrollbars;  
  
    fscommand("openwindow", url+"; "+name+"; "+features);  
}
```

Команда `window.open`

Большинство функциональных параметров команды `window.open` отображают или скрывают элементы, такие как адресная строка, панель инструментов, полосы прокрутки и т.п. Браузеры Netscape требуют указания параметров `screenX` и `screenY`, а браузеры Microsoft — параметров `left` и `top`. Более подробная информация о команде `window.open` имеется в справочных материалах по JavaScript, в частности на Web-узлах.

С другой стороны, JavaScript использует команду `split` для того, чтобы разбить строку на три части. Затем для создания окна из этих частей используется команда `window.open`:

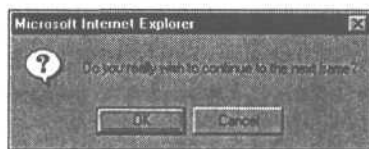


Рис. 26.2. Данное диалоговое окно подтверждения было создано путем взаимодействия Flash-фильма с браузером Internet Explorer версии 6.0 для Windows


```
function openwindow_DoFSCommand(command, args) {
    var openwindowObj = InternetExplorer ? openwindow :
document.openwindow;
    if (command == "openwindow") {
        var argsArray = args.split(";");
        window.open(argsArray[0], argsArray[1], argsArray[2]);
    }
}
```

Другой способ заключается в том, что мы забываем о стандартной взаимосвязи ActionScript и JavaScript и используем команду `javascript:` в сочетании с `getURL`. От Web-страницы вам ничего не нужно. Просто измените последнюю строку сценария кнопки с `fscommand` на приведенный ниже код.

```
getURL("javascript: var myWindow = window.open('"+url+"', '"+name+"',
'"+features+"');");
```

Необходимо, чтобы строка выглядела так, как при задании переменной. В противном случае результат команды `window.open` будет помещен в текущее окно браузера. Это означает, что Flash-фильм будет заменен на "[Object]".

Если вы склоняетесь к применению более продвинутой методики, то в начале фильма отправьте текстовое сообщение в JavaScript. Затем JavaScript отправит сообщение обратно в фильм, что равноценно установлению переменной `jsCommunication` в значение `true`. Если в системе пользователя установлен браузер, поддерживающий взаимосвязь ActionScript и JavaScript, то переменная `jsCommunication` дает значение `true`, а это означает, что вы можете отправлять и получать сообщения. Если этого не происходит, воспользуйтесь альтернативной методикой. Например, если вы хотите **открыть** окно браузера, можно воспользоваться командой `javascript:` либо открыть обычное окно, указав целевой объект в качестве параметра команды `getURL`.

УПРАВЛЕНИЕ ПРОЕКТОРОМ

Команда `fscommand` функционирует совершенно по-другому, если она выполняется внутри Flash-проектора. В этом случае сообщения передаются в плеер проектора, который воспринимает ограниченный, но полезный набор команд. Например, проектор может самостоятельно изменить свои размеры или завершить сеанс, когда пользователь щелкает на кнопке.

МОДИФИКАЦИЯ ОКНА

При использовании команды `fscommand` в проекторе все равно необходимо указать два параметра: `command` и `arguments`. Однако команды в проекторе являются жестко закодированными, поэтому выбирать их приходится из ограниченного набора. Команда `fullscreen` увеличивает проектор до полного размера монитора пользователя. Она растягивает содержимое Flash-фильма так, чтобы оно соответствовало новому размеру окна. В результате фильм будет занимать целый экран так, что не останется даже места для границ окна или строки заголовка.

Чтобы увеличить проектор до размеров полного экрана, воспользуйтесь следующей командой:

```
fscommand("fullscreen", true);
```

Чтобы вернуться к исходному размеру проектора, воспользуйтесь командой

```
fscommand("fullscreen", false);
```

Кроме того, можно указать, будет ли масштабироваться фильм внутри окна. Для отключения масштабирования воспользуйтесь такой командой:

```
fscommand("allowscale", false);
```

После этого, когда проектор займет весь экран, фильм будет оставаться в исходных размерах, располагаясь по центру окна. Это относится и к случаю, когда пользователь перетаскивает окно за угол в целях его расширения или сжатия. Чтобы снова включить масштабирование, воспользуйтесь следующей командой:

```
fscommand("allowscale", true);
```

СОКРЫТИЕ КОНТЕКСТНОГО МЕНЮ

Чтобы открыть контекстное меню в проекторе, пользователь может щелкнуть мышью, удерживая клавишу <Cmd> (Macintosh), либо щелкнуть правой кнопкой мыши (Windows). Это меню позволяет пользователю приостановить фильм или изменить его громкость. Чтобы отключить данную функцию в проекторе, воспользуйтесь следующей командой:

```
fscommand("showmenu", false);
```

Если вы не хотите, чтобы пользователь мог регулировать различные параметры фильма, данное меню следует отключить. Чтобы вернуться к его отображению, воспользуйтесь командой

```
fscommand("showmenu", true);
```

Обратите внимание на то, что отключение данной функции не приводит к полному отключению контекстного меню. Пользователь все равно сможет иметь доступ к категориям Settings (Настройки) и About (О программе), но он не сможет управлять фильмом.

Если вы хотите отключить данные контекстные меню при воспроизведении фильма в браузере, снимите флажок с поля опции Display Menu (Отображать меню), расположенный во вкладке HTML диалогового окна Publish Settings. При этом в дескрипторы OBJECT и EMBED будет вставлен параметр menu со значением false. Эти дескрипторы на HTML-страницу можно вставить и вручную.

ДРУГИЕ КОМАНДЫ

Команда quit предназначена для выхода из проектора. Второй параметр в команде fscommand обязателен для указания, несмотря на то, что фактически он не нужен. Его можно ввести следующим образом:

```
fscommand("quit", " ");
```

С помощью функции fscommand можно запускать внешние приложения. Для этого необходимо подать команду "exec" и указать путь к приложению:

```
fscommand("exec", "C:\notepad.exe ");
```

Пользуйтесь данной командой с осторожностью. На момент написания этой книги я не смог заставить ее работать на своем компьютере. Может быть, вам повезет больше.

ВЗАИМОСВЯЗЬ с ПРОГРАММОЙ DIRECTOR

Одной из наиболее сильных сторон Flash является возможность внедрения фильмов в приложение Macromedia Director в качестве элементов изобразительного ряда. Можно также сказать, что эта же возможность является одной из самых сильных сторон приложения Director.

Когда Rash-фильм внедрен в Director, можно передавать сообщения между **ActionScript** и Lingo, языком программирования внутри Director.

В приложение Macromedia Director 8.5 могут быть внедрены фильмы Flash 5, что дает возможность реализации всех способов взаимодействия, о которых пойдет речь в этом разделе. На момент чтения вами этой книги Director может быть обновлен и поддерживать Flash MX, благодаря чему могут появиться новые способы взаимосвязи.

ВЗАИМОСВЯЗЬ FLASH—DIRECTOR

Отправлять сообщения в Lingo можно двумя способами, причем в обоих используется команда `getURL`. В первом случае сообщения передаются в предварительно заданный обработчик Lingo, являющийся эквивалентом **Flash-функций** приложения Director. Сообщение передается посредством приведенного ниже сценария кнопки.

```
on (release) {  
    getURL("lingo: this is a test");  
}
```

В приложении Director в спрайт, содержащий Flash-фильм, необходимо поместить сценарий, который иногда называют *поведением*. В нем должен содержаться обработчик `on getURL`, как показано в приведенном ниже примере.

```
on getURL me, stringFromFlash  
    alertstringFromFlash  
end
```

Параметр `me` является используемым в Lingo средством связи обработчика с объектом спрайта. Вторым параметром является строка, передаваемая из Flash; в данном случае это все, что стоит после команды `lingo:.`

Передавать можно любую нужную строку, например имя кадра, к которому должно перейти приложение Director, или значение, в которое следует установить текстовое поле. В результате обработчик Lingo перейдет к кадру, имя которого указывается при обращении Flash:

```
on getURL me, frameToJumpTo  
    go to frame frameToJumpTo  
end
```

Еще одним, более прогрессивным способом обращения к Director из Flash является использование команды `event:`, а не `lingo:.` Это позволяет направить сообщение в определенный обработчик, присоединенный к спрайту. Например, приведенный ниже код во Flash-фильме вызовет обработчик `myHandler`.

```
getURL("event: myHandler");
```

Код со стороны Director будет следующим:

```
on myHandler me  
    -- сделать что-нибудь  
end
```

Если вы хотите передать параметры в обработчик Lingo, добавьте их в строку `getURL`:

```
getURL("event: myHandler 7");
```

Если вы хотите использовать кавычки, поставьте перед ними символы косой черты, чтобы **ActionScript** не интерпретировал кавычки, а передавал их вместе со строкой:

```
getURL("event: myHandler \"jump\", \"frame name\");
```

Посредством команды `event:` можно настроить помещенное в спрайт поведение так, чтобы оно имело несколько разных функций, каждая из которых определялась бы собственным обработчиком.

ВЗАИМОСВЯЗЬ DIRECTOR – FLASH

Процедура передачи данных из Lingo во flash-фильм, внедренный внутрь изобразительного ряда, аналогична передаче данных с Web-страницы. При этом задействуется похожий набор команд.

В данном подразделе рассматриваются команды, использующиеся для того, чтобы определить, насколько Flash-фильм готов к воспроизведению. Когда объект Flash внедряется в фильм Director, вероятно, что он сразу же будет доступен. Однако Director можно использовать и для создания объекта Flash, связанного с внешним Flash-файлом. В этом случае Flash-фильм будет представлять собой поток данных, такой же, как и на Web-странице.

Чтобы определить, готов ли Flash-фильм к воспроизведению, воспользуйтесь свойством `percentstreamed`:

```
if member("my flash movie").percentStreamed = 100 then
```

Чтобы определить, был ли загружен определенный кадр, воспользуйтесь функцией `frameReady()`. Свойство `frameCount` подскажет, сколько всего кадров содержится в фильме.

Чтобы подробнее узнать о том, что происходит при загрузке объекта Flash, можно воспользоваться свойством `state`. Ниже приводится перечень его возможных значений.

- 0 — не загружен;
- 1 — загружается заголовок;
- 2 — загрузка заголовка завершена;
- 3 — загружается мультимедиа;
- 4 — загрузка мультимедиа завершена;
- -1 — ошибка.

После загрузки фильма можно воспользоваться элементом `frameRate` для получения значения частоты смены кадров. На панели инспектора свойств программы Director можно выбрать объекты и спрайты Flash и изменить их частоту кадров так, чтобы она совпадала с частотой смены кадров фильма Director. За более подробной информацией по этому вопросу обратитесь к документации Director.

Ускорение показа графического изображения Flash

Многие разработчики, работающие в программе Director, используют Flash-фильмы для замены растровых изображений. В этом случае Flash-фильм представляет собой всего лишь статичное изображение, а не анимацию. Поскольку содержимое Flash-ряда не будет обновляться в каждом кадре, воспроизведение фильма Director можно ускорить. Для этого следует установить свойство `static` элемента изобразительного ряда или спрайта в значение `false`.

Код Lingo для управления Flash-фильмом можно использовать несколькими способами. Команды `play()` и `stop()` применяются к спрайту и выполняют ту же самую роль, что и команды `play` и `stop` во Flash. Приведенный ниже сценарий кнопки инициирует воспроизведение Flash-фильма в спрайте 7.

```
on mouseUp me  
    sprite(7).play()  
end
```

Для перемещения Flash обратно в кадр 1 можно воспользоваться командой `rewind()`. Чтобы выяснить, какой кадр фильма является активным в данный момент, воспользуйтесь свойством `frame`. Чтобы преобразовать любой номер кадра в имя, воспользуйтесь функцией

`getFrameLabel()`. Функцией, выполняющей обратное преобразование, является `findLabel()`. Она возвращает номер кадра, содержащий указанное имя. Чтобы определить, воспроизводится ли фильм в данный момент, используется свойство `playing`.

Для управления воспроизведением фильма предназначена функция `goToFrame()`, которая переносит Flash-фильм к кадру с указанным именем или номером. Кроме того, свойство `frame` спрайта можно установить в значение номера кадра.

Если необходимо выяснить значение свойства Flash-фильма, воспользуйтесь функцией `getProperty()`. Эта функция в качестве аргументов принимает и целевой объект, и специальный символ свойства. Например, чтобы выяснить значение свойства `_x` видеоклипа `myMovieClip`, можно применить код

```
x = sprite(7).getFlashProperty("myMovieClip", #posX)
```

В табл. 26.1 перечислены специальные имена каждого основного свойства видеоклипа. К сожалению, в Lingo приходится использовать другой набор имен свойств. Из следующей строки станет достаточно очевидно, с каким свойством связан каждый из символов Lingo: `#posX`, `#posY`, `ttscaleX`, `ttscaleY`, `#visible`, `ttrotate`, `ttalpha`, `tfname`, `#width`, `tfheight`, `#target`, `#url`, `#dropTarget`, `tttotalFrames`, `ttcurrentFrame`, и `#lastframeLoaded`.

Для изменения любого свойства видеоклипа можно воспользоваться функцией `setFlashProperty()`. Например, для задания свойства `_alpha` видеоклипа можно применить следующий код:

```
sprite(7).setFlashProperty("myMovieClip", #alpha, 50)
```

С помощью Lingo можно получать и задавать собственные переменные, но данная методика применима только к переменным корневого уровня. Так, для получения переменной можно использовать такую команду:

```
sprite(7).getVariable("myVariable")
```

Для задания переменной на корневом уровне можно воспользоваться командой

```
sprite(7).setVariable("myVariable", 42)
```

Самой мощной во взаимосвязи Director–Flash является команда `callFrame`. Она исполняет код **ActionScript** в кадре Flash-фильма на корневом уровне. Например, посредством приведенной ниже строки можно выполнить сценарий **ActionScript** в кадре 5 Flash-фильма.

```
sprite(7).callFrame(5)
```

Любые команды в кадре будут выполняться, как если бы Flash-фильм только что достиг этого кадра. Таким образом, функции не будут выполняться, если в кадре они не будут специально вызваны.

Тем, кто работал с Hash 4, знакома эта функциональная возможность. Дело в том, что во Hash 4 нет функций. Вместо этого код помещался в различные кадры Hash-фильма, а для его исполнения в кадрах использовалась команда `callFrame`.

Комбинируя команды `setVariable` и `call`, можно инициировать исполнение любого элемента Hash-фильма. Например, если вы хотите вызвать функцию **ActionScript** с параметрами `hello` и `79`, воспользуйтесь командой `setVariable`, чтобы установить переменные `param1` и `param2` в значения `hello` и `79`. После этого для обращения к сценарию в кадре X

используется команда `callFrame`. Данный кадр просто примет переменные `param1` и `param2` и передаст их в функцию, которую необходимо выполнить:

```
myFunction(param1,param2);
```

Кроме того, Lingo может инициировать функции `ActionScript print` и `printAsBitmap` посредством передачи сообщения с этими именами во Flash-спрайт. Эти функции работают точно так же, как и при реализации функции печати в `ActionScript`:

```
sprite(7).print("myMovieClip",#bframe)
```

Еще одной интересной функциональной возможностью программы Director является способность перенаправлять любые команды передачи сообщений Flash на определенный видеоклип. Для этого сначала подается команда `tellTarget`. Кроме того, последовательность сообщений необходимо завершить командой `endTellTarget`. Таким образом, для вызова сценария `ActionScript` в кадре 5 видеоклипа `myMovieClip` можно применить такой код:

```
sprite(7).tellTarget("myMovieClip")
sprite(7).setVariable("myVariable",99)
sprite(7).callFrame(5)
sprite(7).endTellTarget()
```

Чтобы заставить Flash-фильм перейти к какому-либо кадру, применяется код

```
sprite(7).tellTarget("myMovieClip")
sprite(7).goToFrame(10)
sprite(7).endTellTarget()
```

На заметку

Ранее уже упоминалось, что версия Director 8.5.1 поддерживает только фильмы Flash 5. Однако после выхода новой версии программы, способной работать с фильмами Flash MX, вероятно, появится и возможность непосредственного обращения к функциям.

КОМАНДЫ LINGO

Перечисленные ниже команды Lingo позволяют передавать сообщения или получать данные об элементе Flash-фильма. Здесь пропущены команды, которые влияют на внешний вид спрайта или элемента. Программисты Lingo могут легко найти их сами на панели инспектора свойств программы Director.

- `endTellTarget()`. См. `tellTarget`.
- `findlabel(имя_кадра)`. Данная функция возвращает номер кадра, соответствующий имени данного кадра.
- `frame`. Это свойство представляет номер текущего кадра Flash-фильма. Его можно и проверять, и задавать.
- `frameCount`. Данное свойство возвращает общее число кадров на временной шкале Flash.
- `f rameRate`. Это свойство возвращает частоту смены кадров Flash-фильма.
- `frameReady(номер_кадра)`. Данная функция возвращает значение `true`, если указанный кадр Flash-фильма уже был загружен.
- `getFlashProperty(цель, свойство)`. Данная функция извлекает значение свойства видеоклипа. Для указания свойства используются следующие символы: `ttxsx`, `#posY`, `ttscaleX`, `#scaleY`, `frvisible`, `#rotate`, `ftalpha`, `#name`, `ftwidth`, `ttheight`,

fttarget, #url, ttdropTarget, #totalFrames, ttcurentFrame и #lastframeLoaded.

- `getFrameLabel(номер_кадра)`. Данная функция возвращает имя кадра.
- `getVariable(имя_переменной)`. Данная функция **возвращает** значение переменной текущего уровня, как правило, корневого. См. также `tellTarget`.
- `goToFrame(кадр)`. Данная функция переносит текущую временную шкалу в кадр с указанным именем или номером. См. также `tellTarget`.
- `percentStreamed`. Это свойство возвращает значение в диапазоне от 0 до 100, указывающее, какая часть фильма готова к воспроизведению.
- `play()`. Данная команда начинает воспроизведение Flash-фильма. См. также `tellTarget`.
- `playing`. Это свойство возвращает значение `true`, если Flash-фильм воспроизводится.
- `rewind()`. Данная команда возвращает Flash-фильм к кадру 1.
- `setFlashProperty(цель, свойство, значение)`. Данная команда задает значение свойства видеоклипа. Символы, используемые для использования в качестве свойств, приведены в описании команды `getFlashProperty`.
- `setVariable(имя_переменной, значение)`. Данная команда задает значение переменной на текущем уровне. См. также `tellTarget`.
- `state`. Это свойство возвращает значение от 0 до 4 или -1. См. перечень, приведенный в подразделе "Взаимосвязь Director-Flash".
- `stop()`. Данная команда останавливает воспроизведение в текущей временной шкале. См. также `tellTarget`.
- `tellTarget(цель)`. Данная команда направляет все последующие обращения Flash на определенный видеоклип фильма. Она особенно полезна для реализации команд `play`, `stop`, `goToFrame`, `setVariable` и `getVariable`. Чтобы снова направить обращения на корневой уровень, воспользуйтесь командой `endTellTarget`.

ХРАНИЕНИЕ ИНФОРМАЦИИ ЛОКАЛЬНО

Одной из методик, чаще всего обсуждающихся разработчиками **ActionScript**, работающих во Flash 4 и Flash 5, была реализация взаимосвязи с JavaScript с целью задания cookie-файлов на машине пользователя. Данная методика позволяла фильмам хранить такую информацию, как имя пользователя или максимальный счет, набранный в процессе игры. Однако в связи с применением JavaScript эта методика работала не во всех браузерах.

Во Flash MX имеется встроенное средство хранения небольших блоков данных на машине пользователя. Эта функция, именуемая *локальным коллективным объектом*, аналогична cookie в JavaScript, но она является полностью встроенной во Flash.

Операция по созданию локального коллективного объекта занимает несколько строк кода и является относительно простой. Сначала необходимо создать объект. Затем его свойство устанавливается в нужное значение. И наконец, необходимо заставить Flash сохранить значение на жестком диске компьютера пользователя. Ниже приведен пример кода.

```
mySO = SharedObject.getLocal("mySharedObject");  
mySO.data.username = "Gary";  
mySO.flush();
```

Первая строка кода выполняет одну из двух функций. В первую очередь производится попытка получить доступ к коллективному объекту `mySharedObject` на локальном диске компьютера пользователя. Если объект находится на жестком диске, то код устанавливает `myso` как ссылку на данный объект. Если его на жестком диске нет, код создает новый объект.

Каждый коллективный объект имеет важное свойство `data`. Это — место, в котором хранится информация. Для создания нового свойства `data` просто присвойте ему значение, как сделано во второй строке кода.

Последняя строка подает команду `flush`, которая необходима для того, чтобы сообщить Flash об изменении коллективного объекта и о необходимости обновления файла на жестком диске компьютера пользователя.

При первом использовании команды `flush` применительно к коллективному объекту пользователь увидит диалоговое окно Macromedia Flash Player Settings (Параметры Macromedia Flash Player), показанное на рис. 26.3, в котором запрашивается разрешение на хранение данных на жестком диске.

Для внедрения этой функции никаких операций с кодом выполнять не нужно. Отключить эту функцию тоже нельзя. Пользователь может отклонить запрос или установить предельный объем данных. Если данные не были записаны по какой-то из этих причин, команда `flush` вернет значение `false`. Если пользователя запрашивают о разрешении, вы получите результат `pending`.

Можно указать минимальный размер файла, который команда `flush` зарезервирует для коллективного объекта в процессе ожидания решения пользователя. Если размер объекта превысит этот объем, диалоговое окно откроется снова и запросит разрешение пользователя на хранение большего количества данных.

Получить данные из коллективного объекта так же просто, как и задать объект. Если для своего фильма вы еще не назначили коллективный объект глобальным, то начать нужно с этого шага. Затем доступ к свойству можно получить следующим образом:

```
mySO = SharedObject.getLocal("mySharedObject");  
thisUsername = mySO.data.username;
```

Если коллективный объект еще не существует или если отсутствует свойство `username`, мы получим значение `undefined`.

Каждый раз при создании нового коллективного объекта он располагается так, что доступ к нему могут получить только фильмы с одного и того же Web-узла. По умолчанию доступ к коллективному объекту могут получить только запросы из одного и того же фильма. Однако, указав полный путь к коллективному объекту, вы добьетесь того, что доступ к нему смогут получить и другие фильмы с того же узла.

Например, если Flash-фильм расположен на узле `http://www.mysite.com/cool/stuff`, то путь к коллективному объекту на жестком диске пользователя будет `/cool/stuff`. Доступ к объекту можно получить с помощью следующего кода:

```
mySO = SharedObject.getLocal("mySharedObject",  
"/cool/stuff/mymovie.swf/");
```

Эта строка кода позволяет фильмам коллективно использовать информацию в пределах всего сеанса пользователя или при повторном сеансе.



Рис. 26.3. Диалоговое окно Macromedia Flash Player Settings открывается внутри Flash-фильма при первой попытке сохранения коллективного объекта

ВЗАИМОСВЯЗЬ FLASH-ФИЛЬМОВ

Еще одним способом взаимодействия, о котором вам следует знать, является взаимосвязь двух фильмов в одном и том же браузере.

На заметку

Взаимосвязь двух фильмов может быть реализована только в случае, если фильмы находятся на одном и том же узле.

Для реализации такого взаимодействия сначала необходимо настроить принимающий фильм для приема входящих сообщений. Эта операция выполняется путем задания нового объекта локального соединения. Затем для подготовки к приему входящих сообщений используется команда `connect`:

```
myLC = new LocalConnection();  
myLC.connect("myconnection");
```

Далее необходимо создать функции для обработки входящих сообщений:

```
myLC.onMessage = function(myString) {  
    // сделать что-то с myString  
}
```

Фильм, отправляющий сообщение, создаст объект локального соединения и будет использовать команду `send` для передачи сообщений. Первым параметром является имя соединения принимающего фильма. Вторым параметром является имя сообщения, соответствующее вызываемой функции. Последним параметром являются отправляемые данные.

```
myLC = new LocalConnection();  
myLC.send("myconnection", "onMessage", "Hello World!");
```

Если вы планируете отправить только одно короткое сообщение, то для закрытия соединения можете использовать команду `close`:

```
myLC.close();
```

Внимание!

На момент написания этой книги локальные коллективные объекты в реальных ситуациях использовались разработчиками достаточно редко. Информация о последних разработках и "подводных камнях" содержится в технических примечаниях Macromedia и на Web-узлах разработчиков.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему не работает взаимосвязь ActionScript—JavaScript?

Чтобы такая взаимосвязь заработала, необходимо выполнить несколько действий: задать параметры ID дескрипторов OBJECT и EMBED, параметр `swLiveConnect` дескриптора EMBED, а также включить функцию JavaScript для получения сообщения и функцию VBScript — для передачи сообщения из VBScript в JavaScript. Если хотя бы одно из этих действий пропущено или в нем допущена ошибка, взаимосвязь работать не будет. Кроме того, проверьте, поддерживает ли такую взаимосвязь ваш браузер.

Взаимосвязь ActionScript—JavaScript работает на моем компьютере, но не работает на других машинах.

Вероятно, пользователи работают с браузерами, не поддерживающими такой тип взаимодействия. Кроме того, в их браузерах может быть просто отключена поддержка JavaScript или ActiveX. Некоторые организации в целях безопасности по умолчанию отключают на своих компьютерах эту функцию.

Некоторые пользователи сообщают, что мой локальный коллективный объект у них не работает.

Некоторые компьютеры, особенно в школах и библиотеках, настроены так, что сохранение локальных данных браузерами или Flash не разрешается.

Когда я работаю с Flash-фильмами в программе Director, воспроизведение фильма Director существенно замедляется.

Фильмы Director воспроизводятся со скоростью, почти в 40 раз превышающей скорость Flash-фильмов. Поэтому добавление Flash-фильма в приложение Director действительно может существенно снизить скорость воспроизведения фильма Director. По возможности выполните максимально большую обработку данных в Lingo и как можно меньшую в элементе Flash. Кроме того, не забывайте установить свойство `static` для любого Flash-элемента, не являющегося анимацией.

ВЗАИМОСВЯЗЬ с СЕРВЕРОМ

В ЭТОЙ ГЛАВЕ...

Загрузка текстовых данных	625
Внедрение данных на HTML-страницу	628
Загрузка динамических данных	630
Отправление данных	631
Возможные проблемы	633
Flash за работой: отправление электронной почты из Flash	634

ЗАГРУЗКА ТЕКСТОВЫХ ДАННЫХ

Существует множество способов передачи данных между фильмом Macromedia Flash и сервером, из которых можно выбрать наиболее приемлемый для себя. Фильмы могут считывать простые текстовые файлы или данные с HTML-страницы. Эти данные не обязательно должны поступать из статичного источника. Фильмы также могут загружать обновленные данные из CGI-программ и баз данных. Можно даже пересылать данные обратно на сервер.

Во Flash MX представлен новый способ загрузки данных с сервера. Он называется объектом **LoadVars**. Данный объект занял место использовавшейся ранее функции Loadvars и ее разновидностей.

ЗАГРУЗКА ДАННЫХ с помощью ОБЪЕКТА LOADVARS

Для получения блока данных с сервера необходимо просто создать объект LoadVars, а затем подать команду load следующим образом:

```
var myLoadVars = new LoadVars();
myLoadVars.load("data.txt");
```

В приведенном выше коде в качестве примера загружаемого файла используется `data.txt`. Этот текстовый файл должен иметь специальный формат: *ИМЯ_СВОЙСТВА=значение_свойства* &. Обратите внимание на применение двух специальных знаков: равенства и амперсанда. Знак равенства разделяет имя свойства и его значение. Знак амперсанда стоит после значения свойства. В текстовый файл можно поместить сколько угодно пар свойство-значение. Например:

```
a=7&b=42.8&c=Hello World!&
```

Строку не нужно заключать в кавычки. Кроме того, внутри строк могут быть вставлены символы разрыва строки, которые будут включены в значение свойства, как показано в приведенном ниже примере.

