

Владимир Дронов

Macromedia
Flash MX

Санкт-Петербург
«БХВ-Петербург»
2002

УДК 381.3.06
ББК 32.973.26-018.2
Д75

Дронов В. А.

Д75 Macromedia Flash MX. — СПб.: БХВ-Петербург, 2002. — 848 с.: ил.
ISBN 5-94157-187-9

Книга содержит полное описание всех возможностей программного пакета-векторной графики и анимации Macromedia Flash. Описываются средства для создания статической графики, анимации и интерактивных элементов. Рассматриваются возможности по включению созданных графических изображений в Web-страницы (основная область применения Flash-графики). Приводятся полезные советы по работе с пакетом и описание выполнения типичных задач.

Для начинающих и опытных Web-дизайнеров и Web-программистов

УДК 381.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Анна Кузьмина</i>
Редактор	<i>Леонид Кочин</i>
Компьютерная верстка	<i>Татьяны Олоновой</i>
Корректор	<i>Татьяна Звертановская</i>
Дизайн обложки	<i>Игоря Цырульниково</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 26.08.02.

Формат 70×100¹/₁₆. Печать офсетная. Усл. печ. л. 68,37.

Тираж 4000 экз. Заказ №

"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953 Д.001537.03.02 от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов

в ФГУП ордена Трудового Красного Знамени "Техническая книга"
Министерства Российской Федерации по делам печати,
телерадиовещания и средств массовых коммуникаций.
198005, Санкт-Петербург, Измайловский пр., 29.

ISBN 5-94157-187-9

© Дронов В. А., 2002

© Оформление, издательство "БХВ-Петербург", 2002

СОДЕРЖАНИЕ

Введение.....	17
Компьютерная графика: прошлое и настоящее	18
Что дает нам Flash.....	20
Компактность	20
Безопасность.....	21
Интерактивность	22
Что еще?	22
Что делают на Flash.....	24
 ЧАСТЬ I. ОСНОВНЫЕ ПРИНЦИПЫ РАБОТЫ С FLASH	27
 Глава 1. Основы пользовательского интерфейса Flash	29
Как работать в среде Flash	29
Просмотр графики Flash.....	31
Среда Flash	33
Главное окно программы	33
Управление окнами и панелями Flash.....	39
Работа с окном документа.....	43
Окно документа.....	43
Принципы работы с графикой Flash	45
Управление окном документа.....	46
Средства позиционирования	49
 Глава 2. Типовые функции Flash.....	54
Файловые операции	55
Создание нового документа	55
Создание нового документа на основе шаблона	57
Работа с документами	59
Создание нового шаблона.....	60
Печать.....	61
Интерактивная справка Flash.....	64

Глава 3. Настройка Flash	69
Настройка программы	70
Настройка клавиатурных комбинаций	79
ЧАСТЬ II. РАБОТА СО СТАТИЧНОЙ ГРАФИКОЙ	83
Глава 4. Форматы статичной графики	85
Растровая и векторная графика	86
Растровая графика	86
Векторная графика	90
Гибридная графика	92
Применение разных видов графики	93
Форматы графических файлов	94
Растровые форматы	94
BMP	94
PCX	94
GIF	95
PNG	96
JPEG	97
TIFF	98
Другие растровые графические форматы	98
Векторные форматы	99
Shockwave/Flash	99
Windows Metafile и Enhanced Windows Metafile	100
Adobe Illustrator	100
CorelDRAW!	101
Encapsulated PostScript	101
VML	101
Другие векторные графические форматы	102
Другие форматы	102
PDF	102
VRML	103
Использование графических форматов	103
Глава 5. Рисование	105
Использование инструментов рисования	106
Базовые средства управления цветом	106
"Линия"	107
"Прямоугольник" и "эллипс"	108
"Карандаш"	109
"Перо"	111
"Кисть"	114

"Ведро с краской"	117
Правка графики	119
Выделение графики.....	119
Фрагментация и слияние графики.....	122
Группировка.....	125
Правка графики.....	126
Перемещение и удаление графики	126
"Притягивание" графики	126
Работа с буфером обмена	127
Изменение цвета графики.....	129
Изменение формы фигуры	129
Точная правка кривых.....	131
Сложное выделение. "Лассо"	133
Стирание графики. "Ластик".....	136
Дополнительные возможности работы с контурами	139
Отмена и повтор выполненных действий	145
Глава 6. Работа с цветом	146
Инструменты выбора цвета.....	147
Использование селектора цвета	147
Область <i>Colors</i> инструментария.....	149
Создание новых цветов. Смеситель	149
Работа с линиями.....	152
Задание параметров линий	152
"Пузырек с чернилами"	156
Работа с заливками.....	156
Виды заливки.....	156
Простейшие виды заливки.....	156
Градиентная заливка.....	157
Использование растрового изображения в качестве заливки.....	160
Настройка заливки.....	161
Фиксация заливки.....	165
"Пипетка".....	166
Работа с палитрами	167
Глава 7. Работа с текстом.....	170
Текстовые блоки.....	171
Работа с текстовыми блоками.....	171
Форматирование.....	173
Форматирование текста.....	173
Поддержка шрифтов во Flash	176
Форматирование абзаца	178
Параметры текстового блока	179