```
a=7&myString=Это проверка  
многострочной переменной,  
переданной во Flash&
```

После загрузки этих данных свойства добавляются в объект `LoadVars`. Так, в приведенном выше примере добавляется свойство со значением 7. Проверить его можно следующим образом:

```
trace(myLoadVars.a);
```

Совет

Подробная информация о строках приведена в главе 12 "Управление переменными, данными и типами данных".

ИСПОЛЬЗОВАНИЕ ОБЪЕКТА LOADVARS

При использовании команды `load` вы не увидите немедленных результатов. Сначала Flash запрашивает файл с сервера, а затем необходимо подождать, пока он загрузится. И только после этого можно прочитать содержащуюся в нем информацию. Весь процесс может занять от нескольких долей секунды до нескольких секунд.

Как узнать, когда завершилось выполнение команды `load`? Для этого существуют два способа.

Можно проверить, дает ли свойство `loaded` объекта `LoadVars` значение `true`, но для этого потребуется неоднократно проверять свойство `loaded`. Вместо этого можно использовать событие `onLoad` объекта `LoadVars` и создать функцию, которая будет выполняться при завершении загрузки. Ниже приведен пример.

```
myLoadVars = new loadVars();  
myLoadVars.onLoad = function() {  
    results = myLoadVars.toString();  
}  
myLoadVars.load("loaddata.txt");
```

Данный сегмент кода, который, к примеру, присоединен к кнопке в имеющемся на прилагаемом компакт-диске демонстрационном файле `load fla`, установит значение переменной `results` равным результату исполнения функции `toString`. Данная функция при использовании с объектом `LoadVars` покажет свойства объекта в виде одной длинной строки. Результат выглядит следующим образом:

```
c=Hello%20World%21&b=42%2E8&a=7&onLoad=%5Btype%20Function%5D
```

Файл `loaddata.txt` содержит строку

```
a=7&b=42.8&c=Hello World!&
```

Здесь порядок переменных обратный. Кроме того, символы преобразованы так, чтобы использовались знаки перехода, похожие на те, что используются при работе с данными Internet. Например, %20 означает *шестнадцатеричное* число 20, равное десятичному числу 32, и является кодом символа пробела.

Типы MIME

Любые даты следует хранить на сервере в виде типа MIME `application/x-www-urlform-encoded`. Типы MIME определяют, какие типы файлов данных содержатся на сервере. Обычный текстовый файл представляется как `text/html`. Flash-фильмы должны корректно обрабатывать оба типа файлов, но в предыдущих версиях программы возникают проблемы с типом `text/html`. При возникновении каких-либо вопросов об этих типах MIME обратитесь к своему Web-мастеру.

Ход загрузки большого файла данных можно отслеживать с помощью `ActionScript`. Объект `LoadVars` имеет свойства `getBytesTotal` и `getBytesLoaded`, которые можно использовать для создания панели отображения хода загрузки. Однако в большинстве случаев файлы данных имеют слишком маленькие размеры, и необходимости отображения хода их загрузки просто не возникает.

Вопросы безопасности

Flash может считывать данные только с того сервера, на котором расположен Flash-фильм. Это утверждение не является истинным при тестировании фильма во Flash MX или автономном проекторе, но при просмотре фильмов через Internet такое ограничение накладывается. Поэтому по причинам безопасности вы не сможете использовать Flash для загрузки данных с одного узла на другой.

СТАРЫЙ СПОСОБ ЗАГРУЗКИ ПЕРЕМЕННЫХ

Стоит упомянуть и о старом методе `LoadVariables` получения данных с сервера. Дело в том, что очень многие разработчики, научившиеся программировать в старых версиях Flash, до сих пор по привычке продолжают им пользоваться.

Метод `LoadVariables` похож на объект `LoadVars`, но вместо хранения получаемых данных внутри своего объекта он создает переменные и заполняет ими видеоклип, в котором расположен, или корневой уровень, если он находится там. Метод выглядит следующим образом:

```
this.LoadVariables("data.txt");
```

Если рассматривается тот же файл `data.txt`, что и в предыдущем примере, то будут созданы и занесены в видеоклип переменные `a`, `b` и `c`.

При использовании метода `LoadVariables` обнаруживаются два его основных недостатка. Во-первых, данные не хранятся компактно внутри объекта `LoadVars`, а разбросаны по переменным на текущем уровне. Во-вторых, для работы с переменными после их загрузки нельзя создать события, подобные `onLoad`. Вместо этого нужно использовать обработчик `onClipEvent(данные)`, что не всегда удобно.

И еще более серьезная проблема при работе с методом `LoadVariables` заключается в том, что при его использовании для отправления данных (со вспомогательным вторым параметром `GET` или `POST`) все переменные передаются на текущий уровень. Это означает, что необходимо создавать специальный видеоклип только для хранения данных, подлежащих отправке. В подобных ситуациях работать с объектом `LoadVars` значительно проще. Далее в этой главе мы рассмотрим, как использовать объект `LoadVars` для отправки данных обратно на сервер.

ВНЕДРЕНИЕ ДАННЫХ НА HTML-СТРАНИЦУ

Наиболее надежным и эффективным способом передачи данных с сервера в фильм является внедрение данных на Web-страницу в то место, с которого фильм сможет легко их считать. Эта процедура реализуется путем помещения данных в параметры src/movie дескрипторов OBJECT и EMBED на Web-странице. Например, типовой параметр OBJECT фильма будет выглядеть следующим образом:

```
<PARAM NAME=movie VALUE="frompage.swf">
```

Кроме того, в список определений переменных после имени фильма можно поместить вопросительный знак:

```
<PARAM NAME=movie  
VALUE="frompage.swf?pageVar1=7&pageVar2=42.8&pageVar3=Hello  
World!&">
```

В результате три переменные, начиная с pageVar1, будут установлены в соответствующие значения. Это происходит сразу же после загрузки фильма, поэтому можно ожидать, что эти значения будут присутствовать в момент начала выполнения сценария. Все переменные будут находиться на корневом уровне.

Включение переменных с помощью метода loadMovie

Методика помещения переменных после имени файла применима и к командам loadMovie. Flash-фильму можно указать на необходимость загрузки нового фильма, включить в команду вопросительный знак и затем объявить переменные. В результате новый фильм будет загружен с указанным набором переменных. Например:

```
loadMovie("mynewmovie.swf?a=7b=42.8&");
```

Те же самые данные необходимо указать и после параметра src дескриптора EMBED. В противном случае переменные смогут получить только пользователи Windows, работающие с браузером Internet Explorer. Ниже приведен полный дескриптор OBJECT/EMBED с переменными, заданными для его обеих частей.

```
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"  
codebase="http://download.macromedia.com/pub/  
shockwave/cabs/flash/swflash.cab#version=6,0,0,0"  
WIDTH="550" HEIGHT="400" id="frompage">
```

```
<PARAM NAME=movie  
VALUE="frompage.swf?pageVar1=7&pageVar2=42.8&pageVar3=Hello World!&">
```

```
<PARAM NAME=quality VALUE=high>  
<PARAM NAME=bgcolor VALUE=#FFFFFF>
```

```
<EMBED  
src="frompage.swf?pageVar1=7&pageVar2=42.8&pageVar3=Hello World!&"
```

```
quality=high bgcolor=#FFFFFF  
WIDTH="550" HEIGHT="400" NAME="frompage" ALIGN=""  
TYPE="application/x-shockwave-flash"  
PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer">  
</EMBED>  
</OBJECT>
```

Просмотреть данную страницу в действии можно с помощью демонстрационных файлов `Frompage.html` и `Frompage fla`.

Данная методика достаточно мощная и **без** серверной поддержки. Например, с ее помощью можно **создать** **Rash-фильм** с заголовком **Web-узла**. Внутри фильма можно зарезервировать место для отображения заголовка страницы, на которой этот фильм появляется. Созданное динамическое текстовое поле можно связать с переменной, заданной на **Web-странице**. В результате переменная `titleText` на одной странице может принимать значение `My Home Page`, а на другой — `my Links`. Обе страницы используют один и тот же **Flash-фильм**, но с различными значениями переменной `titleText`, заданной в параметрах `src/movie`. Один и тот же фильм с заголовком может выполняться на всех **Web-страницах**.

Кроме того, с помощью данной методики в фильм можно передавать динамические данные. Самым простым примером является передача в фильм текущего времени. Объект `Date` ссылается на время, установленное на компьютере пользователя. Как правило, оно установлено правильно, но может быть легко изменено на любое другое значение. Если ваш **Web-узел** достаточно популярен, то велики шансы, что каждый день на него будут заходить посетители, на компьютерах которых дата и время установлены неправильно. Кроме того, даже посетители, на компьютерах которых дата и время установлены правильно, могут жить в разных часовых поясах. Что же нужно сделать для того, чтобы отобразить точную дату и время в том виде, в каком они установлены на сервере?

Правильное время можно отобразить с помощью серверного включения. Серверное включение представляет собой маленький дескриптор, который помещается на **Web-страницу** для добавления небольшого количества данных или целого файла перед отправлением пользователю. Пользователь видит готовую страницу, которая внешне никак не отличается от обычной **Web-страницы**, но ее **HTML-код** написан специальным образом так, чтобы сервер знал, **какие данные** и куда следует включать.

Серверные включения

Серверные включения разрешены не на всех **Web-серверах**. Чтобы выяснить, включена ли данная функция, и уточнить, нужны ли для ее поддержки какие-то специальные настройки, обратитесь к администратору своего сервера или к **Internet-провайдеру**. Например, многие серверы не поддерживают серверные включения для файлов `.html`, а только для файлов `.shtml`.

Чтобы воспользоваться серверными включениями для отправления во **Flash-фильм** текущего времени, параметр `src/movie` необходимо изменить так, чтобы он выглядел следующим образом:

```
timefrompage.swf?timeVar=<!--#echo var='DATE_LOCAL' -->&
```

Серверным включением является часть `<!--#echo var='DATE_LOCAL' -->&`. Если сервер корректно анализирует синтаксис файла серверного включения, он увидит данный код и заменит его текущим значением даты и времени. Результат может выглядеть так: `Sunday, 24-Feb-2002 10:18:19 MST`.

Изменения вносятся до того, как страница будет отправлена пользователю. Если вы просмотрите исходную страницу, то увидите фактическую дату и время и никаких признаков того, что на реальной **HTML-странице** фактически появляется дескриптор серверного включения.

Демонстрационные файлы `TimefromPage.shtml` и `TimefromPage fla` не будут работать, если запустить их с жесткого диска компьютера или с прилагаемого к книге компакт-диска, поскольку вы не передаете их через **Web-сервер**. Выгрузите файлы `.shtml` и `.swf` в тестовый каталог на узле и просмотрите их в действии. Если в динамическом текстовом поле вместо значений даты и времени вы увидите дескриптор `echo`, то это означает, что серверное включение в вашей системе не поддерживается.

Использование серверных включений представляет собой самый простой способ добавления динамических данных на **Web-страницы**. В настоящее время стали популярными и другие технологии, такие как **Active Server Pages (ASP)**, **Java Server Pages (JSP)**, **Personal Home Pages (PHP)**,

WebObjects и **ColdFusion**. Любую из них можно использовать для создания динамических Web-страниц, содержащих пользовательские переменные, подлежащие передаче во **Rash-фильм**.

Основной недостаток передачи данных в фильм с помощью параметров `src/movie` заключается в том, что, как правило, данные должны быть краткими и что пользователь может увидеть их, просмотрев исходный код Web-страницы, которую он посещает.

ЗАГРУЗКА ДИНАМИЧЕСКИХ ДАННЫХ

Использование динамически создаваемых Web-страниц является лишь одним из способов наполнения фильмов динамическими данными. Помимо этого, для получения данных из динамически созданного источника можно также использовать объект **LoadVars**. Чтобы его можно было использовать, необходимо написать серверную программу, которая называется также *общим иллюзовым интерфейсом*, или **CGI-сценарием**. **CGI-сценарии** пишутся на многих языках, таких как Perl, C и Java. Поскольку обучение этим языкам выходит за рамки данной книги, мы постараемся рассматривать самые простые примеры, в частности, с применением Perl, поскольку этот язык наиболее распространенный и понятный.

Рассмотрим пример получения с сервера текущего времени. Написанная с этой целью на языке Perl программа выглядит так:

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
$time = localtime();
print "$time\n";
exit 0;
```

Если вы хотите испытать эту программу на своем сервере, необходимо предпринять несколько действий. Данный сценарий есть в файле **Time.pl**, имеющемся на прилагаемом к книге компакт-диске. Выгрузите его на свой Web-сервер. Если на вашем сервере **CGI-сценарии** могут выполняться только в специальном каталоге **cgi-bin**, необходимо выгрузить сценарий в этот каталог. Затем необходимо задать разрешения для исполнения сценария.

Написание CGI-программы

Если вы уже растерялись, это означает, что, вероятно, вы никогда не писали **CGI-программу**. В этом случае я настоятельно рекомендую обратиться за помощью к своему Web-администратору или кому-нибудь, имеющему опыт написания **CGI-программ**. Малейшая вариация конфигурации вашего сервера может привести к тому, что программа без помощи опытного программиста не заработает.

Теперь направьте свой браузер к месту расположения **CGI-программы** в Web. Вы должны увидеть страницу, выглядящую приблизительно так:

Sun Feb 24 12:13:02 2003

Если на странице отображено сообщение об ошибке, это означает, что либо сервер не сконфигурирован для исполнения **CGI-программ**, либо не заданы надлежащим образом разрешения. Есть еще множество мелких причин, по которым программа, написанная на языке Perl, может не работать: Perl не установлен на сервере, он находится не в каталоге `/usr/bin/perl`, файл **Time.pl** не был выгружен как текст и т.п. Самый лучший способ решения этих проблем — обратиться за помощью к человеку, который пользовался этими программами раньше.

Теперь, когда есть программа Perl, генерирующая простую Web-страницу, которую может использовать браузер, можно внести такие изменения, чтобы она могла использоваться Flash.

Первым шагом будет использование для возвращаемых данных соответствующего типа MIME. Для этого предназначена вторая строка кода. В данном случае задан тип `text/html`, но нужно указать `application/x-www-urlform-encoded`.

Следующим шагом будет указание выходных данных в формате *имя_свойства=значение_свойства*&. Ниже приведена результирующая программа.

```
#!/usr/bin/perl
print "Content-type: application/x-www-urlform-encoded\n\n";
$time = localtime();
print "currentTime=$time&";
exit 0;
```

Чтобы Flash-фильм мог считывать эти данные, воспользуйтесь объектом `LoadVars`. Новый сценарий будет похож на приведенный в начале настоящей главы:

```
on (release) {
    myLoadVars = new loadVars();
    myLoadVars.onLoad = function() {
        currentTime = myLoadVars.currentTime;
    }
    myLoadVars.load("flashtime.pl");
}
```

Этот код можно увидеть в демонстрационном фильме `Flashtime fla`. Когда пользователь щелкает на кнопке, фильм вызывает файл `Flashtime.pl` и получает возвращаемое им значение. Результат должен выглядеть приблизительно следующим образом: `currentTime=Sun Feb 24 12:13:02 2002`&.

Затем свойство `currentTime` объекта `myLoadVars` копируется в переменную корневого уровня `currentTime`. Эта переменная связана с динамическим текстовым полем, что позволяет увидеть полученный результат.

Чтобы фильм мог работать на сервере, необходимо выгрузить файл `Flashtime.pl`, задать его разрешения и выгрузить файлы `Flashtime.html` и `Flashtime.swf`. Все они должны быть размещены в одном каталоге.

Очень важно понимать, что время, отображаемое во Flash-фильме, является отражением реального времени, установленного на сервере, т.е. время будет разным при каждом запуске фильма. Теперь Flash-фильм и вправду загружает динамические данные.

Установка времени является самым простым примером использования динамических данных. Начиная с этого момента возможность использования подобных данных целиком зависит от ваших навыков серверного программирования. Описанную простую программу Perl можно заменить на такую, которая считывает информацию из базы данных и возвращает данные более сложного вида. Например, программа может возвращать случайный набор простых вопросов или последние биржевые котировки.

ОТПРАВЛЕНИЕ ДАННЫХ

Процедура передачи данных из Flash обратно на сервер похожа на процедуру передачи данных на сервер с HTML-страницы. Если ранее вы уже передавали информацию подобным образом, то наверняка знаете, что для возможности реализации приема и записи отправленной информации на другой стороне линии передачи должна размещаться CGI-программа. Вы также должны знать, что используются два протокола: GET и POST.

Метод GET является самым простым. Если он применяется, то увидеть это можно в адресной строке браузера. Например, в ней может содержаться URL, использующий метод GET для отправления данных на сервер:

```
http://myserver.com/myscript.cgi?name=Gary&age=32&
```

В данном случае выполняется передача двух блоков данных в сценарий `Myscript.cgi`. Последующая обработка данных целиком зависит от сценария.

Ниже приведен пример подобного сценария. В данном случае сценарий, написанный на языке Perl, захватывает входную строку GET и записывает ее в файл **Collect.txt**. В браузер возвращается текст **Success!**:

```
#!/usr/bin/perl

$input_data = $ENV{'QUERY_STRING'};

open(OUTFILE, ">>collect.txt") ;
print OUTFILE "$input_data\n";
close(OUTFILE);

print "Content-type: text/html\n\n";
print "Success!";
```

Исходным URL могут быть данные, вводимые пользователем, начиная от стандартной HTML-ссылки до HTML-формы, в которой используется метод GET.

Данные берутся из части URL, стоящей за вопросительным знаком, которую также называют *строкой запроса*. Команда `open` открывает файл **Collect.txt**. Перед именем файла стоит символ `>>`, означающий, что новые данные будут присоединены к имеющимся в файле. Затем выполняется запись информации, сопровождаемой символом разделителя строк. После закрытия файла записывается слово **Success!**. Данное сообщение заменяет текущую Web-страницу в браузере. Оно свидетельствует о том, что **CGI-программа** выполнила свою работу.

Чтобы этот сценарий работал на сервере, необходимо выгрузить файл **Collect.pl** и пустой текстовый файл **Collect.txt**. Для обоих файлов необходимо задать разрешения. Затем файлы можно протестировать в браузере, вводя в URL различный текст после вопросительного знака. Любые вводимые данные должны быть записаны в файле.

Записывать данные в файл **можно** посредством объекта **LoadVars**. Кроме `load` в объекте **LoadVars** имеется также команда `send`. Приведенный ниже сценарий кнопки собирает три значения переменных в объект **LoadVars** и передает их в **CGI-сценарий**:

```
on (release) {
    myLoadVars = new loadVars();
    myLoadVars.name = username;
    myLoadVars.age = userage;
    myLoadVars.comment = usercomment;
    myLoadVars.send("collect.pl", "_self", "GET");
}
```

Первым параметром команды `send` является местоположение **CGI-программы**. Вторым параметром является целевое окно, в котором будут содержаться результаты, возвращенные **CGI-программой**. В данном случае целевое окно `"_self"` означает, что полученные результаты должны заменить текущую страницу, в том числе и Flash-фильм. Третьим параметром является либо `"GET"`, либо `"POST"`, в зависимости от метода, ожидаемого **CGI-программой**.

Предположим, что вы не хотите заменять страницу и Flash-фильм результатами, возвращаемыми **CGI-программой**, а хотите, чтобы Flash-фильм продолжался. Тогда вместо команды `send` следует использовать команду `sendAndLoad`. Вторым параметром этой команды является не целевое окно браузера, а другой объект **LoadVars**.

В исходном объекте **LoadVars** содержатся переменные, подлежащие отправке, а во втором — возвращаемые данные. Ниже приведен пример.

```
on (release) {
    myLoadVars = new loadVars();
    myLoadVarsReceive = new loadVars();
    myLoadVars.id = userid;
```

```

myLoadVars.password = userpassword;
myLoadVars.onLoad = function() {
    // сделать что-нибудь с информацией
}
myLoadVars.sendAndLoad("collect.pl", myLoadVarsReceive, "GET");
}

```

Для отправления значений переменных на сервер можно также воспользоваться методом POST. Фактически этот метод более предпочтителен, поскольку он позволяет передавать длинные строки. Метод GET, в зависимости от браузера и сервера, накладывает определенные ограничения на размер передаваемой строки.

Чтобы в программе Perl считать информацию, переданную с помощью метода POST, строку, в которой задаются входные данные (`$input_data`), необходимо заменить на такую:

```
read (STDIN, $input_data, $ENV{'CONTENT_LENGTH'});
```

Теперь программа Perl будет считывать данные стандартного входящего потока, поступающего с помощью метода POST. Больше в программу не нужно вносить никаких изменений. Естественно, что во Flash-фильме необходимо изменить команды `load`, `send` или `sendAndLoad`, чтобы вместо метода GET использовался метод POST.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Мои сценарии Perl не выполняются на сервере. Что я делаю неправильно ?

Важно понимать, что **CGI-программирование** отличается от Flash-программирования. Если электрику нужно провести водопровод для того, чтобы иметь возможность прокладывать провода, он не будет делать это сам, а обратится к водопроводчику. Некоторые Flash-программисты могут дополнительно освоить навыки написания CGI-программ, а другим эта задача покажется тяжелой. При возникновении проблем лучше всего обратиться за помощью к людям, обладающим опытом решения подобных задач.

Когда я запускаю фильм во Flash MX, он корректно связывается с сервером. Но когда я выгружаю его в Web, он перестает работать.

Убедитесь в том, что объект `LoadVars` пытается получить данные с того же самого сервера. Если Flash-фильм пытается обратиться к файлу или сценарию другого домена, он наверняка не будет работать. Однако Flash MX и проектор могут взаимодействовать с любым сервером.

Я включаю переменные в дескриптор OBJECT фильма. В браузере Internet Explorer под Windows все работает нормально, а в других браузерах — переменных нет.

Вероятно, вы забыли добавить переменные в параметр `src` дескриптора `EMBED` или допустили опечатку в дескрипторе `EMBED`, а не в дескрипторе `OBJECT`. Браузер Internet Explorer для Windows использует дескриптор `OBJECT` и игнорирует дескриптор `EMBED`, тогда как Netscape и другие браузеры используют дескриптор `EMBED` и игнорируют дескриптор `OBJECT`.

Я уверен, что моя CGI-программа написана правильно и должна работать, но я не могу заставить Flash загрузить из нее данные.

Вы можете и всегда должны тестировать свои **CGI-программы** в браузере с помощью HTML. Для программ, использующих метод GET, переменные можно добавлять в адресную строку браузера. Для программ, использующих метод POST, можно создать простую HTML-страницу с **формой**, которая вызывала бы **CGI-программу** так же, как это должен делать

Flash-фильм. Единственное изменение, которое необходимо внести в CGI-программу, — это задать тип MIME как text/html. Если протестировать CGI-программу таким образом, вы сможете определить, происходит ли проблема из CGI-программы или из Flash-фильма.

FLASH ЗА РАБОТОЙ: ОТПРАВЛЕНИЕ ЭЛЕКТРОННОЙ почты из FLASH

Распространенной задачей, которую приходится решать разработчикам Flash, является возможность отправки электронной почты непосредственно из Flash-фильма. Обычно электронная почта отправляется посредством полнофункциональной почтовой программы, установленной на компьютере пользователя, например, такой как Microsoft Outlook. Однако Flash-фильм выполняется в браузере, а не в почтовой программе. А браузеры, как правило, вообще не имеют встроенных функций отправки электронной почты.

Чтобы из Flash-фильма можно было отправить электронную почту, ее необходимо заставить пройти через сервер. Фактически фильм должен передать информацию на сервер, который выполнит всю остальную работу.

Имеющийся на прилагаемом компакт-диске файл `Email fla` содержит несколько текстовых полей и кнопку Submit (Отправить). Кнопка, содержащая приведенный ниже сценарий, соберет данные из текстовых полей и отправит их в CGI-программу `Email.pl`. Для того чтобы свойства могли быть считаны с сервера, в ней используется команда `sendAndLoad`. Единственным свойством будет `result`, содержащее строку OK в случае, если электронная почта была отправлена.

```
on (release) {  
    // задание переменных для отправки  
    myLoadVars = new loadVars();  
    myLoadVars.name = username;  
    myLoadVars.email = email;  
    myLoadVars.subject = subject;  
    myLoadVars.message = message;  
  
    // задание объекта, подлежащего получению  
    myLoadVarsReceive = new loadVars O;  
    myLoadVarsReceive.onLoad = function() {  
        // переход к кадру, в зависимости от возвращенного сообщения  
        if (myLoadVarsReceive.result == "OK") {  
            gotoAndStop("done");  
        } else {  
            gotoAndStop("error");  
        }  
    }  
}  
  
// отправка электронной почты  
myLoadVars.sendAndLoad("email.pl", myLoadVarsReceive, "POST");  
}
```

На сервере программа `Email.pl` установит связь с серверной программой отправления почты. Эта небольшая программа позволяет CGI-программам отправлять электронную почту с сервера.

Внимание!

Программа отправления почты установлена не на всех серверах. Кроме того, она может быть установлена на сервере, но немного в другом месте. Если вы не уверены, обратитесь за помощью к администратору сервера.

Мне не хотелось бы уделять слишком много времени обучению Perl, однако в приведенном ниже коде даны комментарии, которые позволят увидеть, какую функцию выполняет тот или иной блок кода. Некоторые участки кода даже похожи на `ActionScript`.

```
#!/usr/local/bin/perl

# местоположение почтовой программы сервера
$mailprog = '/usr/lib/sendmail';

# адрес, по которому следует отправить электронную почту
$mailto = 'myemail@myserver.com';

# заголовок для отправления результата обратно во Flash
print "Content-type: application/x-www-form-urlencoded\n\n";

# получение данных по методу POST
read (STDIN,$input_data,$ENV{CONTENT_LENGTH});

# разбивка полученных данных в массив
# и удаление кодировки знаков переключения
@data_array = split('&',$input_data);
foreach $data_item (@data_array) {
    ($tag,$val) = split('=', $data_item, 2);
    $val =~ s/\+/ /g;
    $val =~ s/%([\da-f]{1,2})/pack(C,hex($1))/eig;
    $tags{$tag} = $val;
}

# связь с почтовой программой сервера
open (MAIL, "|$mailprog $mailto") || die "result=Can't open
$mailprog!&";

# запись сообщения
print MAIL "From: $tags{'username'} ($tags{'email'})\n";
print MAIL "Subject: $tags{'subject'}\n\n";
print MAIL "$tags{'message'}\n";
close (MAIL);

# сообщение обратно во Flash
print "result=OK&";
exit 0;
```

Стоит задуматься, нужна ли на самом деле заказчику возможность отправки электронной почты. Например, если вы создаете игру и хотите, чтобы ее участники имели возможность заключать пари, то с помощью данной методики можно легко отправить имя пользователя и его электронный адрес посредством электронной почты. Однако вместо этого можно создать базу данных и хранить в ней записанные имена.

XML-ДАННЫЕ

В ЭТОЙ ГЛАВЕ...

Знакомство сXML	637
Синтаксический анализXML-данных	639
Создание XML-данных	643
ИмпортированиеXML-документов	644
ОтправлениеXML-документовнасервер	646
XML-сокеты	646
Возможные проблемы	647
Flash за работой: простое XML-приложение	647

ЗНАКОМСТВО С XML

Последние несколько лет тема XML является наиболее актуальной среди компьютерных разработчиков. Тем не менее лишь немногие знают, что такое XML и как им пользоваться.

В приложение Flash MX включен ряд функций, позволяющих импортировать, экспортировать и обрабатывать XML-данные. Эти функции являются одними из наиболее мощных, хотя и недостаточно широко используемых инструментов Flash.

XML (Extensible Markup Language — расширяемый язык разметки) представляет собой простой способ хранения информации баз данных. Данные хранятся в виде простых текстовых файлов. Для описания данных используются дескрипторы, во многом похожие на дескрипторы описания данных HTML-страницы.

Язык XML похож на HTML. Рассматривая его под таким углом, легко понять, что он из себя представляет. При этом следует также знать, что между языками XML и HTML существует много отличий. HTML используется для описания Web-страницы, а XML можно использовать

для описания чего угодно. В HTML имеется ряд предварительно заданных дескрипторов, таких как `<P>`, `` и `<A>`, тогда как в XML предварительно заданных дескрипторов нет вообще.

Теперь рассмотрим простой пример XML. Приведенный ниже код описывает склад продуктового магазина, на котором имеется пять яблок (apple), семь апельсинов (orange) и два персика (peach).

```
<inventory>
  <fruit>
    <apples>5</apples>
    <oranges>7</oranges>
    <peaches>2</peaches>
  </fruit>
</inventory>
```

Дескриптор, в который заключен весь код, имеет имя `inventory` (склад). В документе может быть больше одного дескриптора верхнего уровня. Например, помимо `inventory`, в нем могут быть дескрипторы `cash` (касса), `supplies` (поставщики) и `employees` (работники). Из данного кода следует, что в настоящий момент этот XML-документ касается только склада, поэтому в нем есть только один узел верхнего уровня.

Узлами называются *элементы* XML-документа. Дескриптор `inventory` является узлом. Он имеет один дочерний узел, `fruit` (фрукт), который, в свою очередь, имеет три своих дочерних узла: `apples`, `oranges` и `peaches`.

Узлы, подобные `inventory`, `fruit` и `apples`, называют *XML-узлами*. Узлы, подобные 5, 7 и 2, называют *текстовыми узлами*. XML-узел имеет имя, но не имеет значения. Обычно вместо значения он имеет несколько дочерних узлов. Текстовый узел, наоборот, не имеет имени, но имеет значение.

На рис. 28.1 графически представлен описанный XML-документ. Каждый прямоугольник представляет собой узел. Если один прямоугольник расположен внутри другого, это означает, что он является дочерним для большего прямоугольника.

Первым шагом на пути к использованию XML-документов является понимание того, как следует перемещаться внутри документа. Например, `inventory` является головным узлом, а `fruit` — дочерним для него. В свою очередь `fruit` имеет три дочерних узла, первым из которых является `apples`. Узел `apples` имеет один дочерний текстовый узел со значением 5.

Таким образом, 5 — это первый дочерний элемент `apples`, являющегося первым дочерним элементом `fruit`, являющегося первым дочерним элементом `inventory`, являющегося первым дочерним элементом документа.

Поскольку это описание слишком многословно, вам может показаться, что при работе с XML всегда придется иметь дело с такими длинными цепочками. Однако есть способ их сократить. Существует основное правило XML, позволяющее превратить сложные документы в простые. Оно гласит, что все XML-узлы являются, по сути, более мелкими XML-документами.

Поэтому узел `apple` можно считать самостоятельным документом. В этом случае головным узлом является `apples`, а его дочерним элементом будет текстовый узел со значением 5. В оставшейся части этой главы для анализа XML-данных будет применяться данная методика.

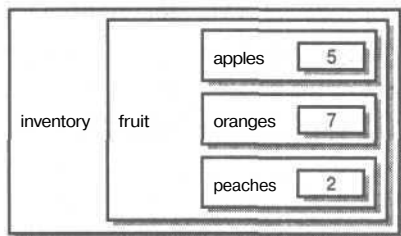


Рис. 28.1. Графическое представление XML-документа продуктового магазина

Основы XML

Одним из самых важных аспектов XML является его удобочитаемость как компьютерами, так и людьми. Это отличает его от других языков представления баз данных, в которых либо используется двоичный код, либо данные совершенно недоступны для чтения.

Таким образом, вы легко можете самостоятельно создать и отредактировать простой XML-документ, воспользовавшись обычным текстовым редактором, например Notepad (Блокнот) или SimpleText. Однако в большинстве случаев XML-документы создаются в других программах. Передавать XML-данные могут многие серверные программы управления базами данных. Некоторые приложения баз данных, которые вы запускаете на **своем** компьютере, имеют функцию экспортирования в XML.

Чтобы XML-документ мог быть считан во Flash, он должен быть *правильно создан*. Это означает, что должен быть соблюден определенный набор правил, некоторые из которых являются очевидными.

Все дескрипторы XML-документа должны иметь соответствующие закрывающие дескрипторы, т.е. дескриптору `<apples>` должен соответствовать закрывающий дескриптор `</apples>`. Кроме того, дескрипторы должны быть расположены в правильном порядке. Например, порядок `<fruit><apples>2</apples></fruit>` является правильным, а порядок `<fruitxapples>2</fruitx/apples>` — **ошибочным**, и в последнем случае синтаксический анализ документа не будет проведен корректно.

Весь XML-документ можно поместить в одну строку, без дескрипторов и символов обрыва строки. Но при этом документ может стать "нечитабельным" для пользователей. Во Flash MX существует возможность игнорировать символы табуляции и обрыва строки. Чтобы избавиться от пробелов во Flash 5, приходится писать специальный код.

Дескрипторы в XML имеют не только значения или дочерние узлы, но и атрибуты. Например, если в дескрипторе необходимо указать больше информации, чем просто `apples`, можно использовать атрибут для задания более специфичного параметра:

```
<apples type='green'>5</apples>
```

Во Flash атрибуты помечают одинарными, а не двойными кавычками.

ИСПОЛЬЗОВАНИЕ XML ВО FLASH

XML во Flash используется по двум причинам. Во-первых, это облегчает взаимодействие с другими системами. Если, к примеру, в фильм необходимо передать информацию из медицинской базы данных, существует большая вероятность того, что администратор этой базы данных совершенно не знаком с Flash, но имеет познания в области XML. Данные можно представить в формате XML и использовать XML для передачи информации. Во многих случаях в базах данных или в сетях связи, доступ к которым необходимо получить из Flash, уже используется XML.

Во-вторых, функциональные возможности XML гораздо лучше подходят для обработки больших и сложных данных, чем другие части Flash.

Например, если необходимо считывать массив данных, то как бы это можно было сделать без XML? Можно воспользоваться объектом `LoadVars` для считывания данных в виде большой строки, а затем разбить эту строку в массив с помощью команды `split`. Но этот подход слишком медленный. Его не ускорит даже написание собственной альтернативы команды `split`. Либо же данные можно считывать в виде десятков и сотен переменных, но этот подход является трудно организуемым.

С помощью XML-объекта можно быстро считывать и легко получать доступ к большому количеству данных. Этот подход оптимален для получения и отображения информации из базы данных. Например, его можно использовать для считывания ряда вопросов игры или отображения данных пользователя в его базе данных. Выполнение тех же самых операций с помощью обычных переменных и массивов будет гораздо медленнее и потребует значительно больших усилий.

СИНТАКСИЧЕСКИЙ АНАЛИЗ XML-ДАННЫХ

Как же получить XML-данные во Flash? Для этого XML-документ должен быть перенесен во Flash и преобразован в XML-объект, который можно использовать в ActionScript. Преобра-

ние текста в другой формат, как в данном случае, называют *синтаксическим анализом*. В следующих подразделах описываются основные функции XML-объекта и способы их применения для получения и считывания XML-данных.

ПРЕОБРАЗОВАНИЕ ТЕКСТА в XML

Самым легким способом создания XML-объекта является преобразование небольшого участка надлежащим образом отформатированного текста в XML. Эта операция заключается в создании нового XML-объекта и наполнении его текстом:

```
myText = "<inventory><fruit><apples>5</apples><oranges>7</oranges>";  
myText += "<peaches>2</peaches></fruit></inventory>";  
myXML = new XML(myText);
```

Эту же самую операцию можно выполнить с помощью команды `parseXML`:

```
myText = "<inventory><fruit><apples>5</apples><oranges>7</oranges>";  
myText += "<peaches>2</peaches></fruit></inventory>";  
myXML = new XML();  
myXML.parseXML(myText);
```

Хотя операция по созданию XML из текста, уже находящегося внутри Flash-фильма, имеет ограниченное применение, этот подход хорош для знакомства с XML. Далее в этой главе мы расскажем о том, как импортировать XML-документы больших размеров из внешних файлов и программ.

ИССЛЕДОВАНИЕ XML-ДАННЫХ

В предыдущей функции команда `trace` предназначена для вывода имени узла первого дочернего элемента в документе. Поскольку в файле `Produce.xml` содержится пример продуктового магазина, описанный ранее в этой главе и представленный на рис. 28.1, легко предугадать, что в результате должно быть выведено имя `inventory`. Узел `inventory` является первым дочерним элементом XML-объекта.

На заметку

Работая с XML, имейте в виду, что некоторые термины обозначают одно и то же. Например, *XML-документ* и *XML-объект* во Flash обозначают одно и то же; узел XML-объекта — это то же самое, что и другой *XML-объект*.

Свойство `firstChild` возвращает первый дочерний узел XML-документа. С помощью подобного массиву свойства `childNodes` можно получить любой дочерний узел. Таким образом, `firstChild` и `childNodes[0]` означают одно и то же.

Помните, что узел XML-объекта представляет собой другой XML-объект. Узлы могут представлять собой либо XML-узлы с именами и дочерними элементами, либо текстовые узлы без имен или дочерних элементов, но имеющие значения.

Для получения имени XML-узла можно воспользоваться свойством `nodeName`, а для получения значения текстового узла — свойством `nodeValue`.

Ниже перечислены различные XML-свойства, которыми можно воспользоваться для получения дочерних элементов, имен и данных XML-объектов.

- **firstChild**. Возвращает первый дочерний элемент узла. То же самое можно получить посредством `childNodes[0]`.
- **hasChildNodes**. Возвращает значение `true`, если узел имеет дочерние элементы.
- **lastChild**. Возвращает последний дочерний элемент узла. Количество дочерних узлов можно получить с помощью свойства `childNodes.length`.

- `nextSibling`. Возвращает следующий узел, если таковой имеется.
- `nodeName`. Возвращает имя дескриптора XML-узла или значение `null`, если данный узел является текстовым.
- `nodeType`. Возвращает 1, если узел является XML-узлом, либо значение 3, если он является текстовым.
- `nodeValue`. Возвращает значение текстового узла либо `null`, если узел является XML-узлом.
- `parentNode`. Возвращает узел, являющийся родительским для данного.
- `previousSibling`. Возвращает предыдущий узел, если таковой имеется.
- `attributes`. Возвращает объект переменной, содержащий атрибуты (если имеются), либо узел.
- `childNodes`. Возвращает структуру дочерних узлов, подобную массиву. Для получения общего количества узлов можно воспользоваться свойством `length`, а для указания определенного узла — квадратными скобками.

Несмотря на простоту примеров, в которых было представлено свойство `firstNode`, для навигации по XML-документу чаще всего используют свойство `childNodes`. Ряд приведенных ниже примеров поможет понять, как следует использовать это свойство для получения нужной информации. Сверяйтесь с рис. 28.1 или с предшествующим ему текстом.

Данная строка вернет `inventory`:

```
myXML.childNodes[0].nodeName
```

Следующая строка вернет значение `null`, поскольку первый узел является XML-узлом, который имеет имя и содержит дочерние элементы, но сам по себе значения не имеет:

```
myXML.childNodes[0].nodeValue
```

Эта строка вернет `fruit`:

```
rayXML.childNodes[0].childNodes[0].nodeName
```

Данная строка вернет `apples`:

```
myXML.childNodes[0].childNodes[0].childNodes[0].nodeName
```

Следующая строка вернет значение `null`, поскольку узел `apples` все равно является XML-узлом и не имеет значения, а имеет только имя и дочерние элементы:

```
myXML.childNodes[0].childNodes[0].childNodes[0].nodeValue
```

Чтобы получить значение 5, содержащееся в узле `apple`, необходимо взглянуть на дочерний элемент этого узла, которым является текстовый узел. Приведенная ниже строка вернет значение 5.

```
myXML.childNodes[0].childNodes[0].childNodes[0].childNodes[0].nodeValue
```

В связи с тем, что текстовые узлы не имеют имен, при попытке получить имя такого узла будет возвращено значение `null`:

```
myXML.childNodes[0].childNodes[0].childNodes[0].childNodes[0].nodeName
```

Узел `fruit` имеет три дочерних элемента. Это можно определить, анализируя свойство `length` узла `childNodes`. Приведенная ниже строка вернет значение 3, поскольку узел `fruit` имеет три дочерних элемента.

```
myXML.childNodes[0].childNodes[0].childNodes[0].length
```

Получить имя второго дочернего элемента узла **fruit**, каковым является узел **orange**, можно следующим образом:

```
myXML.childNodes[0].childNodes[0].childNodes[1].nodeName
```

Чтобы определить, является ли элемент XML-узлом или текстовым узлом, можно использовать свойство **nodeType**. Например, узел **apple** является XML-узлом, и приведенная ниже строка вернет значение 1.

```
myXML.childNodes[0].childNodes[0].childNodes[0].nodeType
```

С другой стороны, узел, имеющий значение 5, является текстовым, и приведенная ниже строка вернет значение 3.

```
myXML.childNodes[0].childNodes[0].childNodes[0].  
childNodes[0].nodeType
```

ОБЛЕГЧЕНИЕ ОБРАЩЕНИЯ К XML

Некоторые из строк приведенного выше примера уже являются достаточно длинными, и это при том, что рассмотрен был *простой* пример. Представьте, что было бы, если бы документ имел еще несколько уровней. Мы получили бы строки, содержащие полдесятка обращений к узлам **childNodes**. Все эти обращения могли бы сделать код слишком длинным и запутанным.

К счастью, существует более простой способ углубиться в структуру XML-документа и получить нужные данные. Предположим, например, что необходимо получить количество яблок, имеющихся в магазине. Ниже приведен код, с помощью которого можно решить данную задачу.

```
inventory = myXML.childNodes[0] ;  
fruit = inventory.childNodes[0] ;  
apples = fruit.childNodes[0] ;  
appleCount = apples.childNodes[0] ;  
numApples = appleCount.nodeValue ;
```

Несмотря на то что данный код ненамного короче, чем строка с многочисленными обращениями к узлам **childNodes**, понять его гораздо легче. Кроме того, если далее необходимо будет получить количество апельсинов, то у вас в наличии уже есть готовая переменная **fruit**.

ОБРАЩЕНИЕ К АТТРИБУТАМ

Помимо дочерних элементов, XML-узлы могут также иметь атрибуты. Например, приведенный ниже XML-документ имеет атрибут с именем **type**, который имеет значение 4.

```
<test type='4'>123</test>
```

Чтобы получить значение атрибута, необходимо знать его имя. Ниже приведен код с обращением к атрибуту из предыдущего примера.

```
myXML.childNodes[0].attributes.type
```

Свойство **attributes** фактически возвращает объект переменной, представляющий собой перечень имен свойств и значений. Если вы хотите увидеть все имена свойств и значения указанного атрибута, можете воспользоваться кодом, подобным приведенному ниже.

```
myAttributes = myXML.childNodes[0].attributes  
for(attribute in myAttributes) {  
    trace(attribute+": "+myAttributes[attribute]);  
}
```

СОЗДАНИЕ XML-ДАННЫХ

ActionScript позволяет создать XML-документ с нуля или изменить уже существующий. После создания пустого объекта к нему можно добавить узлы посредством множества команд.

Однако процедура создания XML-объекта немного беспорядочна. Начнем с простого примера и создадим небольшой XML-документ, который выглядит так:

```
<test>123</test>
```

Сначала необходимо следующим образом создать XML-объект:

```
myXML = new XML();
```

Затем необходимо добавить узел с именем **test**. Эта процедура выполняется в два этапа. Первый из них заключается в создании узла:

```
testNode = myXML.createElement("test");
```

На данном этапе создается узел с именем **test**. Даже несмотря на то, что он является частью объекта **myXML**, фактически это ничего не дает. Его все равно необходимо присоединить к узлу. В связи с тем, что в данном случае мы имеем дело с новым пустым XML-объектом, только сам объект является узлом:

```
myXML.appendChild(testNode);
```

Для создания текстового узла, который будет дочерним для узла **test**, необходимо воспользоваться немного другой командой:

```
textNode = myXML.createTextNode("123");
```

Теперь данный узел необходимо добавить к **test** в качестве дочернего элемента. К счастью, для обращения к этому узлу все еще можно воспользоваться переменной **testNode**:

```
testNode.appendChild(textNode);
```

Для проверки заданных команд можно воспользоваться функцией **toString()** и отправить XML-документ в окно **Output**:

```
trace(myXML.toString());
```

В демонстрационном файле **Build.fla** показан описанный пример с фруктами, реализованный с помощью **ActionScript**. Ниже приведены другие команды, которые можно использовать для модификации XML-объекта.

- **appendChild(узел)**. Добавляет узел, который был создан с помощью команды **createElement** или **createTextNode**.
- **cloneNode(глубина)**. Создает копию узла. Если параметр *глубина* имеет значение **true**, то копируются все дочерние элементы и последующие потомки.
- **createElement(имя_узла)**. Создает новый XML-узел. Этот узел не будет являться частью документа до применения команды **appendChild** или **insertBefore**.
- **createTextNode(значение_узла)**. Создает новый текстовый узел. Этот узел не будет являться частью документа до применения команды **appendChild** или **insertBefore**.
- **insertBefore(узел, номер_узла)**. Вставляет узел, созданный с помощью команд **createElement** или **createTextNode**.
- **removeNode(номер_узла)**. Удаляет дочерний узел.

СОЗДАНИЕ и МОДИФИКАЦИЯ АТРИБУТОВ

Пока что за пределами команд о модификации XML оказались способы добавления или изменения атрибутов. Модифицировать атрибуты **проще** простого. Достаточно присвоить значение создаваемому или модифицируемому узлу. Например:

```
myXML.childNodes[0].attributes.myAttribute = 42;
```

Для создания простого XML-объекта с атрибутами, например `<test type="4">7</test>`, можно воспользоваться приведенным ниже кодом.

```
myXML = new XML();

testNode = myXML.createElement("test");

testNode.attributes.type = 4;
myXML.appendChild(testNode);

text = myXML.createTextNode("7");
testNode.appendChild(text);
```

ИМПОРТИРОВАНИЕ XML-ДОКУМЕНТОВ

В большинстве случаев использования XML-документов они фактически импортируются извне Flash-фильма. В качестве XML-документов могут выступать текстовые файлы, расположенные на сервере или сгенерированные серверными **CGI-программами**.

Внимание!

XML-документы можно загружать только из того же домена, в котором находится Flash-фильм. Из других узлов XML-документы загрузить нельзя. Это справедливо только в том случае, если фильм выполняется в браузере.

В некоторых случаях XML-объект подобен объекту **LoadVars**. В нем для загрузки данных из внешнего файла или программы можно воспользоваться командой **load**.

Однако вместо файла формата **свойство=значение&** он должен быть в формате XML. Если файл является корректным XML-документом, он будет немедленно проанализирован как XML-объект.

Теперь рассмотрим следующий пример. Приведенный ниже код, будучи помещенным в первый кадр фильма, создаст новый XML-объект и загрузит его в фильм. Между этими двумя командами я задал свойство **ignoreWhite** и установил его в значение **true**. Это означает, что в документе будут проигнорированы символы табуляции и символы разрыва строки. Если не установить это свойство, то вам придется написать XML-документ вообще без символов табуляции и пустых строк или же в XML-объекте будут содержаться всевозможные нежелательные текстовые узлы.

```
myXML = new XML();
myXML.ignoreWhite = true;
myXML.load("produce.xml")
```

Команда **load** не завершает задачу немедленно. Вам придется подождать до тех пор, пока текстовый файл не будет загружен и проанализирован. Даже если все файлы находятся на локальном жестком диске вашего компьютера, то все равно доступ к следующей строке кода нельзя будет получить слишком быстро.

XML-объект, как и объект **LoadVars**, имеет событие **onLoad**. Можно определить функцию, которая будет запущена сразу же по завершении загрузки. Ниже приведен пример.

```

myXML = new XML();
myXML.ignoreWhite = true;
myXML.load("produce.xml")
myXML.onLoad = function() {
    trace(myXML.firstChild.nodeName);
}

```

Еще один способ сообщения о том, что XML-файл был **загружен**, заключается в использовании свойства `loaded` XML-объекта. Оно вернет значение `true` только после завершения импортирования.

Кроме того, имеется еще ряд команд, предназначенных для отслеживания хода выполнения загрузки большого документа. Ниже приводится их полный список.

- `getBytesLoaded`. Возвращает общее число битов, загруженных на данный момент.
- `getBytesTotal`. Возвращает общее число битов XML-файла.
- `loaded`. Возвращает значение `true` только после того, как загрузится весь документ.
- `status`. Возвращает значение 0, если загрузка была выполнена успешно. Другие возможные значения представлены в табл. 28.1.

ТАБЛИЦА 28.1. ЗНАЧЕНИЯ, ВОЗВРАЩАЕМЫЕ с помощью КОМАНДЫ `XML.STATUS`

Код	ЗНАЧЕНИЕ
0	Анализ завершен без ошибок
-2	Дескриптор CDATA не завершен
-3	XML-дескриптор не завершен
-4	Дескриптор DOCTYPE не завершен
•5	Компонент не завершен
-6	Плохо сформирован XML-элемент
-7 •	Нехватка памяти
-8	Параметр атрибута не завершен
-9	Начальный дескриптор не имеет соответствующего закрывающего дескриптора
-1°	Найден конечный дескриптор без соответствующего начального дескриптора

Событие `onLoad` наступает после того, как произойдет следующее. Во-первых, должен полностью загрузиться текстовый файл, содержащий XML. Во-вторых, XML-документ должен быть проанализирован как XML-объект.

Однако XML-текст можно перехватить, не анализируя. С этой целью используется событие `onData`, которое позволяет захватить данные до их анализа. Затем данные можно анализировать самостоятельно, хотя перед этим придется создавать определенные пользовательские функции. Ниже приведен пример.

```

myXML.onData = function(xmlText) {
    myXML.parseXML(xmlText);
}

```

В данном примере код не выполняет никаких функций, которые нельзя было бы выполнить с помощью обычной команды `load`. Однако видно, как выполнить некоторые нетривиальные задачи. Например, если серверная программа возвращает XML-документ, в начале которого имеется дополнительная строка, то ее можно изъять из параметра `xmlText` до его передачи в команду `parseXML`.

После того как XML-документ был проанализирован посредством команды `load` или `parseXML`, из него можно извлекать всевозможную информацию с помощью одного из приведенных ниже свойств.

- `xmlDecl`. При использовании стандартов XML в начале каждого XML-документа должен быть XML-дескриптор. Данное свойство возвращает строку, содержащую этот дескриптор.
- `docTypeDecl`. Еще одним стандартным дескриптором XML-документов является `!DOCTYPE`. Данное свойство вернет строку со значением этого дескриптора.

ОТПРАВЛЕНИЕ XML-ДОКУМЕНТОВ НА СЕРВЕР

После создания XML-документа с помощью `ActionScript` его необходимо отправить обратно в серверную программу для хранения или обработки. Эту операцию можно выполнить с помощью команд `send` и `sendAndLoad`.

Эти две команды выполняют те же функции, что и при использовании с объектом `LoadVars`. Команда `send` отправляет полный XML-объект обратно в адрес сервера. Если в ней изменен второй параметр, как в приведенном ниже примере, то текст, возвращенный сервером, будет передан в другой фрейм или в окно браузера.

```
myXML.send("http://www.myserver.com/getxml.cgi", "_self");
```

На заметку

Если XML-данные необходимо передать на сервер с использованием специфического типа MIME, то перед использованием команды `send` или `sendAndLoad` свойство `contentTypeXML`-объекта можно установить в значение нужного типа MIME.

В приведенной выше строке кода в качестве возвращаемого целевого элемента указан элемент `"_self"`. Это означает, что Web-страница будет целиком заменена содержимым, возвращаемым серверной программой. Если второй параметр опустить, то операция объявлена не будет и продолжится воспроизведение Flash-фильма.

Совет

Информация об объекте `LoadVars` приведена в *главе 27* "Взаимосвязь с сервером".

Применение команды `sendAndLoad` означает, что вторым параметром должен быть еще один XML-объект. Этот новый объект получит возвращаемый с сервера XML так, как если бы использовалась команда `load`.

```
returnXML = new XML();  
myXML.send("http://www.myserver.com/getxml.cgi", returnXML);
```

Если вы хотите отследить ход загрузки в `returnXML`, необходимо использовать события `onData` и `onLoad` объекта `returnXML`, а не объекта `myXML`.

XML-СОКЕТЫ

XML-сокеты тематически связаны с XML-объектами только в том, что и в тех и в других используются данные формата XML. Однако в объекте `XML Socket` для получения и коллективного использования данных не применяются команды `load` и `send`. Вместо них для соединения с серверной программой используется команда `connect`.

Серверная программа сокета отличается от CGI-программы тем, что она выполняется всегда, а не только при ее вызове. Это означает, что сервер все время должен ожидать входящих со-

единений. Затем для установления соединения Flash-фильм использует команду `connect`. После установления соединения XML-данные можно передавать в обоих направлениях в режиме реального времени. Сервер может передавать данные в фильм даже в случае, если фильм их не запрашивает. Эта методика известна под названием *технологии принудительной накачки*.

Большинство атрибутов, которые необходимо знать для использования XML-сокетов, не относятся к Flash или **ActionScript**. В этом случае требуется писать серверную программу, обычно на языке Java или C++. Кроме того, необходимо хорошо ориентироваться в том, как работают серверы и **сокеты**. По этой причине мы не будем подробно рассматривать функционирование XML-сокетов.

Если для связи Flash с сокетами используется сторонняя программа, то ее поставщик должен предоставить документацию о том, как использовать **XML-сокеты** для взаимосвязи с его продуктом.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

После импортирования XML-документа я не могу найти узлы, которые должны быть на месте. Есть ли способ удостовериться в том, что операция импортирования выполняется?

Существует много способов проверки документа после его импортирования. Проще всего воспользоваться функцией `toString()` с командой `trace` для передачи XML-объекта в окно Output. Кроме того, можно проверить свойство `status` объекта. Если в каком-то из случаев возникли **проблемы**, это означает, что в XML-документе имеются ошибки. Необходимо построчно просмотреть документ, чтобы найти пропущенный или написанный с ошибкой дескриптор, либо использовать стороннее средство создания XML.

После импортирования XML-документа у меня получилось очень много пустых текстовых узлов. Почему это произошло?

Любые пробелы, имеющиеся в документе, например символы возврата или табуляции, будут преобразованы в немного бесполезные текстовые узлы, если перед анализом документа не установить свойство `ignoreWhite` в значение `false`.

Я могу получить имя узла, но не могу получить значение этого же узла. Почему?

Узлы не имеют имен и значений. Они имеют или имена, или значения. Существует вероятность того, что вы фактически ищите не значение, а текстовый узел, **являющийся** дочерним для узла, имеющего имя.

FLASH ЗА РАБОТОЙ: ПРОСТОЕ XML-ПРИЛОЖЕНИЕ

XML, в первую очередь, считается технологией баз данных. Flash редко используется для работы с базами данных, но благодаря возможностям XML такие операции можно выполнять.

В данном примере мы рассмотрим создание простой программы управления базой данных. Эта программа позволяет пользователю вводить новые записи в базу данных, перечислять записи в базе данных и редактировать записи. Кроме того, можно загрузить из внешнего файла полную базу данных.

Окончательная версия программы содержится в демонстрационном файле **Books fla**. Рассмотрим ее **покадрово** и проанализируем код.

В первом кадре содержатся две строки, инициализирующие базу данных. Первая строка создает пустой XML-объект, вторая устанавливает в значение о глобальную переменную **numEdit**. Эта переменная будет использоваться для отслеживания записи, которую редактирует пользователь.

```
database = new XML();
numToEdit = 0;
```

Во втором кадре содержатся четыре кнопки: Load Books (Загрузить книги), List Books (Перечислить книги), New Book (Новая книга) и Edit Book (Редактировать книгу). Кроме того, в данном кадре содержится команда `stop()`.

Динамическое поле связано с переменной `total`. Приведенные ниже две строки кода помещают в данное поле общее количество записей XML-объекта.

```
t = database.childNodes.length;
total = "Total Number of Books: "+t;
```

В каждой из четырех кнопок содержится простой обработчик `on (release)` с командой `gotoAndStop`. После щелчка на кнопках выполняется переход соответственно к кадрам `load`, `list`, `new` или `edit`.

В кадре `load` содержится простой сценарий загрузки XML-документа из файла `books.xml`. Этот файл должен храниться в том же каталоге, что и Flash-фильм. На прилагаемом к книге компакт-диске имеется демонстрационный файл `Books.xml`.

```
database = new XML();
database.ignoreWhite = true;
database.load("books.xml");
database.onLoad = function() {
    gotoAndStop("menu");
}
```

После загрузки и анализа файла фильм вернется к кадру меню. Если вы хотите написать более надежную версию кода, можете выполнить проверку свойства `status` базы данных. Если оно даст любое значение, отличное от 0, воспользуйтесь **расшифровкой** возвращенных значений, приведенных в табл. 28.1, и сообщите пользователю, что было сделано неправильно.

Как выглядит запись этой базы данных? В файл `books.xml` включены три записи. Вот первая из них:

```
<book isbn='0441142109'>
  <title>Deathworld</title>
  <author>Harry Harrison</author>
  <copies>1</copies>
  <price>5.00</price>
</book>
```

Главный узел имеет имя `book` (книга). Он имеет один атрибут `isbn`. Имеется четыре дочерних узла: `title` (название), `author` (автор), `copies` (экземпляры) и `price` (цена). Каждый из них имеет один дочерний текстовый узел.

Файл может содержать множество подобных записей. Каждый узел `book` является дочерним элементом XML-документа.

В кадре `new` не содержится большое количество кода. Данный кадр является совокупностью текстовых полей ввода данных пользователем. Код просто сбрасывает предыдущие значения полей, устанавливая переменные, с которыми они связаны, в значения пустых строк:

```
isbn = "";
title = "";
author = "";
copies = "";
price = "";
```

На рис. 28.2 изображен кадр `new`. Вы видите поля, которые может заполнять пользователь для каждой записи.

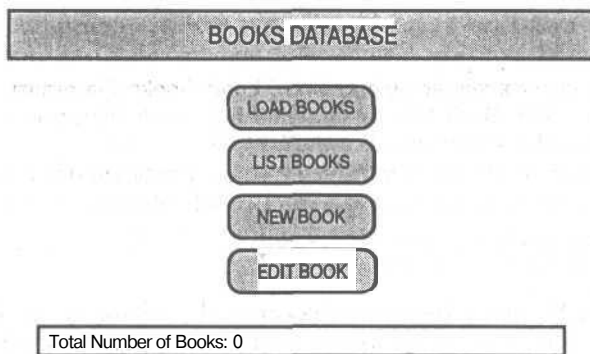


Рис. 28.2. Кадр new позволяет пользователю вводить записи в базу данных

В нижней части кадра new имеются две кнопки. Первая отменяет действие, игнорируя введенные данные и возвращаясь к кадру меню. Вторая кнопка создаст XML-узлы, необходимые для добавления записи в базу данных. Ниже приводится полный код этой кнопки. Проанализируйте его построчно и выясните, как создается и добавляется каждый элемент.

```
on (release) {
    // создать узел book с атрибутом isbn
    book = database.createElement("book");
    book.attributes.isbn = isbn;

    // добавить название книги
    node = database.createElement("title");
    book.appendChild(node);
    text = database.createTextNode(title);
    node.appendChild(text);

    // добавить автора книги
    node = database.createElement("author");
    book.appendChild(node);
    text = database.createTextNode(author);
    node.appendChild(text);

    // добавить число экземпляров книги
    node = database.createElement("copies");
    book.appendChild(node);
    text = database.createTextNode(copies);
    node.appendChild(text);

    // добавить цену книги
    node = database.createElement("price");
    book.appendChild(node);
    text = database.createTextNode(price);
    node.appendChild(text);

    // добавить книгу к XML
    database.appendChild(book);

    gotoAndStop("menu");
}
```

Следующим является кадр `list`. Он содержит большое динамическое текстовое поле, связанное с переменной `booklist`. Оно задано многострочным, без переносов, так что в нем можно отобразить много строк, каждая из которых будет представлять книгу.

Код данного кадра циклически проходит по всем XML-записям и получает значение текстового узла, являющегося дочерним для узла `title` (заголовок). Эти значения добавляются в переменную `booklist` с символом новой строки после каждой книги:

```
booklist = "";
for(var i=0;i<database.childNodes.length;i++) {
    book = database.childNodes[i];
    titleNode = book.childNodes[0];
    title = titleNode.firstChild.nodeValue;
    booklist += title + "\n";
}
```

Этот код можно изменить так, чтобы отображать не только название книги. Например, чтобы добавить в каждую строку имя автора, необходимо получить дочерний текст второго (номер 1) дочернего элемента узла `book`. Либо же можно поместить номер узла `book` в начало строки, в результате чего книги будут пронумерованы. Чтобы получить ISBN книги, взгляните на атрибут `isbn` узла `book`. Полный синтаксис номера ISBN выглядит так: `book.attributes.isbn`.

Последний кадр фильма, `edit`, похож на кадр `new` в том смысле, что он имеет те же самые текстовые поля ввода данных. Но на этот раз в поля будут заранее внесены функции. Это дает значения записи, соответствующей параметру `numToEdit`, и помещает их в переменные, связанные с полями:

```
function populateForEdit() {
    book = database.childNodes[numToEdit];
    isbn = book.attributes.isbn;
    title = book.childNodes[0].firstChild.nodeValue;
    author = book.childNodes[1].firstChild.nodeValue;
    copies = book.childNodes[2].firstChild.nodeValue;
    price = book.childNodes[3].firstChild.nodeValue;
}
```

Сценарий кадра завершается вызовом функции `populateForEdit()` для задания полей текущей записи. Однако пользователь может воспользоваться одной из двух кнопок в нижней части экрана, чтобы перейти к следующей или предыдущей записи. Сценарий кнопки перехода к предыдущей записи выглядит следующим образом:

```
on (release) {
    numToEdit--;
    if (numToEdit < 0) numToEdit = 0;
    populateForEdit();
}
```

Ниже приводится пример сценария кнопки перехода к следующей записи,

```
on (release) {
    numToEdit++;
    if (numToEdit > database.childNodes.length-1) {
        numToEdit = database.childNodes.length-1;
    }
    populateForEdit();
}
```

Кнопка применения действий принимает значения полей ввода данных и применяет их к записи в базе данных XML:

```

on (release) {
    book = database.childNodes[editNum];
    book.attributes.isbn = isbn;
    book.childNodes[0].firstChild.nodeValue = title;
    book.childNodes[1].firstChild.nodeValue = author;
    book.childNodes[2].firstChild.nodeValue = copies;
    book.childNodes[3].firstChild.nodeValue = price;
}

```

Кнопка выполнения вернет пользователя обратно к меню, игнорируя любые изменения, внесенные в поля ввода данных. В качестве альтернативного варианта эта кнопка может использоваться и для применения изменений, и для возврата к меню. В этом случае будет необходима кнопка отмены.

На этом рассмотрение программы завершено. Здесь преднамеренно опущено описание способа сохранения введенных данных. Загрузить данные во Flash очень легко, а вот сохранить их во внешнем файле гораздо сложнее. Одним из вариантов является сохранение данных в виде локального коллективного объекта. Но тогда эти данные будут доступны только в определенном Flash-фильме. Если вы стремитесь именно к такому результату, тогда все в порядке.

Совет

Локальные коллективные объекты рассматривались в главе 26 "Локальная связь".

В еще одном способе сохранения базы данных XML **задействуется** серверная программа, которая будет принимать XML-объект из команды `send` или `sendAndLoad`. Если вы имеете опыт написания подобных серверных программ, то легко освоите и эту методику. В противном случае для выполнения подобной задачи вам потребуется помощь серверного программиста.

ВЫХОДНЫЕ ПАРАМЕТРЫ FLASH

В ЭТОЙ ЧАСТИ...

Глава 29. Печать Flash-фильма

Глава 30. Оптимизация, публикация и экспортирование фильмов

ПЕЧАТЬ FLASH-ФИЛЬМА

В ЭТОЙ ГЛАВЕ...

Печать содержимого Flash	655
Подготовка файлов к печати	656
Создание кнопки Print	663
Возможные проблемы	667
Flash за работой: невидимые чернила	668

ПЕЧАТЬ СОДЕРЖИМОГО FLASH

Печать многих типов содержимого **Flash**, таких как анимация, видео и аудио, может показаться невыполнимой задачей. Разве можно зафиксировать и эффективно отобразить на бумаге мультимедийное содержимое фильма? Однако нельзя не учитывать тот факт, что пользователям может понадобиться распечатать части Flash-узлов. Такие элементы, как подтверждение заказа, инструкции, длинные фрагменты текста и данные формы, должны предусматривать возможность вывода на печать.

Учитывая многообразие типов содержимого Flash-узлов, очень важно уметь управлять параметрами печати. Как и в вопросах, касающихся доступности и используемости узлов, на разработчиках и дизайнерах, работающих с Flash, лежит ответственность за обеспечение средств печати важных фрагментов Flash-узлов.

Печать Web-содержимого может привести к нежелательным и непредсказуемым результатам. Прежде всего, Web-содержимое предназначено для отображения в Web, а не для печати. Особенно это касается содержимого Flash. Вложенная сущность большей части Flash-содержимого может привести к тому, что процедура печати покажется совсем непростой операцией. Поэтому очень важным этапом является подготовка Flash-содержимого (по крайней мере, его части) к печати.

ПОДГОТОВКА ФАЙЛОВ к ПЕЧАТИ

Содержимое Flash можно распечатывать из Flash-плеера двумя способами: из контекстного меню внутри Flash-плеера или посредством действия печати, как правило, **щелчка** на кнопке, из Flash-фильма. Доступ к контекстному меню (рис. 29.1) можно получить, либо щелкнув мышью и удерживая при этом нажатой клавишу <Ctrl> (Macintosh), либо щелкнув правой кнопкой мыши (Windows) на Flash-фильме в окне браузера.

Процедура печати посредством контекстного меню имеет определенные ограничения. Распечатать можно только кадры основной временной шкалы. Вложенные видеоклипы, а также эффекты прозрачности и применения цветов не распечатываются. На данный способ печати можно полагаться только в том случае, если фильм имеет одну **временную** шкалу либо если подлежащее печати содержимое расположено исключительно в основной временной шкале и к нему нельзя применить цветовые эффекты.

Для достижения наилучших результатов постарайтесь спланировать, какое содержимое можно будет выводить на печать, еще на этапе компоновки фильма. Если вы планируете разрешить печать только определенного содержимого, убедитесь, что его можно будет надлежащим образом распечатать на бумаге нужного формата. При публикации фильма возможности регулировать параметры размеров, масштабирования и выравнивания **HTML-элементов** уже не будет.

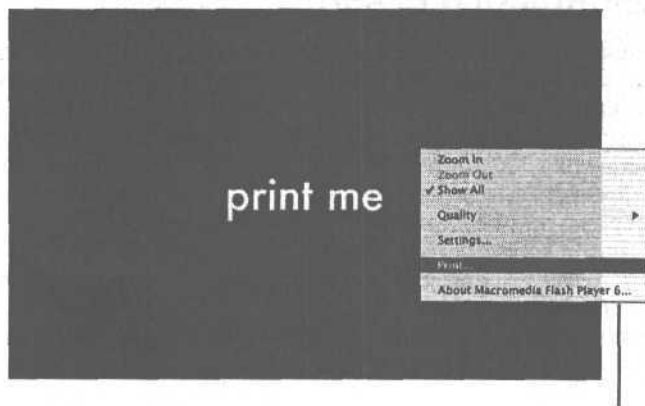
Совет

Подробно о параметрах публикации рассказывается в главе 30 "Оптимизация, публикация и экспортирование фильмов".

НАЗНАЧЕНИЕ МЕТОК КАДРОВ

Flash позволяет указать, какие кадры фильма следует распечатывать. По умолчанию распечатываются все кадры основной временной шкалы. Однако их число можно ограничить, присвоив метки только тем кадрам, которые вы хотите сделать доступными для печати. Эта возможность позволяет сделать процедуру печати более дружелюбной по отношению к пользователю. Если помимо традиционного статического содержимого в фильме содержится и мультимедиа, можно указать, чтобы для печати были доступны только статические элементы. Нет совершенно никаких причин раздражать пользователей, позволяя им распечатывать содержимое, которое все равно не будет распечатано хорошо.

Чтобы указать, какие из выделенных кадров должны быть доступными для печати, выполните следующие действия.



Контекстное меню

Рис. 29.1. Одним из способов печати является выбор опции Print из контекстного меню Flash-плеера

1. Выделите кадр, который необходимо сделать доступным для печати, щелкнув на нем в основной временной шкале фильма.
2. Выделив кадр, щелкните в поле Frame (Кадр) на панели инспектора свойств и введите метку **#p**, как показано на рис. 29.2.

На заметку

Если кадр, который вы не хотите распечатывать, следует за кадром, помеченным для печати, то, для того, чтобы была сброшена метка **#p**, в последующие нераспечатываемые кадры необходимо вставить пустой ключевой кадр.

ОТКЛЮЧЕНИЕ ФУНКЦИИ ПЕЧАТИ

Чтобы отключить функцию печати фильма, необходимо воспользоваться контекстным меню, так как возможности отключить печать с помощью кнопки Print (Печать) браузера не существует. Задачу по отключению печати можно также выполнить посредством меток кадров. Чтобы предотвратить печать фильма, выполните следующие действия.

1. Выделите кадр в основной временной шкале.
2. Выделив кадр, щелкните в поле Frame (Кадр) на панели инспектора свойств и введите метку **!#p**.
3. Протестируйте фильм и убедитесь, что в его контекстном меню команда Print не является активной (рис. 29.3).



Рис. 29.2. Кадры, подлежащие печати, назначаются путем присвоения им метки **#p**

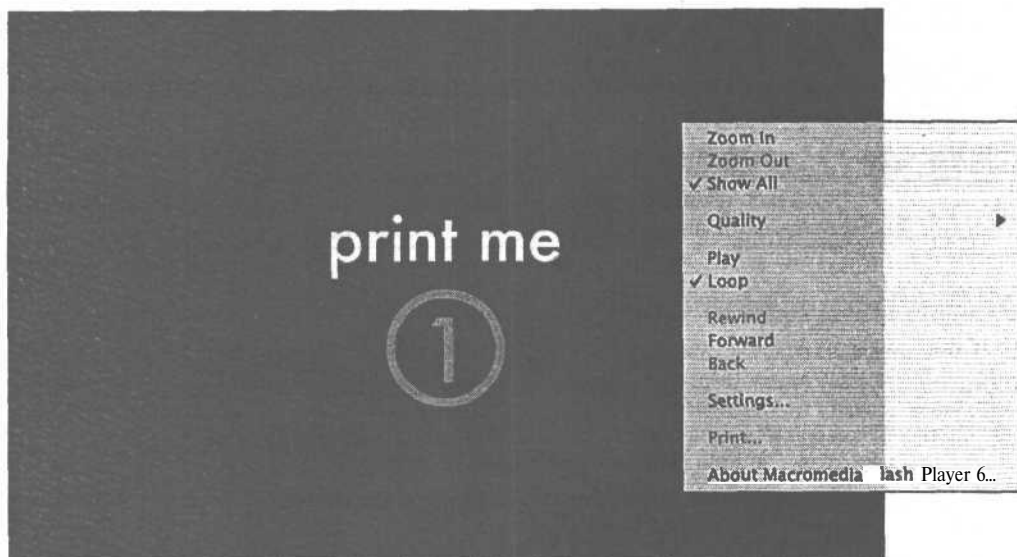


Рис. 29.3. Для отключения функции печати назначьте для кадра метку !#p. При этом будет отключена опция печати в контекстном меню Flash-плеера

Функцию печати можно отключить и в диалоговом окне Publish Settings (Параметры публикации), указав, что контекстное меню Flash-плеера отображать не нужно. Для открытия диалогового окна Publish Settings выполните команду **File** ⇒ **Publish Settings**, а затем выберите вкладку HTML. В группе опций Playback (Воспроизведение) снимите флажок с поля опции Display Menu (Отображение меню), как показано на рис. 29.4. Однако имейте в виду, что при этом не будут отображаться все элементы контекстного меню Flash-плеера.

ИЗМЕНЕНИЕ РАСПЕЧАТЫВАЕМОГО Фонового ЦВЕТА

Flash-плеер распечатывает фоновый цвет, заданный в диалоговом окне Document Properties (Свойства документа). Однако, если для печати доступно ограниченное число кадров, то их можно распечатывать так, что фоновый цвет будет отличаться от фонового цвета остальных кадров фильма. Для изменения выводимого на печать фонового цвета выполните следующие действия.

1. Создайте новый слой под всеми слоями документа, в которых были **выделены** кадры, предназначенные для печати. Присвойте этому слою имя **background color** (фоновый цвет).
2. В кадр, помеченный для печати в слое фонового **цвета**, вставьте ключевой кадр.
3. Выберите инструмент Rectangle (Прямоугольник) и укажите цвет заливки, который будет использоваться в качестве фонового. В слое фонового цвета нарисуйте контур, размер которого будет больше рабочего поля (рис. 29.5).
4. Если последующие кадры не помечены для печати, не забудьте вставить пустые ключевые кадры. В противном случае в них проявится фоновый цвет кадров, подлежащих печати.



Рис. 29.4. Чтобы опубликовать фильм без контекстного меню Flash-плеера, в диалоговом окне Publish Settings снимите флажок с поля опции Display Menu

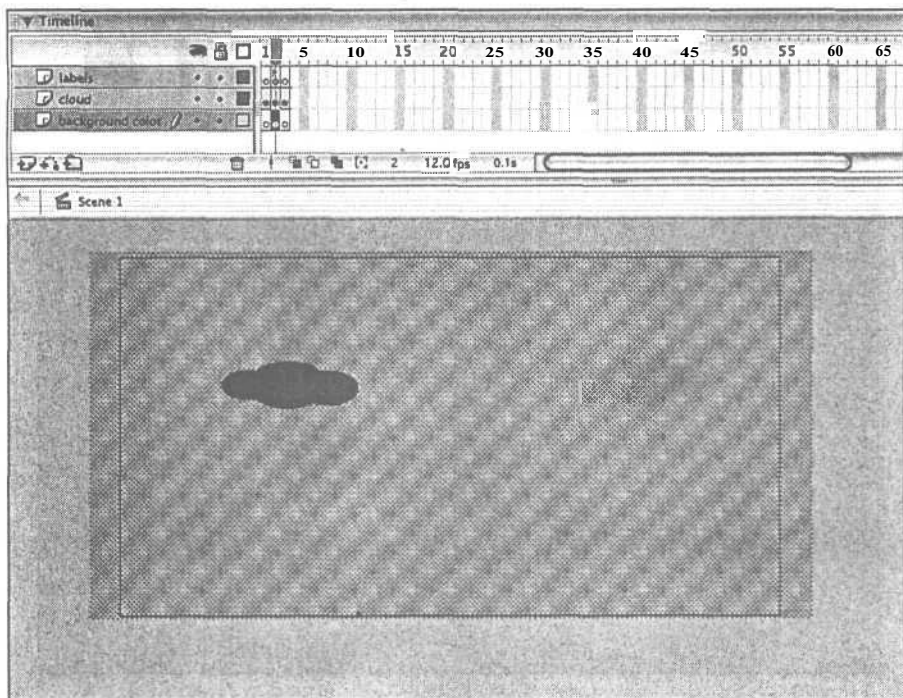


Рис. 29.5. Для изменения фонового цвета, выводимого на печать, нарисуйте покрывающий все рабочее поле контур, залитый нужным цветом

ОПРЕДЕЛЕНИЕ ОБЛАСТИ ПЕЧАТИ

Flash позволяет указать область печати внутри фильма. По умолчанию область печати определяется границами рабочего поля, но можно задать и три другие области.

- Чтобы в качестве области печати назначить отдельное окошко в любом распечатываемом кадре, в любом кадре назначьте это окошко, присвоив кадру метку **#b**.
- Чтобы для всех распечатываемых кадров в качестве области печати назначить составное окошко, задайте аргумент **Max** для действия печати.
- Чтобы задать окошко для каждого кадра, для действия печати назначьте параметр **Frame**.

Для назначения окошка печати выполните следующие действия.

1. Откройте новый файл и с одной стороны рабочего поля нарисуйте простой контур, как показано на рис. 29.6.

Если вы хотите опубликовать и распечатать фильм на этом этапе, то область печати будет равна размерам рабочего поля фильма (рис. 29.7).

2. Над текущим слоем фильма добавьте еще два слоя. Для этого щелкните на кнопке **Insert Layer** (Вставить слой), расположенной внизу списка слоев.
3. Присвойте среднему слою имя **border** (граница). Выберите инструмент **Rectangle** (Прямоугольник) и в исходном слое нарисуйте прямоугольник, имеющий контур, но не имеющий заливки (рис. 29.8).
4. Присвойте верхнему слою имя **labels** (метки). На панели инспектора свойств в качестве метки кадра укажите **#b** (рис. 29.9).

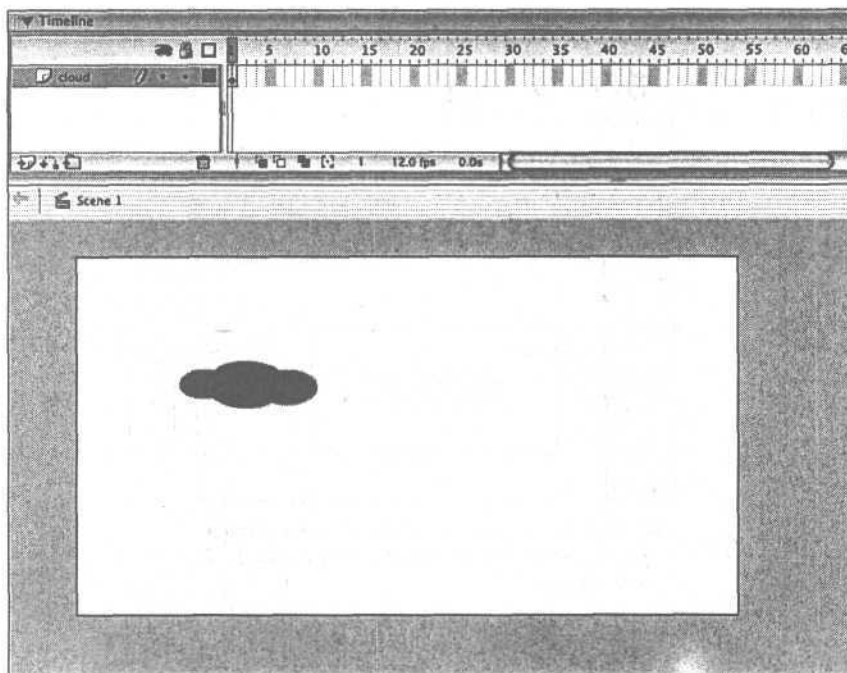


Рис. 29.6. Процедуру назначения окошка печати начните с помещения содержимого в кадры, подлежащие печати

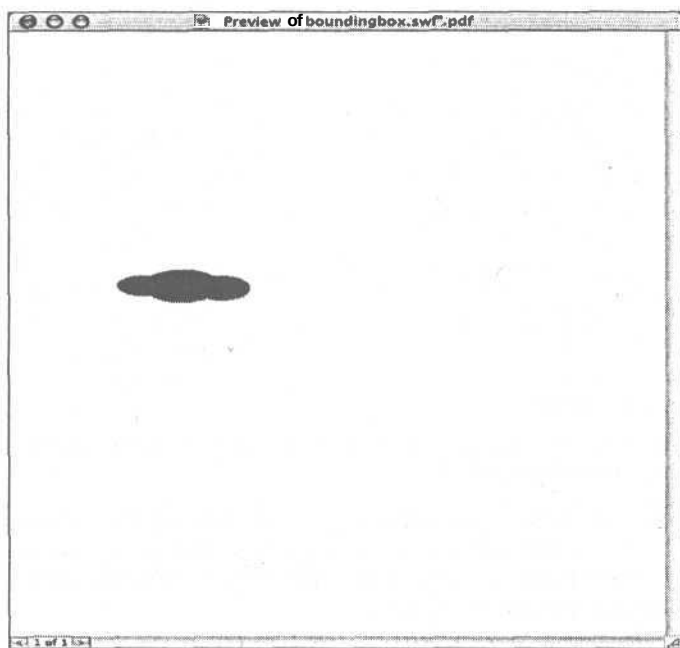


Рис. 29.7. Если размеры ограничительного окошка не указаны, то область печати будет определяться размерами рабочего поля фильма

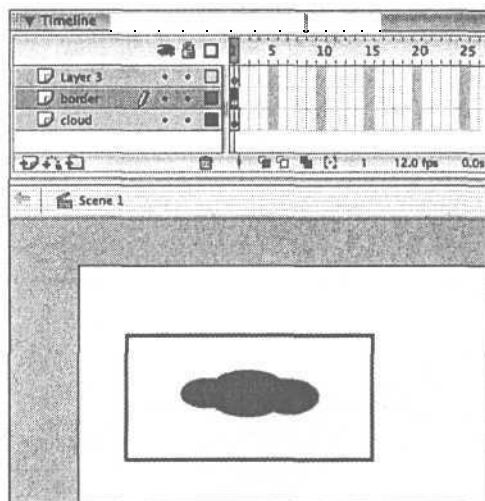
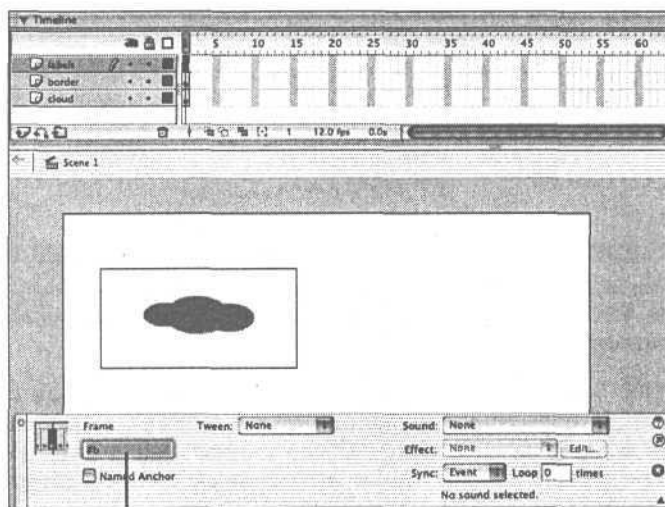


Рис. 29.8. Контур внутри кадра можно назначить в качестве ограничительного окошка области печати



Назначенная метка кадра

Рис. 29.9. Для создания окошка, ограничивающего область печати, для кадра задайте метку #B

5. Сохраните файл под именем **bounding1 fla** и протестируйте фильм. Для тестирования печати откройте SWF-файл в браузере и из контекстного меню Flash-плеера выберите команду Print. Кадр должен быть масштабирован так, чтобы созданная граница заполняла область печати (рис. 29.10).

Чтобы создать составное ограничительное окошко, в котором для определения общей области печати используются объекты всех распечатываемых слоев, выполните следующие действия.

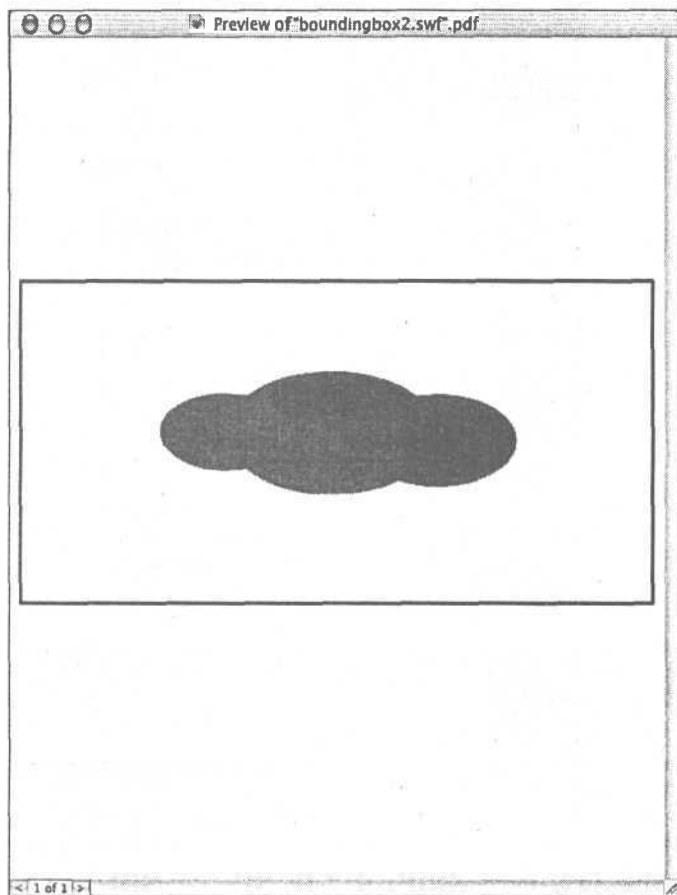


Рис. 29. Ю. Когда кадр назначен в качестве ограничительного окошка, его содержимое масштабируется так, чтобы заполнить область печати

1. Откройте файл **bounding1.fla**. В слое **border** щелкните рядом с именем правой кнопкой мыши (Windows) или щелкните мышью, удерживая нажатой клавишу <Ctrl> (Macintosh). Из контекстного меню выберите пункт **Delete Layer** (Удалить слой). В данном примере для назначения ограничительного окошка используется **ActionScript**.
2. Во всех оставшихся слоях выделите кадр 3 и вставьте пустые кадры нажав клавишу <F5>.
3. Щелкните в кадре 2 слоя, содержащего контур, а затем выделите кадры 2 и 3, щелкнув в кадре 3, удерживая при этом клавишу <Shift>. Щелкните правой кнопкой мыши (Windows) или щелкните мышью, удерживая нажатой клавишу <Ctrl> (Macintosh), и из контекстного меню выберите пункт **Covert to KeyFrames** (Преобразовать в ключевые кадры), как показано на рис. 29.11.
4. В слое контура выберите кадр 2. Переместите контур по рабочему полю. Повторите эту же операцию в кадре 3, перемещая контур в разные участки рабочего поля (рис. 29.12).
5. Сохраните файл под именем **bounding2.fla**.

Чтобы завершить создание составного ограничительного окошка, необходимо создать кнопку печати.

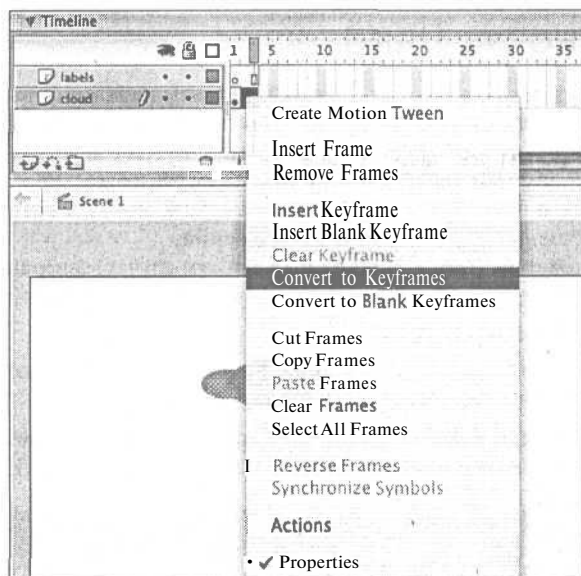


Рис. 29.11. Выделите диапазон кадров, удерживая нажатой клавишу <Shift>. Откройте контекстное меню, щелкнув правой кнопкой мыши (Windows) или щелкнув мышью, удерживая нажатой клавишу <Ctrl> (Macintosh), а затем выберите команду Convert to Keyframes

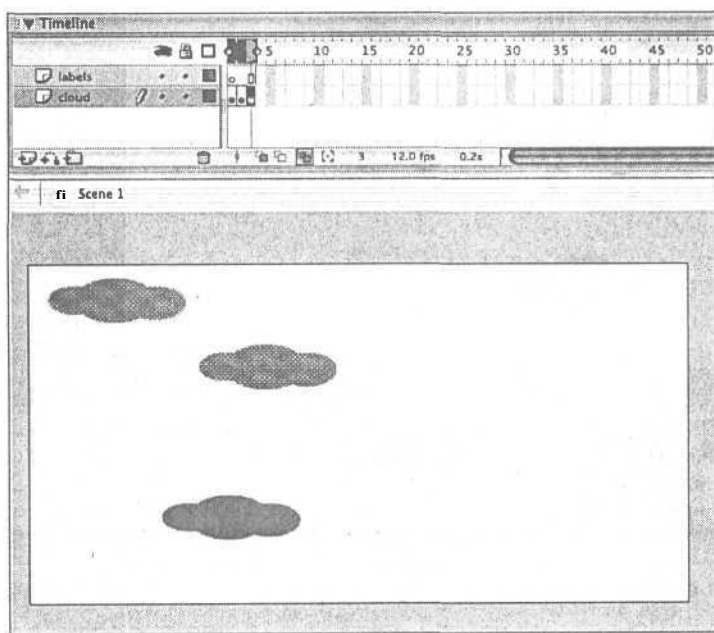


Рис. 29.12. Составное ограничительное окошко будет содержать объекты, находящиеся в различных местах распечатываемых слоев

СОЗДАНИЕ кнопки PRINT

Создание кнопки печати с присоединенным действием печати позволяет в максимальной степени управлять процедурой печати фильма. Действие Print позволяет распечатывать кадры, располагающиеся в нескольких временных шкалах. Каждое действие Print применяется к одной временной зависимости, однако можно использовать несколько кнопок печати. На печать будут выводиться и цветовые эффекты, а кроме того, имеется возможность распечатывать кадры в виде векторных или растровых изображений.

Чтобы добавить действие Print к фильму и (в данном случае) завершить создание составного ограничительного окошка области печати, выполните следующие действия.

- 1*. Откройте файл bounding2 . fla.
2. Вставьте новый слой, щелкнув на кнопке Insert Layer, расположенной внизу перечня слоев. Присвойте слою имя button (кнопка).
3. В данном слое выделите кадр 1 (если вы не хотите выводить на печать каждый кадр, то кадр 1 должен быть ключевым) и создайте кнопку печати (рис. 29.13).

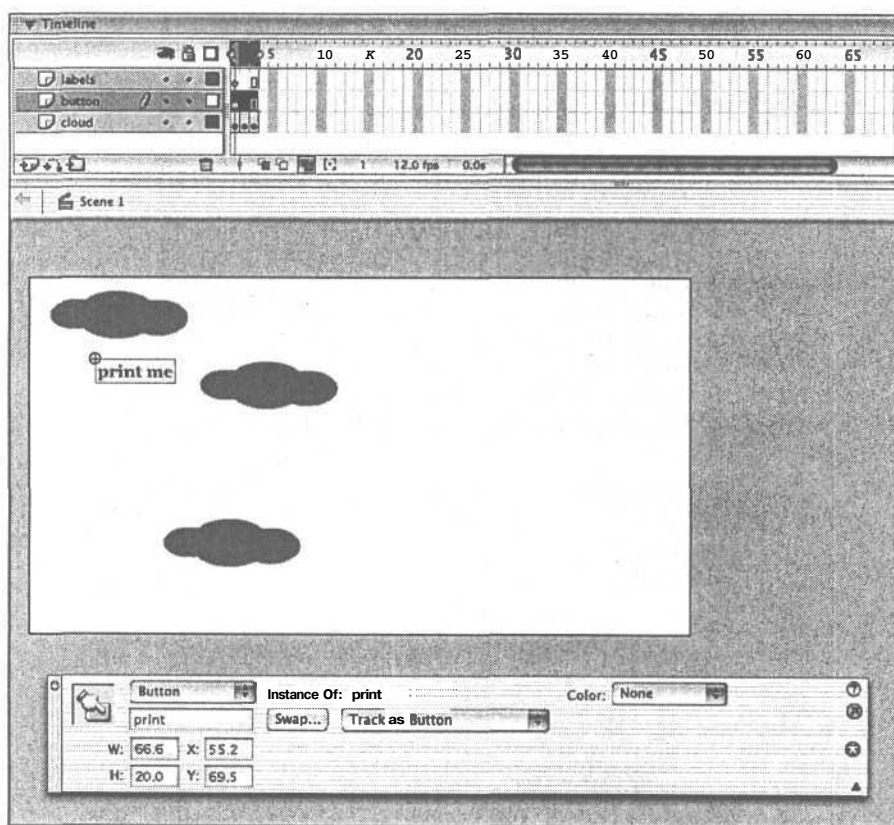


Рис. 29.13. Создайте кнопку печати, к которой можно будет присоединить действие Print

Совет

Подробная информация о кнопках представлена в главе 6 "Символы, экземпляры и элементы библиотек".

1. Выделите экземпляр кнопки и запустите модуль **ActionScript Editor** (Редактор ActionScript), выполнив команду **Window⇒Actions** (Окно⇒Действия) или нажав клавишу <F9>. Из всплывающего меню ActionScript, расположенного в правом верхнем углу панели Actions, выберите пункт **Normal Mode** (Обычный режим).
2. В левой части панели Actions щелкните на папках **Actions** и **Printing** (Печать), после чего дважды щелкните на элементе **Print**, чтобы передать действие Print в окно редактора (рис. 29.14).
3. Из всплывающего меню **Print** (расположенного в правой части панели Actions) выберите пункт **As Vectors** (Как векторы). При выборе данной опции печать будет выполнена с более высоким качеством, но при этом эффекты прозрачности и цветовые эффекты выведены не будут. В качестве альтернативного варианта можно выбрать пункт **As Bitmap** (Как растр), при котором данные эффекты будут выводиться на печать, но если кадр разрешено масштабировать, то качество печати может оказаться более низким.
4. Из раскрывающегося меню **Location** (Местонахождение) выберите **временную** шкалу, кадры которой необходимо распечатать. Выберите элемент **Target** (Целевой элемент) и щелкните на расположенном рядом поле. Введите **this** и установите флажок в поле опции **Expression** (Выражение). В качестве альтернативного варианта можно выбрать пункт **Level** (Уровень), который используется для указания номера уровня основной временной шкалы или загруженного клипа. Номер уровня нужного клипа вводится в поле **Location**. Любое целевое местоположение, как **Level**, так и **Target**, можно трактовать как выражение, установив флажок в поле опции **Expression**.
5. Из раскрывающегося меню **Bounds** (Границы) выберите пункт **Max**. Завершенный сценарий изображен на рис. 29.15. Параметр границ (**Movie** (Фильм), **Max** или **Frame** (Кадр)) определяет ограничительное окошко распечатываемых кадров. При выборе параметра **Movie** в качестве области печати всех кадров будет использоваться ограничительное окошко объекта в кадре с меткой #b. При выборе параметра **Max** для определения составного ограничительного окошка, охватывающего местоположения всех объектов, используются все объекты в распечатываемых кадрах. При выборе параметра **Frame** создаются отдельные ограничительные окошки для объектов в каждом распечатываемом кадре, и объекты в каждом кадре масштабируются так, чтобы заполнить область печати в соответствии с параметром, выбранным из меню **Dimensions** (Размеры) диалогового окна **Publish Settings** (Параметры публикации).

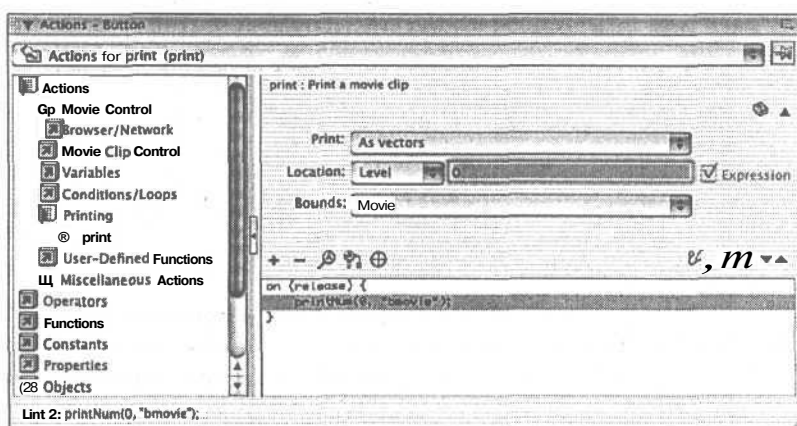


Рис. 29.14. Для доступа к действиям печати в окне ActionScript Editor щелкните на папках **Actions** и **Printing**

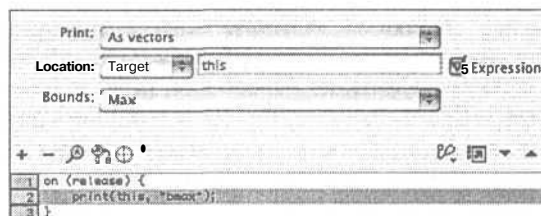
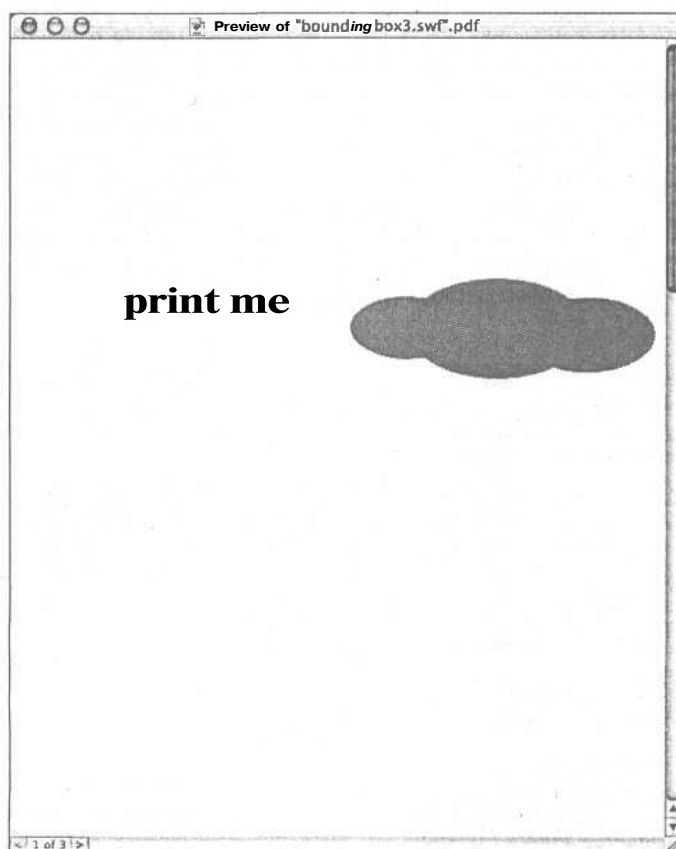


Рис. 29./5. Завершенное действие Print, предназначенное для создания составного ограничительного окошка

1. Выделите кадр 3 в слое labels. Нажмите клавишу <F6>, чтобы вставить ключевой кадр. Откройте окно ActionScript Editor, нажав на клавишу <F9>, и вставьте действие `stop()` ; .
2. Сохраните фильм под именем **bounding3.fl** и протестируйте его. Откройте SWF-файл в браузере и щелкните на кнопке Print. Обратите внимание на то, что каждая страница распечатывается в пределах области, заданной на небольшом участке экрана, а кнопки являются общими для всех кадров (рис. 29.16).

Совет

Подробная информация о загрузке видеоклипов и уровней приведена в главе 17 "Демонстрация мощи видеоклипов". О написании сценариев рассказывается в главе 11 "Знакомство с ActionScript".



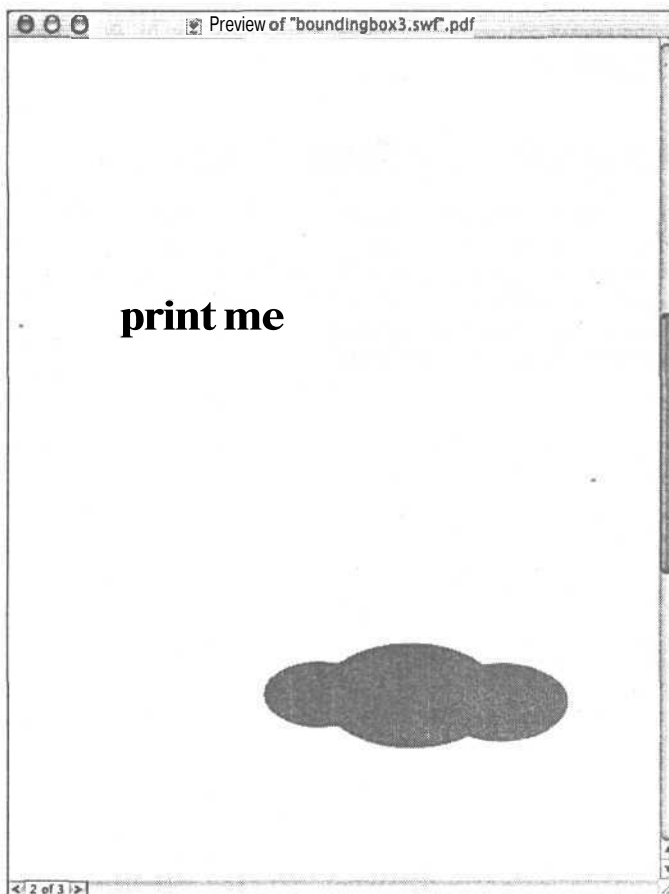


Рис. 29.16. Задание области печати в виде составного ограничительного окошка приводит к тому, что она охватывает местоположение содержимого каждого кадра

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

Почему Flash игнорирует ограничительное окошко, установленное в пределах кадра с меткой, а область печати страниц получается гораздо больше указанной?

Проверьте сценарий **ActionScript** и убедитесь в том, что в пределах действия Print не выбран параметр Max или Frame. При выборе этих параметров ограничительное окошко, заданное в кадре с меткой, будет игнорироваться.

Несколько пользователей сообщили, что они не могут распечатать содержимое моего узла. Я не могу выяснить причину ошибки.

Пользователи, на компьютерах которых установлены версии Flash-плеера, предшествующие 4.0.20 (Macintosh) и 4.0.25 (Windows), не смогут распечатать содержимое узла. Если вы предполагаете, что у посетителей вашего узла могут быть установлены старые версии плееров, добавьте к своему фильму сценарий обнаружения плеера.

FLASH ЗА РАБОТОЙ: НЕВИДИМЫЕ ЧЕРНИЛА

Действие Print позволяет даже распечатывать видеоклипы, параметр видимости которых установлен в значение `false`. Это означает, что видеоклипы, существующие в пределах фильма, во время воспроизведения видны не будут. Для их вывода на печать в действии Print требуется указать целевое местоположение видеоклипа. Преимущество данного решения заключается в возможности сохранения предыдущих элементов экрана при одновременном обеспечении доступного для печати содержимого.

ОПТИМИЗАЦИЯ, ПУБЛИКАЦИЯ И ЭКСПОРТИРОВАНИЕ ФИЛЬМОВ

В ЭТОЙ ГЛАВЕ...

Тестирование фильмов	669
Оптимизация фильмов	671
Задание параметров публикации	673
Экспортирование различных форматов файлов	684
Возможные проблемы	686
Flash за работой: использование именованных анкеров	686

ТЕСТИРОВАНИЕ Фильмов

Завершив проектирование и разработку Rash-фильма, очень важно его протестировать и **посмотреть**, в каком виде данный фильм увидят пользователи. Если этому не уделить должного внимания, то они могут и не получить доступа к содержимому. К счастью, Flash позволяет легко смоделировать представление узла пользователям и обнаружить потенциальные проблемные области.

ИСПОЛЬЗОВАНИЕ МОДУЛЯ BANDWIDTH PROFILER

Модуль Bandwidth Profiler (**Профайлер** полосы пропускания) позволяет протестировать фильм, моделируя поток содержимого таким образом, как если бы он поступал через Internet. Эта функция позволяет выявить проблемные области фильма и выяснить, какое содержимое необходимо оптимизировать. Целью выполнения данной операции является достижение максимально возможного качества при минимально возможном размере файла.

Для доступа к модулю Bandwidth Profiler выполните следующие действия.

1. Протестируйте фильм, выполнив команду **Control⇒Test Movie** (**Управление⇒Тестировать фильм**) или нажав комбинацию **<Cmd+Return>** (Macintosh) либо **<Ctrl+Enter>** (Windows). При этом будет создан SWF-файл, который откроется в новом окне.
2. Выполните команду **View⇒Bandwidth Profiler** (рис. 30.1).

Окно модуля Bandwidth Profiler откроется над SWF-файлом, как показано на рис. 30.2. В его левой части указаны данные о фильме, такие как его размеры, частота смены кадров, длительность выполнения предварительной загрузки первого кадра; здесь же указывается и скорость модема. В правой части окна Bandwidth Profiler указывается диаграмма, на которой каждый кадр фильма представлен в виде отдельного столбца. Высота столбца соответствует размеру файла каждого кадра. Горизонтальная красная линия, проведенная на уровне 400 В (см. рис. 30.2), представляет отметку, в которой при заданной скорости модема поток содержимого будет выполняться в режиме реального времени. Любой кадр, столбец которого находится над этой линией, не будет воспроизводиться до тех пор, пока его содержимое не загрузится.



Рис. 30.1. Протестируйте фильм и выполните команду **View⇒Bandwidth Profiler**, чтобы смоделировать поток Flash-фильма

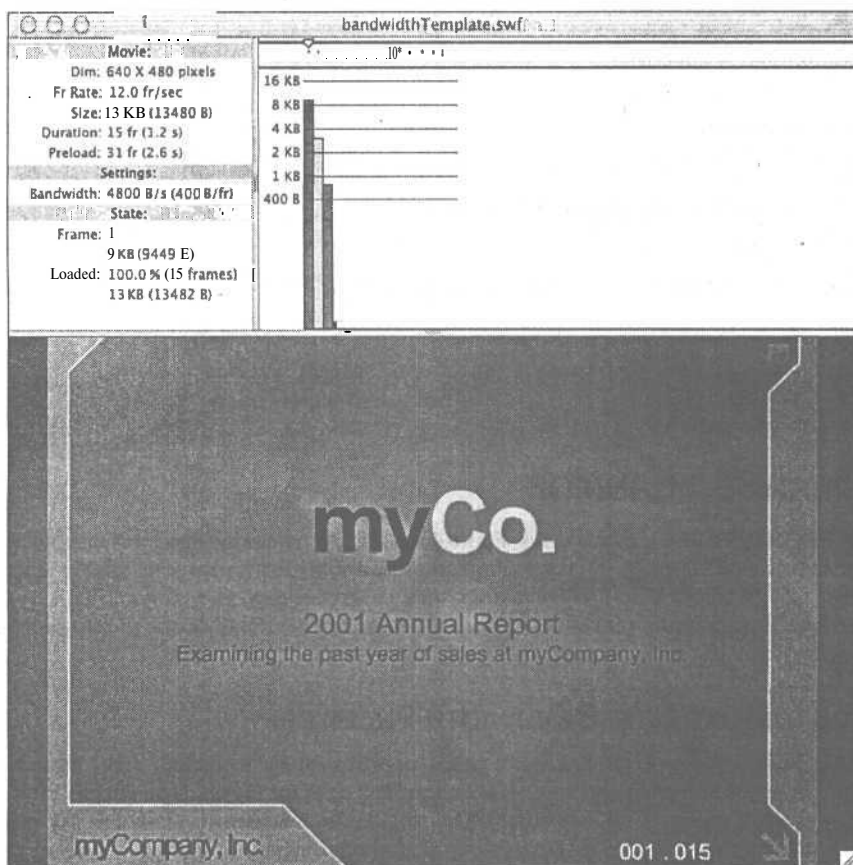


Рис. 30.2. Модуль Bandwidth Profiler моделирует поток содержимого при разных скоростях модема

Для изменения скорости модема, указанной в левой части окна Bandwidth Profiler, из меню **Debug** (Отладка) выберите нужную скорость (рис. 30.3). На сегодняшний день минимальная скорость соединения в большинстве случаев составляет 56 К.

Кроме того, имеется возможность предварительно просмотреть воспроизведение фильма, причем его содержимое будет появляться по мере загрузки. Для отображения потока содержимого выполните команду **View⇒Show Streaming** (Вид⇒Показать содержимое). Фильм будет воспроизводиться так, как если бы он загружался с помощью модема, скорость которого выбрана посредством меню **Debug**. В левой части окна Bandwidth Profiler указывается процент загруженной части фильма, а над диаграммой появляется зеленая полоса, указывающая на процесс загрузки.

Обратите внимание на кадры, в которых поток достигает пиковых величин. Эти кадры в окне Bandwidth Profiler изображены в виде самых высоких столбцов. Содержимое этих кадров необходимо перепроверить и выяснить, можно ли оптимизировать их с целью уменьшения требуемого потока. После щелчка на столбце диаграммы под окном Bandwidth Profiler будет отображено содержимое данного кадра. Если диаграмма имеет много пиков, особенно в начале фильма, необходимо выполнить предварительную загрузку и указать пользователям на то, что выполняется загрузка содержимого.



Рис. 30.3. Для выбора скорости модема воспользуйтесь меню **Debug**

ИСПОЛЬЗОВАНИЕ ОТЧЕТА о РАЗМЕРАХ для ОПТИМИЗАЦИИ ФАЙЛОВ

Еще одним способом оценки размера элементов фильма является создание отчета о размерах. Он представляет собой текстовый файл, который можно создать при публикации фильма. При этом фильм можно разбить покадрово, на отдельные сцены, символы и встроенные шрифты (рис. 30.4).

Чтобы создать отчет о размерах, установите флажок в поле опции **Generate Size Report** (Создать отчет о размерах) диалогового окна **Publish Settings** (Параметры публикации). В этом случае при публикации фильма будет создан текстовый файл, который сохраняется в том же самом каталоге, что и файлы **FLA** и **SWF**. Текстовый файл имеет такое же имя, как и **SWF**-файл (рис. 30.5).

Совет

Подробная информация об использовании диалогового окна **Publish Settings** будет представлена ниже, в подразделе "Задание параметров публикации".

ОПТИМИЗАЦИЯ Фильмов

После выявления "громоздких" элементов фильма вернитесь к соответствующим файлам **FLA** и попытайтесь их оптимизировать. На размер файла, в первую очередь, влияют такие элементы, как растровые изображения, звуки и видео.

Для оптимизации растровых изображений проверьте, чтобы они были импортированы с используемыми в конкретном фильме размерами. Даже если во Flash растровое изображение промасштабировать до меньшего размера, то размер файла все равно останется таким же, как и при импортировании. При необходимости изображение можно обрезать в соответствующем внешнем редакторе и заменить рисунок в фильме.

Используя при работе с растровыми изображениями технологии анимации приводят к существенному увеличению размера файла. Постарайтесь минимизировать использование больших растровых изображений, особенно в анимации, и ограничьте использование эффектов прозрачности. При выполнении трассировки растрового изображения обязательно выполните команду **Modify⇒Curves⇒Optimize** (Изменение⇒Кривые⇒Оптимизировать), чтобы упростить трассировку и уменьшить размер файла.

bandwidthTemplate.swf Report			
Movie Report			
Frame	Frame i	Frame Bytes	Total Bytes Page
1	9449	9449	Scene 1
2	3226	12675	2
3	781	13456	3
4	2	13458	4
5	2	13460	5
6	2	13462	6
7	2	13464	7
8	2	13466	8
9	Г	13468	9
10	Г	13470	10
11	Г	13472	11
12	Г	13474	12
13	Г	13476	13
14	2	13478	14
15	Г	13480	15
Page			
		Shape Bytes	Text Bytes
Scene 1		566	1447
Page			
		Symbol Bytes	Text Bytes
glt, lensFlare		771	0
mc, empty		0	0
slide, background		158	0
btn, print		380	0
mc, frameNav		0	96
mc, prevBtn		0	0
btn, prevBtn		245	0
mc, nextBtn		0	0
btn, nextBtn		247	0
btn, continue		0	0
btn, back		0	0
Sound name			
		Bytes	Format
Arial Italic		2133	.OWabodeIghiklmnopqrstuvwxyz
Arial Bold		1006	%.12578Cmoy
Arial		4879	()_0123456789ABCDEFGHIJOPQRSTWabodeIghiklmnopqrstuvwxyz

Рис. 30.4. В отчете о размерах указываются размеры файлов отдельных элементов фильма

Отчет о размерах, сохраненный как SWF-файл

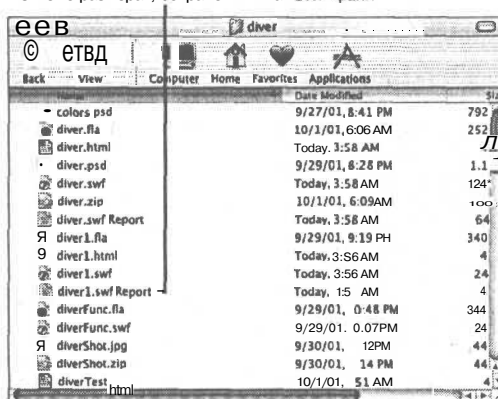


Рис. 30.5. Файл с отчетом о размерах генерируется в процессе публикации фильма и сохраняется в том же каталоге, что и SWF-файл

Подробно об импортировании и оптимизации растровых изображений рассказывается в главе 4 "Использование растровых изображений во Flash".

Использование символов способствует уменьшению размера файла, поэтому проверьте, чтобы каждый повторно используемый элемент фильма был преобразован в символ. При выполнении анимации разнесите статические и анимированные элементы в отдельные слои. В ключевых кадрах ограничьте выполняемые операции как можно меньшим участком рабочего поля. И по возможности максимально используйте анимацию с построением промежуточных отображений, поскольку при применении этой методики размер файла получается меньшим по сравнению с покадровой анимацией, представляющей собой последовательный ряд кадров.

Работая с векторной графикой, постарайтесь не использовать штрихи, имеющие вид пунктирных линий, и штрихи, имеющие тот же цвет, что и заливка. Градиентная и растровая заливка также увеличивают размер файла, поэтому применяйте их осторожно.

Ограничьте число используемых шрифтов и стилей. Каждый внедренный символ и шрифт вносят свой вклад в увеличение размера фильма. Поэтому внедряйте только те символы, которые действительно необходимы.

Для оптимизации звуков отредактируйте их до нужной длины до выполнения импортирования. Максимизируйте повторное использование звука, если это возможно. Постарайтесь применять MP3-сжатие, поскольку оно позволяет достичь наименьшего размера файла.

Подробная информация об оптимизации анимации приведена в главе 7 "Анимация во Flash". Приемы оптимизации текста описаны в главе 5 "Работа с текстом", а процедура оптимизации звука — в главе 8 "Использование звука".

ЗАДАНИЕ ПАРАМЕТРОВ ПУБЛИКАЦИИ

После составления и оптимизации фильма его следует опубликовать. Процедура публикации позволяет создать несколько форматов фильма, помимо SWF- и HTML-файлов с кодом, необходимым для отображения фильма через Internet.

Для доступа к параметрам публикации выполните команду **File⇒Publish Settings** (Файл⇒Параметры публикации), как показано на рис. 30.6.



Рис. 30.6. Для доступа к диалоговому окну **Publish Settings** выполните команду **File⇒Publish Settings**

Диалоговое окно Publish Settings имеет три вкладки: Formats (Форматы), Flash и HTML. На рис. 30.7 видно, что во вкладке Formats можно указать дополнительные форматы файла, помимо создаваемых по умолчанию SWF и HTML.

Для выбора альтернативного формата установите флажок в поле соответствующей опции. Во вкладке Formats можно выбрать следующие форматы: Flash, HTML, GIF, JPEG, PNG, Windows Projector, Macintosh Projector и Quick Time. В процессе публикации будут созданы файлы указанных форматов с теми же именами, что и файл FLA, и с соответствующими расширениями. Если вы хотите присвоить файлам другие имена, снимите флажок с поля опции Use Default Names (Использовать имена по умолчанию) и в соответствующие поля введите новые имена. При выборе форматов (за исключением проекторов) в диалоговом окне Publish Settings появится новая вкладка для данного формата. Открыв эту вкладку, можно задать те или иные параметры для выбранного формата.

ФОРМАТ FLASH

При выборе формата Flash (он выбирается по умолчанию) выполняется экспортирование SWF-файла. Если вы хотите увидеть SWF-файл в Web, то в браузере должен быть установлен Flash-плеер. Во вкладке Flash (рис. 30.8) можно выбрать версию плеера. Можно выбрать одну из предыдущих версий, но тогда некоторые функциональные возможности будут утеряны. Если содержимое фильма не поддерживается предыдущими версиями плееров, на экране появится предупреждающее сообщение.

Опция Load Order (Порядок загрузки) определяет порядок, в котором загружаются слои каждого кадра: либо сверху вниз (Top Down), либо снизу вверх (Bottom Up). Эта функция может оказаться важной при просмотре больших файлов посредством медленных модемов. Поэтому вам стоит оценить структуру слоев и определить, какой порядок загрузки лучше всего подходит для того или иного фильма.

При выборе опции Generate Size Report (Создать отчет о размерах) создается отчет с указанием размеров файлов отдельных элементов фильма. Опция Protect from Import (Защитить от импор-

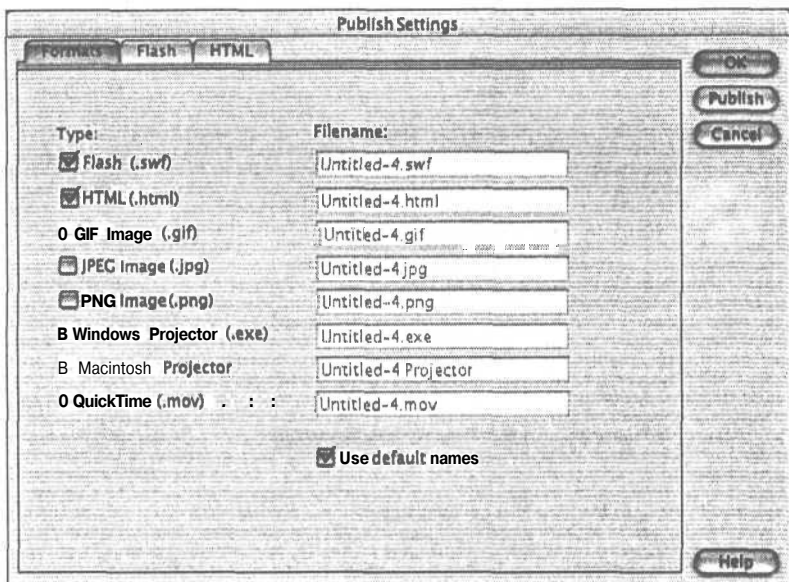


Рис. 30.7. При публикации можно выбрать альтернативные форматы Flash-фильма

тирования) запрещает пользователям загружать фильм и импортировать его во Flash в виде доступного для редактирования файла FLA. Опция Omit Trace Actions (Опустить действия трассировки) при публикации фильма удаляет любые действия трассировки из кода ActionScript. Действия трассировки указываются в окне вывода данных и используются для отладки.

Опция Debugging Permitted (Отладка разрешена) позволяет выполнять удаленную отладку Flash-фильмов. Чтобы защитить фильм в случае разрешения отладки, вы можете (и должны) потребовать от пользователя ввести пароль для доступа к операции отладки. Для того чтобы создать и затребовать пароль, введите его в поле Password (Пароль).

Если из раскрывающегося меню Version (Версия) выбран пункт Flash Player 6, то для сжатия фильма можно установить флажок в поле опции Compress Movie (Сжать фильм). Если версия Flash Player 6 выбрана изначально, то флажок в поле опции Compress Movie установлен по умолчанию. Процедура сжатия файла формата Flash 6 очень важна и может существенно уменьшить размер файла, особенно в тех случаях, когда в нем содержится большое количество текста и кодов сценария.

Ползунок JPEG Quality (Качество JPEG) определяет степень сжатия растровых изображений. Чем выше данная настройка, тем выше качество, но при этом размер файла также будет большим. Поэкспериментируйте с данным ползунком и найдите оптимальное соотношение качества и размера файла.

Совет

Подробная информация о сжатии растровых изображений приведена в главе 4 "Использование растровых изображений во Flash".

Для задания глобального сжатия потоковых или событийных звуков щелкните на кнопке Set (Задать), расположенной справа от опции Audio Stream (Аудиопоток) или Audio Event (Аудиособытие), и укажите тип сжатия, скорость передачи битов и качество. Чтобы выбранные настройки превалировали над любыми отдельными настройками, указанными для звуковых элементов на панели инспектора свойств, установите флажок в поле опции Override Sound Settings (Подменить настройки звука).

Совет

Подробно о сжатии звука рассказывается в главе 8 "Использование звука".

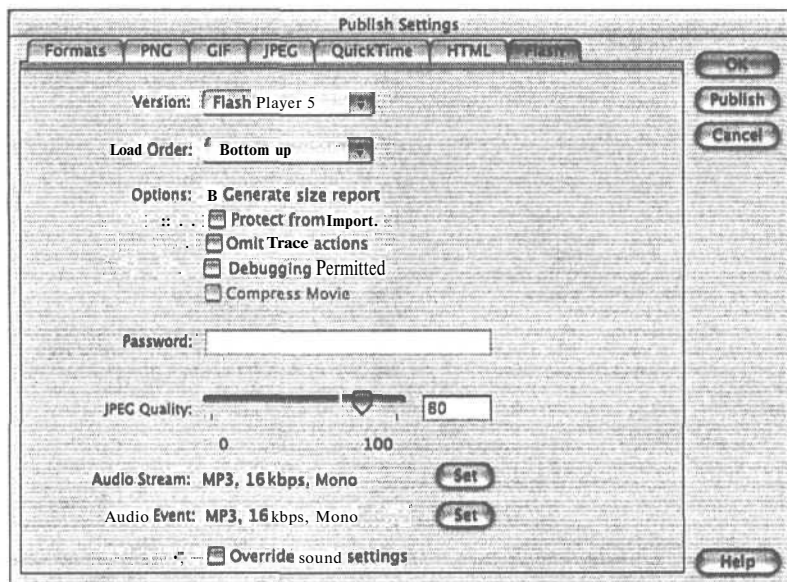


Рис. 30.8. При публикации в формате Flash создается SWF-файл

ФОРМАТ HTML

Чтобы Flash-фильмы можно было отобразить в браузере, их необходимо внедрить в HTML-документы. Опции вкладки HTML (рис. 30.9) позволяют указать, каким образом SWF-файл должен быть отображен в браузере.

Flash создает HTML-документ на основе шаблона, который можно выбрать из раскрывающегося меню Template (Шаблон), изображенного на рис. 30.10. Шаблоны включают в себя функции обнаружения различных версий Flash, устройств и именованных анкеров, позволяющих переходить от одного кадра фильма к другому посредством кнопок Forward (Вперед) и Back (Назад) браузера.

Из раскрывающегося меню Dimensions (Размеры) можно выбрать величины, в которых будут указаны атрибуты дескрипторов ширины и высоты, размещенные в HTML-документе. При выборе опции Match Movie (Соответствует фильму) размеры будут совпадать с размерами фильма, и в поля Width (Ширина) и Height (Высота) нельзя будет вводить значения вручную. Выбрав опцию Percentage (Процент) или Pixels (Пиксели), вы сможете вводить значения в поля Width и Height, модифицируя тем самым размеры фильма.

Опции группы Playback (Воспроизведение) позволяют управлять функциями фильма и способом его воспроизведения. По умолчанию воспроизведение фильма начинается сразу же после его загрузки. При выборе опции Paused at Start (Приостановка в начале) фильм не начнет воспроизводиться до тех пор, пока пользователь не щелкнет на кнопке в пределах фильма или не выберет пункт Play из контекстного меню плеера. По умолчанию флажок в поле данной опции не установлен.

При выборе опции Loop (Цикл) по завершении воспроизведения фильма воспроизводящая головка вернется к кадру 1, и фильм будет повторяться. По умолчанию функция циклического воспроизведения включена. При выборе опции Device Font (Системный шрифт) в блоках статического текста, шрифт которых помечен как системный, выполняется замена шрифтов, не поддерживаемых системой.

Элементы раскрывающегося меню Quality (Качество) определяют приоритет воспроизведе-

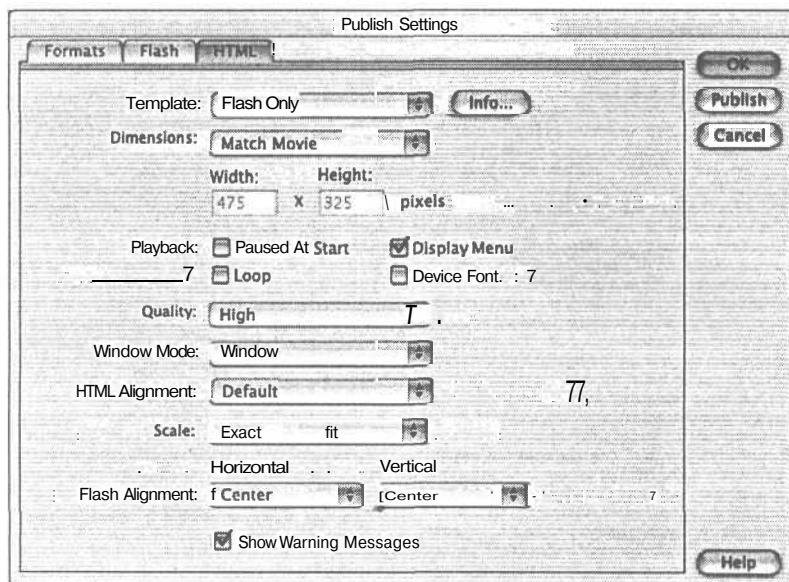


Рис. 30.9. При публикации фильма посредством опций вкладки HTML генерируется HTML-файл, указывающий, каким образом фильм должен отображаться в браузере

ния или качества отображения. Обычно используется заданный по умолчанию параметр High (Высокое). При выборе параметра Low (Низкое) преимущество отдается воспроизведению в ущерб качеству, и при этом отключается функция сглаживания. Параметр Auto Low аналогичен Low, но при его выборе производится попытка улучшить качество изображения. К примеру, может быть включена функция сглаживания, если Flash определит, что процессор пользователя сможет обработать ее. При выборе опции Medium (Среднее) функция сглаживания применяется, однако она не затрагивает растровых изображений. При выборе опции Best (Наилучшее) приоритет отдается качеству отображения без оглядки на воспроизведение. При этом устраняются контурные неровности всего текста и сглаживаются растровые изображения.

Пользователи браузера Internet Explorer 4.0 для Windows с элементами управления Rash ActiveX могут выбрать из раскрывающегося меню Window Mode (Режим окна) параметры прозрачности, размещения и разбиения на слои. При выборе опции Window (Окно) Flash-фильм будет отображен в своем собственном окне для реализации самой быстрой анимации. При выборе опции **Opaque Windowless** (Непрозрачный без окна) фильм не отображается в окне, а любые элементы HTML-страницы, которые могли быть размещены сзади окна Flash, перемещаются таким образом, чтобы они не были видны через прозрачные участки Flash-фильма. При выборе параметра Transparent Windowless (Прозрачный без окна) любые фоновые HTML-элементы будут видны через прозрачные участки Flash-фильма, что может существенно замедлить анимацию.

Элементы раскрывающегося меню HTML Alignment (Выравнивание HTML) определяют выравнивание фильма относительно окна браузера. По умолчанию фильм располагается по центру окна браузера, а если окно браузера меньше фильма, то фильм обрезается. При выборе опций Left (По левому краю), Right (По правому краю), Top (По верху) и Bottom (По низу) фильм выравнивается относительно выбранного края окна браузера.

Элементы раскрывающегося меню Scale (Масштабирование) позволяют разместить фильм в пределах выбранной области окна браузера. По умолчанию фильм располагается в пределах назначенной области HTML-документа. При этом его пропорции сохраняются без искажения. При выборе элемента No Border (Без границы) фильм заполняет всю назначенную область. При необходимости он может быть масштабирован или обрезан, но его исходные пропорции сохраняются. При выборе элемента Exact Fit (Точное заполнение) фильм заполняет всю назначенную область без учета исходных пропорций. Параметр No Scale (Без масштабирования) запрещает масштабирование фильма при изменении размеров окна браузера.

Элементы раскрывающегося меню Flash Alignment (Выравнивание Flash) определяют выравнивание фильма относительно его окна. При выборе одного из элементов группы Horizontal (По горизонтали) фильм можно выровнять по левому краю, по правому краю и по центру, а при выборе элементов группы Vertical (По вертикали) фильм можно выровнять по верху, по низу и по центру окна.

Если в поле опции Show Warning Messages (Показывать предупреждающие сообщения) установлен флажок, на экран будут выводиться предупреждающие сообщения в случае конфликта HTML-дескрипторов. Данная функция включена по умолчанию.

ФОРМАТ GIF

Помимо стандартных форматов Flash и HTML, фильмы можно сохранять в виде статичных или анимированных GIF-файлов. Опции вкладки GIF изображены на рис. 30.11.

Чтобы указать размеры GIF-файла, введите их в поле Height или Width либо установите флажок в поле **опции** Match Movie, в результате чего размеры GIF-файла будут совпадать с размерами фильма.

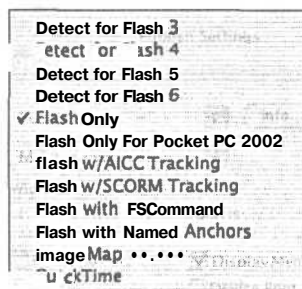


Рис. 30.10. Flash создает HTML-документ на основе одного из шаблонов

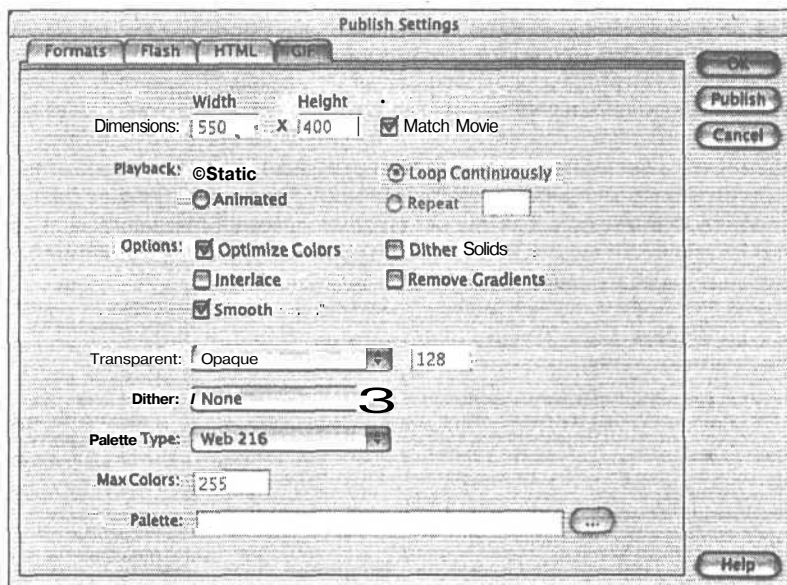


Рис. 30.11. Flash-фильмы можно сохранить в виде статичных или анимированных GIF-файлов

Установите переключатель Playback (Воспроизведение) в поле опции Static (Статическое) или Animated (Анимированное). При выборе опции Static Flash сохранит первый кадр фильма в виде GIF-файла. Чтобы указать, что в виде статического GIF-файла следует экспортировать другой кадр, в файле FLA присвойте этому кадру метку #Static.

При выборе анимированного воспроизведения становятся активными дополнительные опции, позволяющие воспроизводить фильм непрерывно или определенное число раз. По умолчанию Flash экспортирует каждый кадр фильма в виде анимированного GIF-файла. Чтобы указать начальный и конечный кадры, которые следует экспортировать в виде анимированных GIF-файлов, в файле FLA присвойте этим кадрам метки #First (Первый) и #Last (Последний).

Сжатие GIF-изображений осуществляется путем ограничения используемой палитры цветов. Существует несколько опций отображения GIF-файлов.

- Optimize Colors (Оптимизировать цвета). Из таблицы цветов GIF-файла удаляются любые неиспользуемые цвета. Эта опция выбрана по умолчанию.
- Interlace (Чередование). GIF-файл отображается в браузере пошагово, по мере загрузки. Не устанавливайте флажок в поле данной опции при работе с анимированными GIF-файлами.
- Smooth (Сглаживание). К изображению применяется функция устранения контурных неровностей. Эта функция может улучшить внешний вид изображения, но при помешении изображения на цветной фон вокруг него может образоваться серый ореол. Кроме того, процедура сглаживания приводит к увеличению размера файла. Данная опция выбрана по умолчанию.
- Dither Solids (Растривать сплошные цвета). Моделируются цвета, не входящие в цветовую таблицу GIF. Эта процедура выполняется посредством рассеяния имеющихся цветов.
- Remove Gradients (Удалить плавные цветовые переходы). Выполняется преобразование градиентных цветов в первый сплошной цвет градиентной заливки. Эта операция выполняется с целью уменьшения размера файла. Кроме того, градиентная заливка в

GIF-файлах может отображаться некачественно. При выборе данной опции внимательно отнеситесь к выбору первого градиентного цвета.

Работая с GIF-файлами, можно задать параметры прозрачности.

- Из раскрывающегося меню Transparent (Прозрачность) выберите пункт Opaque (Непрозрачный), если хотите, чтобы фон фильма был отображен в виде сплошного цвета.
- Чтобы фон был прозрачным, выберите опцию Transparent.
- Чтобы задать частичную прозрачность, выберите пункт Alpha (Коэффициент прозрачности) и в соседнем поле введите числовое значение от 0 до 255, определяющее степень прозрачности. Чем меньше значение, тем прозрачнее изображение. Значение 128 соответствует 50% прозрачности.

Элементы раскрывающегося меню Dither (Растривание) определяют, каким образом представлены цвета, не входящие в цветовую таблицу.

- При выборе опции None (Нет) предотвращается растривание, а для цветов, не входящих в цветовую палитру, отображаются ближайшие подходящие цвета. При этом цвета могут существенно искажаться, но размер файла при этом уменьшается.
- При выборе опции Ordered (Упорядоченное) выполняется определенное растривание, но оно ограничивается с целью сохранения **небольшого** размера файла.
- При выборе опции Diffusion (Диффузия) достигаются наилучшие результаты за счет того, что выполняется свободное растривание цветов для наилучшего воспроизведения пропущенных оттенков. Функция диффузии приводит к увеличению размера файла и времени обработки. Она работает только с Web-палитрой, состоящей из **216** цветов.

Элементы раскрывающегося меню Palette Type (Тип палитры) определяют цветовую палитру экспортируемого GIF-файла.

- При выборе опции Web 216 для создания GIF-файла используется палитра безопасных Web-цветов и выполняется самая быстрая обработка изображения со стороны сервера.
- При выборе опции Adaptive (Адаптивная) выполняется анализ цветов изображения и создается пользовательская палитра. При выборе данной опции создаются самые точные цвета, но при этом размер файла будет самым большим.
- Опция Web Snap Adaptive (Адаптивная к диапазону Web) аналогична предыдущей, за исключением того, что в данном случае к любым близким к данному диапазону цветам применяется палитра безопасных Web-цветов.
- Опция Custom (Пользовательская) позволяет создать собственную палитру. В связи с тем, что цвета данной палитры являются предварительно заданными, то созданные с ее помощью GIF-изображения загружаются так же быстро, как и изображения, созданные с помощью палитры Web **216**. Однако для достижения максимального эффекта от использования данной опции необходимо обладать определенным опытом и навыками создания цветовой палитры.

И наконец, при выборе опции Adaptive или Web Snap Adaptive можно указать число используемых цветов, введя значение в поле Max Colors (Максимальное число цветов). Указание числа цветов может способствовать уменьшению размера файла, но если число будет слишком малым, то цвета изображения могут оказаться неточными.

ФОРМАТ JPEG

Flash-фильмы можно также сохранять в формате JPEG, в виде 24-битовых растровых изображений. Данный формат лучше всего подходит для изображений с плавно переходящими тонами, таких как фотографии или градиенты. Опции вкладки JPEG показаны на рис. 30.12.

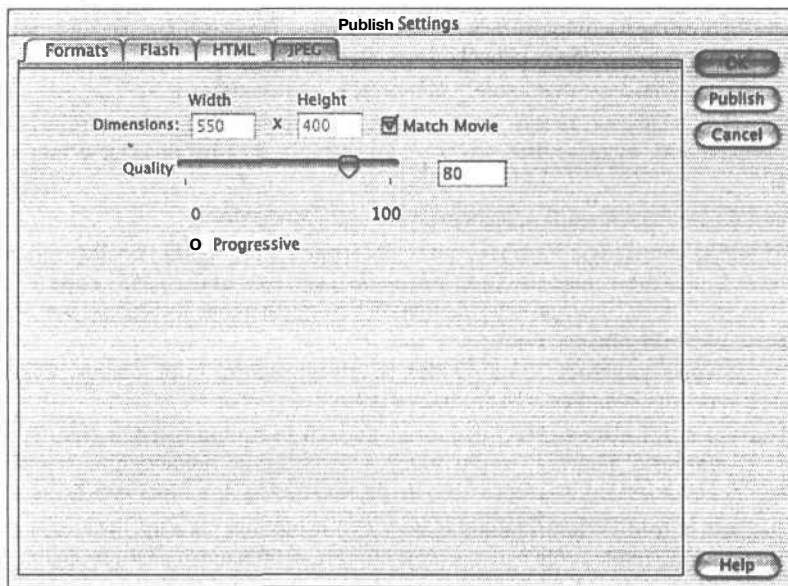


Рис. 30.12. Flash-фильм можно сохранить в виде статического JPEG-файла

Как и в случае с форматом GIF, в виде файла JPEG экспортируется первый кадр фильма, за исключением случаев, когда какому-то другому кадру назначена метка **#Static**. Размеры изображения можно задать совпадающими с размерами фильма либо модифицировать их, введя новые значения в поля **Width** и **Height**.

Качество JPEG-файла задается в диапазоне от 0 до 100, при этом по умолчанию выбрано значение 80. Для задания качества перетащите ползунок **Quality** либо введите значение в находящееся рядом поле. Качество отдельных растровых изображений можно задать в диалоговом окне **Bitmap Properties** (Свойства растра). Глобальная настройка, выполненная в диалоговом окне **Publish Settings**, применяется только в том случае, если свойства отдельных растровых изображений заданы так, чтобы использовать сжатие, заданное по умолчанию.

Если в поле опции **Progressive** (Последовательно) установить флажок, JPEG-изображения будут отображаться пошагово, по мере их загрузки.

ФОРМАТ PNG

Формат PNG, или **Portable Network Graphic** (Портативная сетевая графика), является "родным" для файлов, созданных в приложении **Macromedia Fireworks**, и единственным форматом растровых изображений, поддерживающим прозрачность. Опции вкладки **PNG**, аналогичные GIF, показаны на рис. 30.13.

Чтобы указать размеры PNG-файла, введите их в поле **Height** или **Width** либо установите флажок в поле опции **Match Movie**, в результате чего размеры PNG-файла будут совпадать с размерами фильма.

Сжатие PNG-изображений производится путем ограничения данных о цветах, отображаемых в изображениях. Для указания количества данных о цвете, приходящегося на каждый пиксель, выберите нужный элемент из раскрывающегося меню **Bit Depth** (Битовая глубина):

- 8-bit (8-битовая) — для отображения 256 цветов;
- 24-bit (24-битовая) — для отображения тысяч цветов;
- 24-bit with Alpha (24-битовая с коэффициентом прозрачности) — для отображения тысяч цветов и степени их прозрачности.

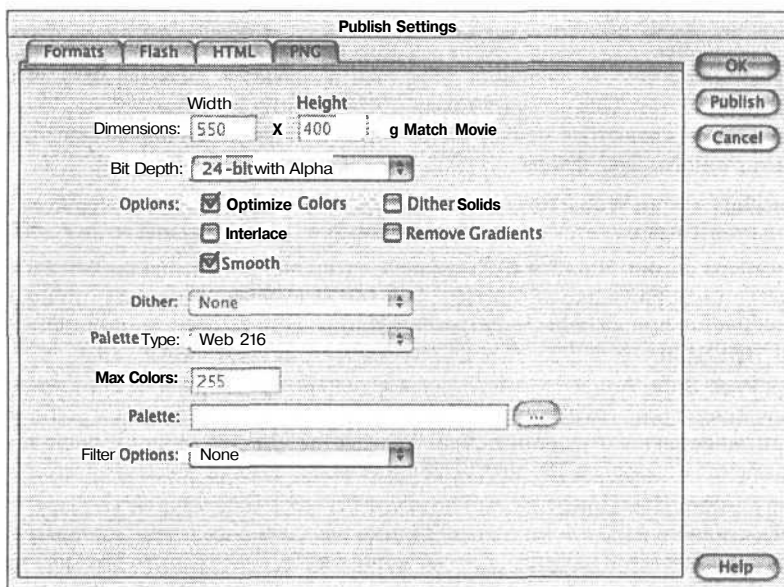


Рис. 30.13. Flash-фильмы можно сохранять в формате PNG

Чем больше включено данных о цвете, тем больше будет размер файла. Существует несколько опций отображения PNG-файлов.

- При выборе опции **Optimize Colors** (Оптимизировать цвета) из таблицы цветов PNG-файла удаляются любые неиспользуемые цвета. Эта функция не действует при выборе адаптивной палитры.
- При выборе опции **Interlace** (Чередование) PNG-файл отображается в браузере пошагово, по мере загрузки.
- При выборе опции **Smooth** (Сглаживание) к изображению применяется функция устранения контурных неровностей. Как и в случае с GIF-файлами, при помещении изображения на цветной фон вокруг него может образоваться серый ореол. Кроме того, процедура сглаживания приводит к увеличению размера файла.
- Опция **Dither Solids** (Растривать сплошные цвета) моделирует цвета, не входящие в цветовую таблицу PNG. Эта процедура выполняется посредством рассеивания имеющихся цветов.
- При выборе опции **Remove Gradients** (Удалить плавные цветовые переходы) выполняется преобразование градиентных цветов в первый сплошной цвет градиентной заливки. Эта операция выполняется с целью уменьшения размера файла. Кроме того, градиентная заливка в PNG-файлах может отображаться некачественно. При выборе данной опции внимательно отнеситесь к выбору первого градиентного цвета.

При выборе 8-битовой глубины активизируется раскрывающееся меню **Dither** (Растривание), определяющее, каким образом представляются цвета, не входящие в цветовую таблицу.

- Опция **None** (Нет) предотвращает растривание, а для цветов, не входящих в цветовую палитру, отображает ближайшие подходящие цвета. При этом цвета могут существенно искажаться, но размер файла при этом уменьшается.
- При выборе опции **Ordered** (Упорядоченное) выполняется определенное растривание, но рассеивание цветов ограничивается с целью сохранения небольшого размера файла.

- При выборе опции Diffusion (Диффузия) наилучшие результаты достигаются за счет того, что выполняется свободное растривание цветов для наилучшего воспроизведения пропущенных оттенков. Функция диффузии приводит к увеличению размера файла и времени обработки. Она работает только с Web-палитрой, состоящей из 216 цветов.

Элементы раскрывающегося меню Palette Type (Тип палитры) определяют цветовую палитру экспортируемого PNG-файла.

- При выборе опции Web216 для создания PNG-файла используется палитра безопасных Web-цветов и выполняется самая быстрая обработка изображения со стороны сервера.
- При выборе опции Adaptive (Адаптивная) выполняется анализ цветов изображения и создается пользовательская палитра. Эта самая лучшая опция для отображения миллионов цветов при выборе 24-битовой глубины. При выборе данной опции создаются самые точные цвета, но при этом размер файла будет самым большим.
- Опция Web Snap Adaptive (Адаптивная к диапазону Web) аналогична предыдущей, за исключением того, что в данном случае к любым цветам, близким к данному диапазону, применяется палитра безопасных Web-цветов. Данная опция хорошо подходит для отображения оттенков в 256-цветовой системе при выборе 8-битовой глубины.
- Опция Custom (Пользовательская) позволяет создать собственную палитру. Однако, для ее эффективного использования необходимо обладать определенным опытом и навыками создания цветовой палитры. Для указания файла, содержащего пользовательскую палитру, щелкните на кнопке, расположенной справа от поля Palette (Палитра).

При выборе опции Adaptive или Web Snap Adaptive можно указать число используемых цветов, введя соответствующее значение в поле Max Colors (Максимальное число цветов). Указание числа цветов может способствовать уменьшению размера файла, но если число будет слишком малым, то цвета изображения могут оказаться неточными.

Элементы раскрывающегося меню Filter Options (Параметры фильтрации) позволяют указать метод фильтрации, применяющийся для лучшего сжатия экспортируемого PNG-файла. При фильтрации выполняется построчный анализ данных о цветах.

- При выборе опции None (Нет) функция фильтрации отключена.
- При выборе опции Sub (Под) передается разница между байтами плюс значение соответствующего байта предыдущего пикселя.
- При выборе опции Up (Над) передается разница между байтами плюс значение соответствующего байта для пикселя, находящегося непосредственно над данным.
- При выборе опции Average (Средняя), для того, чтобы предсказать значение данного пикселя, сравниваются соседние пиксели.
- При выборе опции Path (Путь), для того, чтобы предсказать значение данного пикселя, выполняется расчет функции соседних пикселей.
- При выборе опции Adaptive (Адаптивная) выполняется анализ цветов изображения и создается пользовательская палитра. Данная опция лучше всего подходит для 24-битовой глубины, использующейся для отображения тысяч цветов.

Поэкспериментируйте с различными параметрами фильтрации и найдите наиболее эффективное сжатие, подходящее для того или иного изображения.

ФОРМАТ QuickTime

Flash-фильмы можно сохранять в формате QuickTime 4. При этом сохраняется интерактивность фильма, а в модуле QuickTime Flash-фильмы воспроизводятся точно так же, как и во Flash-плеере. Flash-фильмы копируются на отдельную дорожку QuickTime и могут быть разнесены в слои вместе с другим содержимым QuickTime. Опции вкладки QuickTime представлены на рис. 30.14.

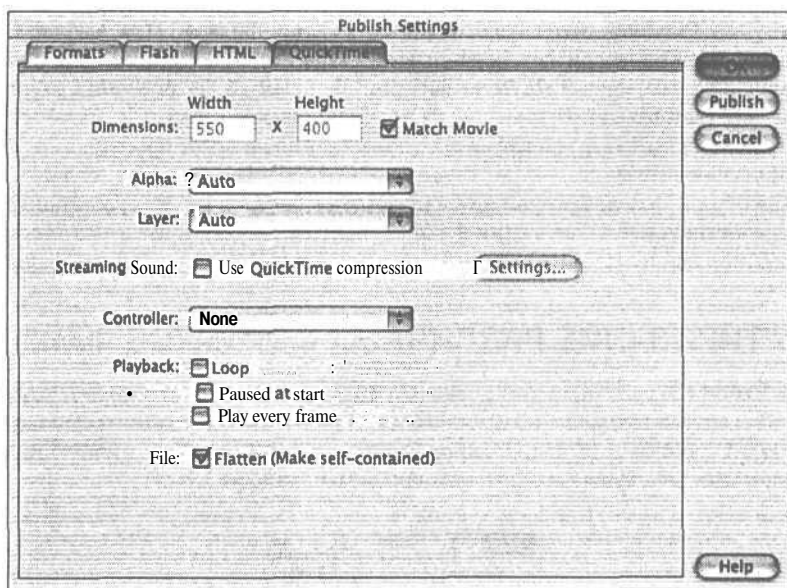


Рис. 30.14. Flash-фильмы можно сохранять в формате QuickTime 4

Как и при работе с форматами GIF и JPEG, необходимо указать размеры файла. Установите флажок в поле опции Match Movie или введите нужные значения в поля Width и Height.

Элементы раскрывающегося меню Alpha регулируют прозрачность Flash-дорожки в пределах QuickTime и не влияют на степень прозрачности Flash-фильма.

- При выборе опции **Alpha-transparent** (Степень прозрачности) Flash-дорожка становится прозрачной, открывая содержимое других дорожек, расположенных позади нее.
- Параметр **Copy** (Копировать) делает Flash-дорожку непрозрачной, закрывая содержимое других дорожек, расположенных позади нее.
- При выборе параметра **Auto** (Авто) прозрачность Flash-дорожки устанавливается в зависимости от ее места в порядке наложения. Если она находится поверх других дорожек, то она будет прозрачной. Если Flash-дорожка находится внизу стека или является единственной дорожкой, то она будет непрозрачной.

Элементы раскрывающегося меню **Layer** (Слой) определяют размещения Flash-дорожки в порядке наложения. При выборе опции **Top** (Вверху) Flash-дорожка размещается поверх остальных дорожек. При выборе опции **Bottom** (Внизу) Flash-дорожка будет располагаться под остальными дорожками. При выборе опции **Auto** Flash-дорожка будет располагаться в зависимости от размещения Flash-объектов относительно видеоэлементов в пределах Flash-фильма. Если Flash-объекты располагаются поверх видеоэлементов, то и Flash-дорожка будет располагаться вверху. Если Flash-объекты располагаются позади видеоэлементов, то Flash-дорожка также будет располагаться внизу.

При установлении флажка в поле опции **Streaming Sound** (Потоковый звук) все потоковые звуки экспортируются из Flash-фильма в звуковую дорожку QuickTime с одновременным сжатием в соответствии с параметрами QuickTime. Чтобы отрегулировать заданные по умолчанию параметры аудиосжатия QuickTime, щелкните на кнопке **Settings** (Настройки).

Элементы раскрывающегося меню **Controller** (Контроллер) определяют, каким образом пользователь сможет управлять фильмом QuickTime. Из данного меню можно выбрать элементы **None** (Нет), **Standard** (Стандартный) и **QuickTime VR** (Видеомагнитофон QuickTime).

Опции группы Playback определяют воспроизведение фильма. При выборе опции Loop (Цикл) фильм воспроизводится непрерывно. При выборе опции Paused at Start (Приостановка в начале) фильм не будет воспроизводиться до тех пор, пока пользователь не щелкнет на соответствующей кнопке. При установлении флажка Play Every Frame (Воспроизводить каждый кадр) во время воспроизведения не будет пропущен ни один кадр, а звук при этом отключается.

Опция File (Файл), в поле которой по умолчанию установлен флажок, объединяет содержимое Flash и импортируемое видео в один фильм QuickTime. В результате этого экспортируемый фильм QuickTime является самодостаточным и не зависит от связи с внешними файлами.

ЭКСПОРТИРОВАНИЕ РАЗЛИЧНЫХ ФОРМАТОВ ФАЙЛОВ

Команда Export (Экспортировать) позволяет создать во Flash содержимое, которое можно редактировать в других приложениях. Для экспортирования содержимого выполните команду File⇒Export Movie (Файл⇒Экспортировать фильм) или File⇒Export Image (Файл⇒Экспортировать изображение), как показано на рис. 30.15.

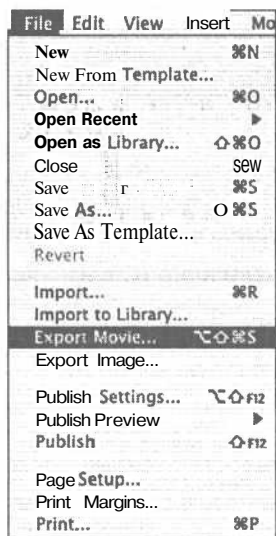


Рис. 30.15. Содержимое Flash можно экспортировать в виде фильмов или изображений

В диалоговом окне Export (рис. 30.16) укажите имя файла, а затем выберите формат файла и место, в котором его следует сохранить.

Фильмы экспортируются в виде последовательностей, а изображения — в виде отдельных файлов. Flash-файлы можно экспортировать в следующих форматах.

- *Adobe Illustrator* (.ai) идеально подходит для перемещения доступной для редактирования векторной графики между приложениями для рисования. Формат *Illustrator* сохраняет точные кривые Безье, линии и заливку. Поддерживаются функции импортирования и экспортирования форматов *Illustrator 3.0, 5.0, 6.0 и 8.0*.
- Формат *GIF* (.gif) включает в себя анимированные GIF-файлы и последовательности GIF. Форматы GIF поддерживают изображения и последовательности и лучше всего подходят для векторной графики и непродолжительной анимации.

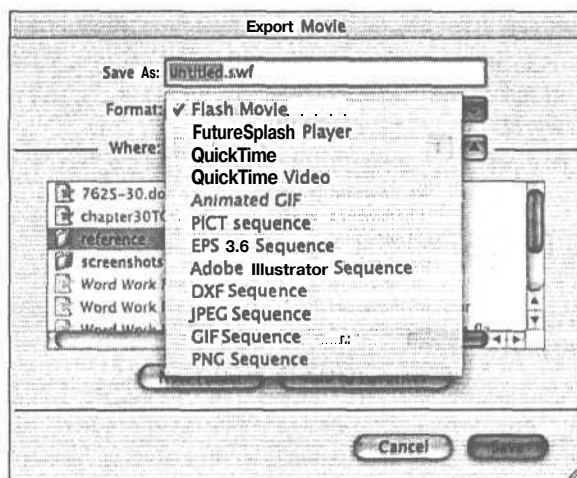


Рис. 30.16. Flash можно экспортировать во многих форматах

- Bitmap (.bmp) используется только в Windows для экспортирования растровых изображений, которые затем можно использовать в других приложениях.
- DXF Sequence и AutoCAD DXF Image (.dxf) экспортируют векторные элементы, которые можно использовать в DXF-приложениях.
- Enhanced MetaFile (.emf) — новый графический формат, поддерживающий векторные и растровые данные в Windows 95 и Windows NT.
- EPS (.eps) позволяет экспортировать текущий кадр с целью его размещения в приложениях макетирования, таких как Quark Express, Adobe PageMaker и Microsoft Word.
- Flash Movie (.swf) позволяет экспортировать весь фильм, чтобы разместить его в других приложениях, таких как Dreamweaver.
- FutureSplash Player (.spl) — оригинальный формат и название Flash до того, как эта технология стала собственностью Macromedia. Параметры экспортирования соответствуют параметрам публикации во Flash.
- JPEG Sequence и JPEG Image (.jpg) экспортируют растровые изображения для использования в других программах. Соответствующие параметры были описаны выше, в подразделе "Формат JPEG".
- PICT (.pct) — стандартный графический формат Macintosh, поддерживающий векторные и растровые данные.
- PNG Sequence и PNG Image (.png) — единственный межплатформенный растровый формат, поддерживающий прозрачность. Он является "родным" форматом приложения Fireworks.
- QuickTime (.mov) может объединять интерактивное Flash-содержимое и мультимедийное содержимое QuickTime в единый фильм QuickTime, который можно просмотреть посредством модуля QuickTime 4.
- QuickTime Video (.mov) для Macintosh преобразует Flash-фильм в ряд растровых изображений, внедренных в дорожку QuickTime. Интерактивностью приходится жертвовать. Данная опция позволяет редактировать Flash-фильмы в приложениях для редактирования видео.

- WAVAudio(, wav) для Windows экспортирует звуковые файлы в единый аудиофайл WAV.
- Windows AVI (, avi) экспортирует фильм в виде видеофайла для Windows. При выборе этого формата интерактивность не учитывается. Данный формат эквивалентен формату QuickTime Video для Macintosh, но для Windows.
- Windows MetaFile(, .wmf) — стандартный графический формат Windows.

ВОЗМОЖНЫЕ ПРОБЛЕМЫ

В чем отличие экспортирования фильма посредством команды Export от его экспортирования при тестировании?

Всякий раз при тестировании фильма (посредством команды **Control⇨Test Movie**) экспортируется **SWF-файл**. Экспортирование файла происходит и при обращении к команде Export. Однако, как вы только что узнали, команда Export является гораздо более мощным инструментом, так как помимо SWF она поддерживает много других форматов. Во время работы над фильмом вам придется постоянно его тестировать, а экспортировать фильм в другие форматы необходимо только один раз — по завершении работы над фильмом.

Почему при просмотре SWF-файла в браузере растровые изображения и шрифты выглядят размытыми?

Проверьте, какой элемент выбран из меню **Scale** (Масштабирование) во вкладке HTML диалогового окна Publish Settings. Обязательно должен быть выбран элемент No Scale (Без масштабирования). Если разрешить масштабирование, то размеры растровых изображений, в том числе и растровых шрифтов, изменятся и это приведет к тому, что они будут выглядеть размытыми. Кроме того, во вкладке HTML из меню Quality (Качество) обязательно нужно выбрать элемент High (Высокое) или Best (Наилучшее).

FLASH ЗА РАБОТОЙ: ИСПОЛЬЗОВАНИЕ ИМЕНОВАННЫХ АНКЕРОВ

Именованные анкеры позволяют пользователям использовать кнопки Back (Назад) и Forward (Вперед) браузера для перемещения по фильму. Навигационные функции браузера выступают в роли вспомогательных средств навигационных систем во Flash-фильмах. Средства навигации, заданные в фильме, позволяют пользователям перемещаться по фильму для выполнения различных действий. Однако именованные анкеры позволяют создавать "закладки" кадров или сцен так, что кнопки браузера реализуют дополнительное управление воспроизведением фильма.

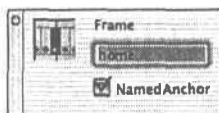
Внимание!

Именованные анкеры функционируют только в браузерах, поддерживающих команду **fscommand**, таких как Internet Explorer для Windows и Netscape 4.

Именованные анкеры назначаются в ключевых кадрах. Для того чтобы указать их, выполните следующие действия.

1. Откройте законченный фильм.
2. На временной шкале выберите ключевой кадр, который вы хотите использовать в качестве именованного анкера.

3. Если панель инспектора свойств закрыта, откройте ее, выполнив команду **Window⇒Properties (Окно⇒Свойства)**, или нажав клавиши **<Cmd+F3> (Macintosh)**, или **<Ctrl+F3> (Windows)**. В поле **Frame (Кадр)** введите имя.
4. Установите флажок в поле опции **Named Anchor (Именованный анкер)**, расположенной под полем **Frame** (рис. 30.17).
5. В ключевом кадре на временной шкале появится пиктограмма именованного анкера (рис. 30.18). Чтобы именованные анкеры могли функционировать, в диалоговом окне **Publish Settings** необходимо выбрать соответствующую опцию. Для этого во вкладке **HTML** из раскрывающегося меню **Template (Шаблон)** выберите пункт **Flash with Named Anchors (Flash с именованными анкерами)**.



*Рис. 30.17. Чтобы сделать кадр именованным анкером, на панели инспектора свойств установите флажок в поле опции **Named Anchor***



Рис. 30.18. Ключевые кадры, являющиеся именованными анкерами, помечаются соответствующей пиктограммой

УЛУЧШЕНИЕ ДОСТУПНОСТИ FLASH

В ЭТОМ ПРИЛОЖЕНИИ...

Правила доступности

688

ПРАВИЛА ДОСТУПНОСТИ

До выхода версии Flash MX содержимое Flash было практически недоступно для пользователей, пользующихся программами считывания данных с экрана, и для пользователей с ослабленным слухом. В настоящее время Flash Player 6 может взаимодействовать с программами считывания данных и другими вспомогательными устройствами работы с визуальным содержанием.

Программы считывания данных с экрана позволяют работать с Internet пользователям с ослабленным слухом. Такие программы могут считывать текст, но они не способны воспроизводить изображения или анимацию. В связи с тем, что содержимое Flash является недоступным для программ считывания, очень важно задать параметры доступности для как можно большего числа элементов Flash MX.

Ниже перечислены пять объектов, которые доступны во Flash и передаются из Flash-плеера в программы считывания данных с экрана:

- текст;
- текстовые поля ввода данных;
- кнопки (в том числе и видеоклипы кнопок);
- видеоклипы;
- фильмы целиком.

Однако отдельные графические изображения передать в программы считывания невозможно. Это особенно важно учесть при работе с текстом. Любой графический текст, т.е. тот, который разбит на отдельные контуры, является недоступным. Это означает, что недоступными могут оказаться целые навигационные системы. Имейте это в виду при разработке фильмов и убедитесь в том, что важные меню созданы в виде текста, а не в виде контуров.

К каждому из пяти доступных объектов Flash позволяет присоединять дополнительные текстовые описания (подобно дескрипторам ALT в HTML), которые сделают содержимое более полезным и доступным для программ считывания. Наиболее важным является свойство Name (Имя), которое можно назначить для объектов. Это имя будет громко зачитываться программами считывания данных. При работе над фильмом можно указать, какие Flash-объекты будут доступными для программ считывания.

Чтобы указать параметры доступности Flash-объектов, выполните следующие действия.

1. Выделите Flash-объект на рабочем поле.
2. На панели инспектора свойств щелкните на кнопке Accessibility (Доступность), как показано на рис. А.1. В результате откроется панель **Accessibility** (рис. А.2).

В качестве альтернативного способа открытия панели Accessibility можно воспользоваться комбинацией клавиш <Opt+F2> (Macintosh) или <Alt+F2> (Windows).



Рис. А.1. После щелчка на кнопке Accessibility панели инспектора свойств открывается панель Accessibility

3. Проверьте, чтобы в поле опции Make Object Accessible (Сделать объект доступным) был установлен флажок, а затем на панели Accessibility введите имя и описание объекта (см. рис. А.2).

На панели Accessibility можно даже назначить комбинации клавиш, инициирующие щелчок на кнопке или ввод данных в текстовые поля. Введите название клавиши, такой как <Alt> или <Ctrl>, и знак "плюс" без пробелов, чтобы добавить названия клавиш.

На заметку

В настоящее время программное обеспечение для считывания информации с экрана доступно только для платформы Windows. Имейте это в виду при создании клавиш быстрого доступа и не задействуйте клавиши, имеющиеся только на компьютерах Macintosh, например <Cmd>.

Обязательно указывайте имена любых доступных объектов. Даже если объект содержит только визуальное содержимое (например, специальный анимационный объект), очень важно указать это содержимое программам считывания данных. Если этого не сделать, то пользователи, работающие со вспомогательными устройствами, могут вообще не узнать о важном содержимом или подумать, что они пропустили что-то важное.

По завершении работы над Flash-фильмом необходимо задать доступность всего фильма для объектов, которые были сделаны доступными для обработки программами считывания данных. Чтобы задать доступность всего фильма, выполните следующие действия.

1. Когда фильм завершен и готов к публикации, снимите выделение всех элементов, находящихся на рабочем поле, а затем откройте панель Accessibility, щелкнув на одноименной кнопке панели инспектора свойств или выполнив команду Window⇒Accessibility.
2. Установите флажок в поле опции Make Movie Accessible (Сделать фильм доступным), как показано на рис. А.3. Если вы хотите скрыть фильм от программ считывания, снимите данный флажок.
3. Чтобы назначенные доступными объекты **могли быть** обработаны программами считывания данных с экрана, установите флажок в поле опции Make Child Objects Accessible (Сделать дочерние объекты доступными).

4. Чтобы использовать текстовые объекты в качестве автоматических меток для доступных кнопок и текстовых полей ввода данных, установите флажок в поле **опции** Auto Label (Автометка). Флажок в поле данной опции установлен по умолчанию.
5. В поле **Name** (Имя) введите краткий описательный заголовок.
6. В поле Description (Описание) введите более длинное описание. Представьте себе, что вы не можете посмотреть фильм, и введите такое описание, которое могло бы представить его, не опираясь на визуальные подсказки.

Несмотря на то что во Flash MX сделаны попытки улучшить доступность содержимого, дизайнеры и разработчики должны убедиться в том, что важное содержимое будет доступным для всех категорий пользователей. Предпримите для этого все возможные усилия.

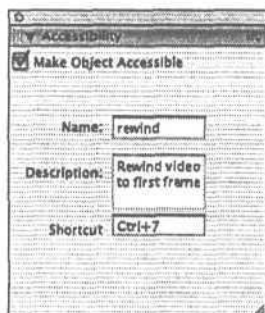


Рис. А.2. На панели Accessibility для Flash-объектов можно назначить имена, описания, а также указать комбинации клавиш быстрого доступа

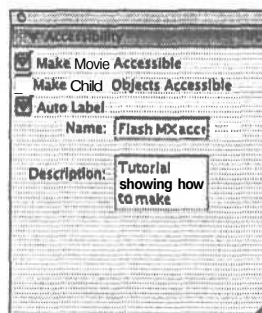


Рис. А.3. Flash-фильм необходимо сделать доступным так, чтобы соответствующие объекты могли быть обработаны программами считывания данных с экрана

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

A

ActionScript, 199

- автоматическое преобразование типов данных, 263
- будущие зарезервированные слова, 224
- выражения, 248
- двоичные числа, 254
- зарезервированные слова, 223
- ключевые слова предложений, 282
- настройка параметров редактора, 58
- нестрогие типы данных, 234
- объявление переменных, 218
- операторы, 245; 246; 247; 248
- панель Actions, 200
- панель ActionScript Reference, 36
- панель Code Hints, 36
- предложения, 248
- работа с цветом, 470
- редактор, 36
- типы данных, 225; 226
- управление ходом выполнения программы, 283

ADPCM, 168

AIFF, 160

Align, 77

Arrow, 46; 67; 68

- модификаторы, 46

Audio Video Interleaved, 188

B

Break Apart, 30

Break Apart, команда, 92

Brush, 47; 67

C

CGI-программы, 630

Code Hints, 36

Color Mixer, 74

D

Distribute to Layers, 31

E

Eraser, 49; 75

Eyedropper, 49

F

Fill Transform, 48

Flash

- Java-классы, 581

- классы ActionScript, 582

- отправление электронной почты, 634

- удаленный доступ, 580

Flash Communication Server, 479; 496

Flash MX

- группы панелей, 29

- диалоговое окно Preferences, 53

- доступ к содержимому, 36

- доступ к наборам панелей, 41

- загрузка внешних изображений и звука, 35

- задание параметров, 53

- импортирование растровых изображений, 81

- интерактивность, 173

- интерфейс, 39

- клавиши быстрого доступа, 50

- методики рисования, 395

- модель событий, 34

- наборы панелей, 39

- настройка предупреждающих сообщений, 57

- новые инструментальные средства, 30

- новые свойства видеоклипов, 363

- общие настройки, 54

- объектная модель, 34

- параметры буфера обмена, 57

- параметры редактирования, 55

- параметры редактора ActionScript, 58

прикладной программный интерфейс
рисования, 35
расширение обработки событий, 333
события, 335
совместимость с Rash 5, 36
создание оптимального интерфейса, 59
создание пользовательского интерфейса,
553
усовершенствование сценариев, 34
шаблоны, 32
Free Transform, 30; 48

Н

Hand, 49
HSB, модель цветопередачи, 72

И

Import, диалоговое окно, 82
Ink Bottle, 49

Л

Lasso, 46; 92
модификаторы, 47
Library, 52
Line, 46; 63

М

Microsoft Active Accessibility, 469
Motion Picture Experts Group, 188
Movie Explorer, 53
MP3, 160; 169

Н

NaN, 250

О

Oval, 47; 63

Р

Paint Bucket, 49
Pen, 47; 63
Pencil, 47; 63
PNG-изображения, 680

Q

QuickTime, 682
QuickTime Movie, 188

R

Rectangle, 47; 63
RGB, 472
RGB, модель цветопередачи, 72
Round Rectangle **Radius**, 40

S

Smart Clips, 537
Snap to Pixels, 31
Sorenson Spark, 32; 188
параметры сжатия, 189
Sound Designer II, 160
Subselection, 46; 65
Sun AU, 160
System 7 Sounds, 160

T

Text, 47; 99

W

WAV, 160
Windows Media **File**, 188

X

XML-данные, 637
синтаксический анализ, 640
XML-документ
дескрипторы, 639
создание, 639
XML-свойства, 640
XML-сокеты, 646
XML-узлы, 638

Z

Zoom, 49

A

Анимация, 137

- вложенные символы, 153
- Анимация**
 - вторичное действие, 154
 - деформация при движении, 154
 - добавление к кнопке, 179
 - добавление кадров, 139
 - добавление ключевых кадров, 139
 - завершение движения, 153
 - кадры, 138
 - калькирование, 148
 - ключевые кадры, 138
 - мультипликация, 152
 - накладывающиеся действия, 154
 - направляющие движения, 143
 - ожидание, 153
 - панель Timeline, 138
 - покадровая, 141
 - правила создания, 153
 - пример создания, 141
 - промежуточное отображение движения, 142
 - распределение во времени и движение, 153
 - растровых изображений, 95
 - регулировка движения, 147
 - с созданием промежуточных отображений, 141
 - создание промежуточных отображений с изменением формы, 149
 - создание эффекта панорамы, 152
 - специальные объекты, 467
 - удаление кадров, 139
 - указатели формы, 151
 - упрощение, 155
 - частота смены кадров, 140
 - штрихи, 141
- Анкерные точки**, 63
 - корректировка, 65
- Аргументы**
 - передача данных в функции, 300
- Атрибуты**, 41
 - диалоговое окно Document Attributes, 41
- Аудиособытие**, 675

Б

- Безье**, кривые, 64
- Библиотека**, 52
 - добавление компонентов, 184
 - импортирование видео, 191
 - импортирование растровых изображений, 82
 - использование с компонентами, 544
 - конфликт библиотеки, 135

- Библиотека**
 - панель Library, 52
 - присоединение загруженных фильмов, 370
 - размещение звуковых файлов, 160
 - размещение символов, 132
 - символы, 117
 - создание общих библиотек, 52
- Библиотеки коллективного использования**, 577
- Битовая глубина звука**, 159
- Булевы данные**, 229

В

- Взаимодействия**, 563
- Взаимосвязь Director—Flash**, 617
- Взаимосвязь Flash—Director**, 616
- Взаимосвязь Flash-фильмов**, 622
- Взаимосвязь с сервером**
 - внедрение данных на Web-страницу, 628
 - метод LoadVariables, 627
 - объект Loadvars, 625
 - передача данных на сервер, 631
- Видео**
 - Audio Video Interleaved, 188
 - Motion Picture Experts Group, 188
 - QuickTime Movie, 188
 - Sorenson Spark, 188
 - Windows Media File, 188
 - внедрение, 189
 - замена, 192
 - импортирование, 188
 - импортирование в библиотеку, 191
 - импортируемые форматы, 188
 - интерактивность, 195
 - маски, 191
 - поддержка во Flash MX, 32
 - преобразование экземпляров, 191
 - сжатие, 188
 - создание кнопок для управления, 192
 - управление воспроизведением, 192
- Видеоклип кнопки**, 334
- Видеоклипы**, 118; 357
 - визуальный порядок наложения, 367; 372
 - временная шкала**, 358
 - выгрузка внешних файлов, 371
 - выделение в подкласс, 378
 - загрузка внешних файлов, 369
 - задание идентификатора связи, 550
 - иерархическая структура, 372
 - иерархия, 372

Видеоклипы

- изменение размеров с помощью масштабирования, 457
- использование для систематизации переменных, 220
- ключевые кадры, 358
- методы, 383
- методы класса **MovieClip**, 358; 359; 360; 361
- новые свойства во Flash MX, 363
- перетаскивание, 387
- показатели глубины, 367, 373;
- проверка перекрытия, 382
- родительско-дочерние связи, 367
- свойства класса **MovieClip**, 361; 362
- свойство `_droptarget`, 389
- свойство `_proto_`, 380
- свойство `enabled`, 364
- свойство `focusEnabled`, 364
- свойство `hitArea`, 365
- свойство `tabChildren`, 365
- свойство `tabEnabled`, 366
- свойство `tabIndex`, 366
- свойство `trackAsMenu`, 366
- свойство `useHandCursor`, 366
- систематизация данных и функций, 211
- создание, 367
- способы создания, 368
- структурирование фильма, 180
- точка регистрации, 367
- удаление, 369
- уровни визуального порядка наложения, 373

Вложенные видеоклипы, 656

Временная шкала, 42

- видеоклипов, 358
- выделение нескольких кадров, 139
- доступ к переменным внутри функции, 299
- использование для систематизации кода, 212
- панель **Timeline**, 42
- перемещение по ней, 151
- систематизация переменных, 220
- создание сценариев, 330
- структурирование документов, 180

Выравнивание

- абзацев, 104
- объектов, 77

Выражения, 248

правила обзора, 211

Динамические данные, 630

3

- Заливка, 72; 406
- градиентная, 73
- инструмент **Fill Transform**, 48
- растровая, 74
- редактирование, 72
- текста, 102
- чистым цветом, 72
- Звук, 157
- ADPCM**, 168
- AIFF**, 160
- MP3**, 160
- MP3-сжатие**, 169
- Sound Designer II**, 160
- SunAU**, 160
- System 7 Sounds**, 160
- WAV**, 160
- битовая глубина, 159
- добавление к кнопке, 178
- загрузка и присоединение, 477
- звуковые форматы, 160
- импортирование, 160
- ключевые кадры, 165
- маркеры огибающей, 164
- моно, 160
- настройка **QuickTime 4**, 160
- объект **Sound**, 166
- оптимизация, 167
- отключение, 170
- параметры синхронизации, 162
- подготовка для **Flash**, 158
- получение данных о нем, 481
- поточковый, 162; 478
- против анимации, 162
- редактирование во **Flash**, 163
- свойства, 161
- сжатие, 167
- синхронизация, 161
- событийный, 162; 478
- стерео, 160
- управление, 165; 479
- фильмов **QuickTime**, 160
- циклическое воспроизведение, 164
- частота дискретизации, 159

И

- Идентификаторы, 223; 296

Д

Данные

Изображения

векторные, 62
вспомогательные элементы компоновки, 76
градиентная заливка, 73
заливка, 72
заливка чистым цветом, 72
использование инструментов рисования, 62
направляющие, 76
направляющие слои, 77
пиксели, 61
растровые, 61; 74; 81
растровые против векторных, 88
растровые, анимация, 95
растровые, оптимизация, 90
растровые, подготовка к импортированию, 83
растровые, разбивка, 92
растровые, свойства, 93
растровые, сжатие, 93
растровые, трассировка, 87
сетка, 76
создание криволинейных сегментов, 64
создание прямых отрезков, 64
Инспектор свойств, 30; 40
замена старых панелей, 30; 40
параметры текста, 100
Инструменты
Arrow, 46; 67; 68
Brush, 47; 67
Eraser, 49; 75
Eyedropper, 49
Fill Transform, 48
Free Transform, 30; 48
Hand, 49
Ink Bottle, 49
Lasso, 46; 92
Line, 46; 63
Oval, 47; 63
Paint Bucket, 49
Pen, 47; 63
Pencil, 47; 63
Rectangle, 47; 63
Round Rectangle Radius, 40
Subselection, 46; 65
Text, 47; 99
Zoom, 49
панель инструментов, 45
Интерполяция изображений, 151
Интерфейсные классы
методы и свойства, 431
Использование XML, 639

К

Кадры, 42; 138
добавление, 139
калькирование, 45
ключевые кадры, 42
удаление, 139
частота смены, 140
Калькирование, 45; 148
маркеры калькирования, 45
Кегль, 101
Кернинг, 102
Клавиши быстрого доступа
к инструментам, 50
к панелям, 51
наборы, 51
настройка, 51
создание новых наборов, 51
Ключевое слово, 281
Ключевые кадры, 42; 138; 165; 662
видеоклипов, 358
вставка, 42
добавление, 139
преобразование в обычные, 139
пустые, 42; 140
управление, 140
Ключевые слова
this, 320; 337
var, 299
создание локальных переменных внутри функции, 298
Кнопки, 118; 174
активная область, 349
курсор в виде руки, 349
невидимые, 179
объекты, 469
присоединение анимации, 179
присоединение действия, 176
присоединение звука, 178
простые, 174
свойства, 181
сложные, 178
Кнопки
создание видеоклипа, 349
состояния, 118; 174; 349
управления видеоклипом, 192
Код
ActionScript, 200
без необходимости централизации, 219
выбор режима, 203
достижимость, 210
жесткое кодирование, 319

Код

- комментарии, 207
- обычный режим, 202; 203
- оптимизация, 208
- повторное использование, 208; 311
- поиск с помощью панели Movie Explorer, 212
- применение наследования, 311
- присоединение действий, 201
- систематизация, 210
- систематизация переменных в основной временной шкале, 220
- систематизация с помощью временной шкалы, 212
- советы по использованию кода, 202
- создание удобочитаемых имен, 208
- стандартизация, 208
- удобочитаемость, 206
- упрощение, 208
- централизация, 210
- чувствительность к регистру символов, 224
- экспертный режим, 202; 203

Кодек, 188

Команды

- Break Apart, 30
- Distribute to Layers, 31
- Snap to Pixels, 31

Команды Lingo, 619

Комбинированный управляющий элемент, 182

Компоненты, 32; 182

- CheckBox, 540
- ComboBox, 538
- ListBox, 539
- PushButton, 182
- Rectangle, свойства, 549
- встроенные, 537
- добавление, 182
- комбинированный управляющий элемент, 182
- нажимные кнопки, 542
- окно списка, 182
- определение свойств, 548
- панель Components, 182
- панель прокрутки, 182
- переключатели, 182; 541
- полоса прокрутки, 182; 495; 543
- прокручивающееся текстовое поле, 542
- редактирование, 184
- редактирование оболочек, 545
- создание новых, 545
- создание пиктограммы, 550
- способы добавления в файлы, 538
- текущий просмотр, 556
- тестирование, 550

Компоненты

- установка на панели Components, 551
- флаговое поле, 182

Кривые Безье, 64

Л

Литералы, 239

- массивов, 240
- объектов, 240
- строковые, 240
- числовые, 240

М

Маски, 35

- динамические, 389
- создание, 78

Массив, 230; 420

- ассоциативный, 425
- градиентный, 408
- добавление элементов, 424
- индексы, 230
- методы, 425
- обращение к элементам, 423
- свойства, 426
- создание, 422
- типы данных, 421
- удаление элементов, 424

Массивы, 205

Матрица преобразования, 409

- вращение объектов, 413
- краткий формат, 410
- масштабирование объектов, 412
- объединение операций, 413
- традиционная, 410
- трансляция, 413
- элементы, 411

Межзнаковый интервал, 102

Методы, 206

- attachMovie(), 379
- Function.apply(), 316
- Function.call(), 316
- registerClass(), 379
- видеоклипов, 383
- градиентной заливки, 407
- класса Camera, 497
- класса Date, 436; 437; 438; 439; 440
- класса LocalConnection, 508
- класса Microphone, 500
- класса MovieClip, 358; 359; 360; 361
- класса Object, 442

Методы

- класса **TextFormat**, 493
 - массивов, 425
 - назначение для класса, 309
 - назначение посредством объявления функции, 310
 - назначение с **помощью** функционального литерала, 311
 - объекта **Key**, 474; 475
 - объекта **Math**, 452; 453
 - объекта **NetConnection**, 502
 - объекта **NetStream**, 503
 - объекта **Video**, 497
 - определение**, 308
 - прикладного программного интерфейса рисования, 396; 397
 - рисования кривых, 402
 - рисования линий и кривых, 399
 - текстовых полей, 485; 486; 487; 488; 489; 490; 491
 - тригонометрические, 456
- Модели цветопередачи, 72
- Модификаторы
- Brash Mode**, 47
 - Distort**, 48
 - Envelope**, 48
 - Erase Fills**, 49
 - Erase Inside**, 49
 - Erase Mode**, 49
 - Erase Normal**, 49
 - Erase Selected Fills**, 49
 - Erase Strokes**, 49
 - Eraser Shape**, 49
 - Faucet**, 49
 - Magic Wand**, 47; 92
 - Magic Wand Settings**, 47
 - Paint Behind**, 47
 - Paint Fills**, 47
 - Paint Inside**, 48
 - Paint Selection**, 47
 - Pencil Mode**, 47
 - Polygon Mode**, 47
 - Pressure**, 48
 - Rotate and Skew**, 48
 - Round Rectangle Radius**, 47; 63
 - Scale**, 48
 - Smooth**, 46; 69
 - Snap to Objects**, 46
 - Straighten**, 46; 69
 - инструмента **Arrow**, 46
 - инструмента **Brush**, 47
 - инструмента **Eraser**, 49
 - инструмента **Free Transform**, 48

Модификаторы

- инструмента **Lasso**, 47
 - инструмента **Pencil**, 47
 - инструмента **Rectangle**, 47
- Модификаторы инструментов
- Distort**, 30
 - Edit Envelope**, 30
- Модуль **Bandwidth Profiler**, 669
- Мультипликация, 152

Н

Навигация

- по **фильму**, 180
- Наиболее распространенные проблемы, 589
- Направляющие, 76
- Направляющие движения, 143
- Направляющие слои, 77
- Наследование
- иерархия, 313
 - использование оператора **new** для создания цепочек, 313
 - переписывание свойств, 522
 - подмена наследуемых свойств, 520
 - применение для создания повторно используемого кода, 311
- Нераспечатываемые кадры, 657

О

Обзор данных

- автоматический, 322
 - в объекте, 321
 - обработчики событий, 337
 - явный, 320
- Область печати, 659
- Обработчики событий, 330
- класса **Camera**, 500
 - класса **LocalConnection**, 508
 - класса **Microphone**, 501
 - обзор данных, 337
 - объекта **NetConnection**, 502
 - объекта **NetStream**, 503
 - отключение и удаление, 340
 - фокусировка, 339
 - явное обращение, 338
- Обучающие взаимодействия, 560
- активная область, 570
 - активный объект, 570
 - конфигурирование, 567
 - множественный выбор, 571

- Обучающие взаимодействия
 - параметры обратной связи, 572
 - структура сценариев, 573
 - истина или ложь, 571
- Объект
 - arguments, 302
- Объекты, 205
 - Accessibility, 469
 - arguments, 301; 302; 305
 - Array, 420
 - Key, 345; 473
 - Key, методы, 474; 475
 - Math, 452
 - Math, методы и свойства, 452; 453
 - Mouse, 345; 476
 - NetConnection, 501
 - NetStream, 501
 - Object, 223
 - prototype, 312; 517
 - Selection, 350; 476
 - Sound, 477
 - Stage, 351; 481
 - System, 484
 - System.capabilities, 470
 - TextField, 352; 484
 - TextFormat, 484; 492
 - Video, 496
 - базовые, 421
 - видеоклипы, 357
 - встроенные, 419
 - глобальные, 419
 - глобальные, отсутствие подмены свойств, 522
 - детализация иерархий классов, 532
 - доступ к свойствам, 273
 - интерфейсные классы, 430
 - использование для отслеживания рисования, 398
 - использование для систематизации переменных, 221
 - класс Boolean, 434
 - класс Camera, 498
 - класс Color, 470
 - класс Date, 436
 - класс Function, 441
 - класс LocalConnection, 507
 - класс Microphone, 500
 - класс Number, 435
 - класс Object, 441
 - класс String, 435
 - кнопки, 469
 - моделирование области действия задачи, 229
 - мультимедийные, 495
 - одноэлементные множества, 420
 - переписывание наследуемых свойств, 522
 - подмена наследуемых свойств, 520
 - подмена свойств в отдельных экземплярах, 522
 - получение локальных свойств надкласса в экземплярах подкласса, 524
 - пользовательские, 515
 - регистрация в качестве приемника, 336
 - разработка надклассов и подклассов, 532
 - свойство _proto_, 518
 - свойство constructor, 524
 - связанные с фильмами, 468
 - создание иерархии классов, 528
 - создание объектов-потомков, 525
 - создание с помощью оператора new, 307
 - создание свойства _constructor_ в прототипе подкласса, 531
 - сравнение, 230; 434
 - функции-конструкторы, 419
- Одноэлементное множество, 420
- Окно Output, 596
- Окно списка, 182
- Окружения Flash-фильмов, 603
- Операнды, 225; 248
 - специальное значение NaN, 250
- Операторы, 225; 245
 - delete, 272
 - instanceof, 272
 - new, 271; 307; 313; 517
 - super, доступ к надклассу, 523
 - typeof, 271
 - void, 272
 - арифметические, 246; 250
 - ассоциативность, 249
- Операторы
 - больше, 262
 - больше или равно, 262
 - булевы, 246; 266
 - версии Flash 4, 264
 - вычитания, 251
 - группирования, 249
 - деления, 253
 - деления по модулю, 253
 - запятая, 270
 - именованные, 271
 - использование в качестве постфикса, 250
 - использование в качестве префикса, 250
 - категории, 246
 - круглые скобки/обращение к функции, 274
 - логические, 246; 266
 - массив-элемент/объект-свойство, 273
 - меньше, 262
 - меньше или равно, 262

Операторы

- неравенства, 262
- операнды, 248
- отношения, 262
- перегруженные, 252
- переназначения, 246
- побитовое **И**, 256
- побитовое **ИЛИ**, 257
- побитовое исключающее **ИЛИ**, 258
- побитовое **НЕ**, 258
- побитовые, 246; 254
- побитовые логические, 255
- полиморфные, 252
- приращения, 250
- присваивания, 246; 260
- равенства, 262
- сдвиг влево со знаком, 260
- сдвиг вправо без знака, 260
- сдвиг вправо со знаком, 259
- сложения, 251
- сложного присваивания, 261
- со сдвигом бита, 259
- сравнение в объектах и массивах, 265
- сравнения, 246; 262
- старшинство, 249
- строого неравенства, 262
- строого равенства, 262
- точка**, 273
- троичные, 269
- умножения, 253
- унарного отрицания, 252
- условные, 268
- циклические, 287

Оптимизация растровых изображений, 671

Отчет о размерах, 671

II

Панели

- Accessibility, 36; 40
- Actions, 40; 200
- ActionScript** Reference, 36
- Align, 40; 77
- Answers, 40
- Code Hints, 36
- Color Mixer, 40; 69; 72; 74
- Color **Swatches**, 40
- Component Parameters, 40
- Components, 32; 40; 182; 537; 551
- Debugger, 40
- Info, 40
- Library, 40; 52
- Library, **использование** с компонентами, 544

Панели

- Movie Explorer, 53; 212
- Output, 40
- Property Inspector, 40
- Reference, 40
- Stage, 40
- Timeline, 40; 138
- Toolbox, 40; 45
- Transform**, 40
- группы панелей, 29
- доступ, 41
- инспектор свойств, 30
- компоновка, 41
- плавающие, 40
- сохранение пользовательских наборов панелей, 41

Панель инструментов

- подраздел Colors, 50
- подраздел Tools, 45
- подраздел View, 49

Панель прокрутки, 182

Панорама, 152

Переключатель, 182

Переменные, 206; 217

- идентификаторы, 223
- использование видеоклипов для систематизации, 220
- использование объектов для систематизации, 221
- назначение значений, 218
- область действия, 219
- объявление, 217; 219
- объявление явным образом, 218
- правила присвоения имен, 223
- создание внутри функций, 298

Печать Web-содержимого, 655

Печать выбранных кадров, 656

Пиксели, 61

Планирование Flash-проектов, 590

Полоса прокрутки, 182

Правила обзора данных, 211

Предложения, 248; 279

- do-while, 288
- for, 288
- for-in, 289
- if, 284
- if-else, 284
- ifFrameLoaded, 286
- startDrag(), 387
- stopDrag(), 387
- switch/case, 285
- while, 287

- Предложения
 - автономные, 279
 - автономные, образование, 280
 - ключевые слова, 281
 - объединение в управляющие блоки, 281
 - управляющие блоки, 279
 - условные, вложение, 285
- Привязка объектов, 78
- Прикладной программный интерфейс
 - рисования, 396
 - методы, 396; 397
- Программа Director, 615
- Профайлер полосы пропускания, 669
- Процедура печати, 656
- Процедура публикации, 673

Р

- Разбивка
 - текста, 110
- Рисование
 - выпрямление и сглаживание контуров, 68
 - корректировка анкерных точек, 65
 - корректировка сегментов контура, 66
 - оптимизация кривых, 69
 - работа со штрихами, 69
 - с помощью инструмента Brush, 67
 - Line, 63
 - Oval, 63
 - Pen, 63
 - Pencil, 63
 - Rectangle, 63
 - создание криволинейных сегментов, 64
 - прямых отрезков, 64
 - штрихов, 70

С

- Серверные включения, 629
- Сетка, 76
- Сжатие
 - ADPCM, 168
 - MP3, 160; 169
- Символы, 117
 - Unicode, 227
 - видеоклипы, 118
 - вложение, 135
 - графические, 118
 - дублирование, 123
 - замена, 131
 - имена, 132
 - кнопки, 118

- Символы
 - преобразование объекта в символ, 122
 - редактирование, 101; 123
 - редактирование в новом окне, 125
 - редактирование на месте, 123
 - режим редактирования, 127
 - создание, 119

- Система LMS, 574

- Слои
 - направляющие, 77
 - папки слоев, 31; 43
 - предназначение, 43
 - размещение звуковых файлов, 161

- События
 - onResize, связанное с рабочим полем, 351
 - Sound, 351
 - ввода пользователем, 334
 - связанные с видеоклипами, 346
 - связанные с выбором, 350
 - связанные с кнопками, 341
 - связанные с мышью, 345
 - связанные с нажатием клавиш, 343
 - связанные с текстовыми полями, 352
 - связанные со звуком, 351
 - системные, 334
 - системные, правила управления
 - последовательностью, 353
 - создание сценариев, 331
 - текстовых полей, 485; 486; 487; 488; 489; 490; 491
 - универсальные процессоры, 354
 - явное обращение к обработчикам, 338

- Сценарии
 - создание на основе временной шкалы, 330
 - создание на основе событий, 331

Т

- Текст
 - атрибуты абзаца, 103
 - атрибуты символов, 101
 - внедрение шрифтов, 108
 - встраивание шрифтов, 109
 - выбор кегля, 101
 - выбор шрифта, 101
 - выравнивание абзацев, 104
 - дескрипторы форматирования, 494
 - динамический, 106
 - динамическое создание текстового поля, 485
 - добавление к фильму, 99
 - задание отступов, междустрочного интервала и полей, 104
 - задание свойств текстового поля, 491

Текст

- изменение формы букв, 112
- инспектор свойств, 100
- инструмент Text, 47; 99
- кернинг, 102
- класс TextField, 484
- класс TextFormat, 484
- копирование текстовых полей, 114
- маркеры, 99
- межзнаковый интервал, 102
- назначение атрибутов, 47
- направление, 102
- объект TextFormat, 492
- параметры, 105
- параметры форматирования, 107
- положение символа, 103
- полоса прокрутки, 495; 543
- преобразование, 111
- привязка к пикселям, 99
- проверка орфографии, НО
- прокручивающееся текстовое поле, 542
- разбивка, 110
- растровые шрифты, 98
- редактирование, 100; 110
- редактирование отдельных символов, 101
- сдвиг контуров, 112
- стилевое оформление шрифтов, 102
- текстовые блоки, 99
- удобочитаемость, 113
- цвет заливки, 102

Текстовые узлы, 638

Тестирование

- действия отслеживания, 597
- дистанционная отладка, 598
- использование отладчика, 594
- код ActionScript, 593
- контрольные точки, 594
- окно Output, 596
- оптимизация, 592
- производительность загрузки, 592
- процедуры, 591

Тестирование и отладка, 590

Типы данных

- movieClip, 231
- null, 231; 240
- object, 229
- undefined, 231; 233; 240
- автоматическое преобразование для сравнения, 263
- булевы, 229
- массивы, 230
- неявное преобразование, 234; 238
- преобразование в булевы значения, 236

Типы данных

- преобразование в строку, 235; 238
- преобразование в число, 234; 236
- составные, 225
- специальные значения типа number, 226
- строковые, 227
- числовые, 226
- элементарные, 225
- элементарные, преобразование в объекты, 526
- явное преобразование, 234; 236

Точка регистрации

- видеоклипов, 367

Точки регистрации

- редактирование, 128

Точки регистрации, 120

Трансляция, 410

Трассировка

- оптимизация изображений, 90
- параметры, 89
- растровых изображений, 87

У

Управление браузером

- команда getURL, 603
- передача сообщений, 608
- функции JavaScript, 605

Управление проектором

- модификация окна, 614
- сокрытие контекстного меню, 615

Управляющие блоки

- циклический и нециклический, 279

Ф

Флаговое поле, 182

Фокусировка, 339; 363

Формат PNG, 680

Функции, 201; 295

- Code Hints, 202

- автоматический обзор данных, 322

- аргументы, 209; 232; 296

- возвращение значений, 305

- вызов, 298

- глобальные, 232; 372

- доступ к переменным временной шкалы, 299

- загрузка содержимого в видеоклип, 370

- идентификатор, 296

- как данные, 316

- как классы, 306

- математические, 453

Функции

методы, 206; 308
обратного вызова, 332; 450
обращения, 274
объявление, 296
определение, 205
передача аргументов по значению, 300
передача аргументов по ссылке, 301
передача данных с помощью аргументов, 300
повторный вызов, 385
псевдонимы, 322
расчет факториалов, 326
рисования кривых, оптимизация, 405
свойство prototype, 308
создание, 296
создание локальных переменных внутри, 298
функции-конструкторы, 306
функциональные литералы, 297
явный обзор данных, 319

Функции-конструкторы, 306; 419

исполнение, 529
модификация дочерних классов, 517
назначение метода посредством
объявления функции, 310
назначение метода с помощью
функционального литерала, 311
назначение методов для класса, 309
размещение локальных свойств в
иерархии наследования, 516
хранение данных о классе, 527

Функция печати фильма, 657

Ц

Цвет

в **ActionScript**, 470
заливки текста, 102
модели цветопередачи, 72
подраздел Colors панели инструментов, 50
шрифта, 102

Циклы

повторение действий, 287
создание списка, 290

Ч

Частота дискретизации, 159

Частота смены кадров, 140

Числа

представление дробных чисел с двойной
точностью, 226
с плавающей точкой, 226
целые, 226
целые, генерация, 454
экспоненциальное представление, 227

Ш

Шаблоны, 32

обучающие взаимодействия, 559
содержимое шаблонов, 560

Шрифты

без засечек, 109
внедрение, 108
встраивание, 109
выбор, 101
выбор цвета, 102
машинописные, 109
моноширинные, 109
растровые, 99
с засечками, 109
стилевое **оформление**, 102
цвет заливки, 102

Штрих

редактирование стиля, 71
создание, 70
создание стиля, 70

Э

Экземпляры

изменение свойств, 129
символов, 129

Экспортирование содержимого, 684

Элементы коллективного использования
сеансовые, 578
созданные на этапе разработки, 579

Научно-популярное издание

Майкл Гурвиц, Лора Мак-Кейб

Использование Macromedia Flash MX.

Специальное издание

Литературный редактор	<i>И.А. Попова</i>
Верстка	<i>Л.В. Терещенко</i>
Художественный редактор	<i>С.А. Чернокозинский</i>
Корректоры	<i>З.В. Александрова, Л.А. Гордиенко, О.В. Мишутина, Л.В. Чернокозинская</i>

Издательский дом "Вильяме".
101509, Москва, ул. Лесная, д. 43, стр. 1.
Изд. лиц. ЛР № 090230 от 23.06.99
Госкомитета РФ по печати.

Подписано в печать 25.07.2003. Формат 70Х100/16.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 59,31. Уч.-изд. -л. 46,28.
Тираж 4000 экз. Заказ № 331.

Отпечатано с диапозитивов в **ФГУП "Печатный двор"**
Министерства РФ по делам печати,
телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.

Мир книг по графике и обработке изображений от издательской группы “ДИАЛЕКТИКА-ВИЛЬЯМС”



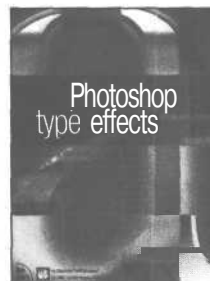
ISBN 5-8459-0423-4



ISBN 5-8459-0397-1



ISBN 5-8459-0380-7



ISBN 5-8459-0414-5



ISBN 5-8459-0472-2



ISBN 5-8459-0396-3



ISBN 5-8459-0320-3



ISBN 5-8459-0457-9



ISBN 5-8459-0381-5



ISBN 5-8459-0117-0



ISBN 5-8459-0479-X

... и много других книг Вы найдете на наших сайтах



www.dialektika.com



www.williamspublishing.com



www.ciscopress.ru

Специальное издание

ИСПОЛЬЗОВАНИЕ

Macromedia® Flash MX

ЕДИНСТВЕННАЯ НЕОБХОДИМАЯ
КНИГА ПО FLASH MX

В книге представлен справочный материал, необходимый для приобретения глубоких навыков и освоения сложных функций программы Flash. Здесь вы найдете все, что необходимо.

- * Информация, которая может понадобиться квалифицированному специалисту
- Полностью обновленный и дополненный материал, охватывающий все аспекты Flash MX, в том числе функции интерфейса и ActionScript
- Разработки опытных специалистов в области Flash
- Практические примеры работы Flash (подразделы "Flash за работой")
- * Секреты привлекательности видеоклипов, мощь компонентов и встроенных объектов
- Использование внешних данных при создании динамических Flash-фильмов
- Создание Flash-фильмов, доступных пользователям с физическими ограничениями



На прилагаемом компакт-диске содержится большое количество бесплатных файлов, которые можно использовать при обучении и работе с Flash.

Категория:	программирование для Internet/Web
Предмет рассмотрения:	Macromedia Flash MX
Уровень:	для пользователей средней и высокой квалификации

"Данная книга позволяет ускорить освоение многочисленных функций Macromedia Flash MX. Полезную информацию обучающего характера

здесь найдут как новички, так

и опытные пользователи. Подобные книги позволяют каждому пользователю внести свой вклад в создание мощного динамического содержимого Internet".

— Майкл Вильямс, заместитель руководителя отдела производства Macromedia Flash

Майкл Гурвиц — сотрудник компании Late Night Design, работает с технологиями Flash, которые являются точкой пересечения искусства, дизайна и технологии. С 1985 года он публикуется в изданиях, посвященных компьютерным технологиям. Майкл является президентом некоммерческой организации Irthlingz, занимающейся вопросами образования в области охраны природы.

Лора Мак-Кейб — независимый дизайнер. Она проживает в Балтиморе, штат Мериленд, США. У Лоры разносторонние интересы — она прослушала базовый курс по философии, имеет образование в области прикладного искусства и дизайна и, кроме того, освоила Flash. За шесть лет работы в области Internet-технологий Лора сотрудничала со многими клиентами, совершенствуя свои познания в области дизайна, разработки, информационной архитектуры и создания Web-продуктов.

ISBN 5-8459-0493-5



03087



ВИЛЬЯМС
que®

Посетите Издательский дом "Вильямс" в Internet по адресу:
<http://www.williamspublishing.com>