Специальные текстовые блоки	180
Поля ввода	180
Динамические текстовые блоки	184
Работа с символами текста как с графикой	186
Подстановка шрифтов	187
Глава 8. Импорт графики	190
Поддержка графических форматов	191
Список поддерживаемых форматов	191
Импорт графики	192
Особенности поддержки графических форматов	193
Macromedia Fireworks	193
Macromedia Freehand	194
Adobe Illustrator	196
AutoDesk AutoCAD	197
Работа с импортированной графикой	197
Векторизация растровой графики	197
Разбиение растровой графики. "Волшебная палочка"	199
Замена одного изображения на другое	201
Задание параметров растрового изображения	202
Глава 9. Работа с графическими фрагментами	204
Простейшие манипуляции	204
Изменение порядка наложения	204
Выравнивание	205
Перемещение и изменение размеров	208
Зеркальное отражение	210
Более сложные манипуляции	210
Изменение размеров	210
Вращение и сдвиг	213
Искажение формы	214
Деформация	215
Свободная трансформация	217
Дополнительные возможности	217
Преобразования копии графического фрагмента	217
Сброс всех преобразований графического фрагмента	218
Блокировка фрагмента	218
Глава 10. Образцы и библиотеки. Проводник Flash	220
Работа с образцами и экземплярами	222
Типы образцов	222
Создание образцов	223

Создание экземпляров.....	226
Преобразование экземпляров	227
Изменение цвета экземпляров	227
Преобразования экземпляра	229
Изменение типа экземпляра.....	229
Смена экземпляра	229
Преобразование экземпляра в обычный графический элемент.....	231
Работа с образцами	231
Изменение образцов.....	231
Дублирование образцов	234
Работа с библиотекой	235
Окно библиотеки.....	235
Управление образцами.....	237
Использование папок	240
Совместное использование образцов и библиотек	244
Копирование образцов из документа в документ	245
Обновляемые образцы	246
Разделяемые образцы.....	247
Создание разделяемых образцов	249
Использование разделяемых образцов	250
Разрешение конфликта имен.....	251
Образцы-шрифты.....	252
Превращение разделяемого образца в обычный	254
Библиотеки общего использования	254
Проводник Flash.....	255

Глава 11. Публикация и экспорт статичной графики.....259

Публикация изображения	259
Выбор формата публикации	260
Задание формата публикации	261
Настройки форматов публикации	262
GIF.....	262
JPEG	265
PNG	266
Предварительный просмотр публикуемой графики	269
Экспорт изображения	269
Форматы экспорта, поддерживаемые Flash	270
Экспорт графики	270
Adobe Illustrator.....	271
GIF.....	271
BMP	272
AutoDesk AutoCAD.....	273
JPEG	273

PNG	274
Shockwave/Flash	276
ЧАСТЬ III. РАБОТА С АНИМАЦИЕЙ.....	277
Глава 12. Форматы анимированной графики и видео	279
Покадровая и трансформационная анимация	281
Покадровая анимация	281
Трансформационная анимация	284
Применение разных видов анимации	286
Форматы видеофайлов	287
QuickTime	287
AVI	287
MPEG	288
DivX	289
WMV	290
RealMedia.....	290
Другие видеоформаты	291
Использование видеоформатов	291
Глава 13. Покадровая анимация	292
Создание покадровой анимации	293
Использование временной шкалы. Создание кадров	293
Просмотр фильма в среде Flash.....	298
Правка анимации	300
Работа с кадрами	300
Дополнительные возможности работы с кадрами	302
Дополнительные возможности временной шкалы.....	305
Использование сцен.....	307
Дополнительные возможности Проводника Flash.....	310
Глава 14. Трансформационная анимация.....	312
Трансформация движения.....	313
Создание простейшей трансформации движения	313
Более сложная трансформация движения.....	317
Параметры трансформации движения.....	318
Трансформация формы	320
Создание простейшей трансформации формы	321
Параметры трансформации формы	323
Маркеры трансформации и их использование.....	324
Использование обоих видов трансформаций	327
Вложенная анимация	328

Глава 15. Слои	335
Применение слоев.....	336
Создание и использование слоев	336
Управление слоями	342
Использование папок в списке слоев.....	345
Специальные слои.....	349
Слои-направляющие	349
Маскирующие слои.....	351
Поддержка слоев Проводником Flash.....	353
Глава 16. Импорт анимации и видео	355
Поддержка форматов анимации и видео.....	356
Список поддерживаемых форматов	356
Как Flash обрабатывает внедренные видеофайлы.....	357
Полезные советы по подготовке видео для импорта	358
Импорт видео и работа с ним.....	359
Внедрение видео.....	360
Связывание видео	364
Использование импортированного видео	365
Работа с импортированными клипами	366
Изменение экземпляра клипа.....	367
Глава 17. Работа со звуком	369
Представление звуковой информации.....	370
Кодирование и хранение звуковых данных	370
Форматы звука, поддерживаемые Flash.....	372
WAV	372
MP3	373
Другие форматы звука, поддерживаемые Flash	373
Форматы звука, не поддерживаемые Flash.....	374
WMA	374
RealMedia.....	374
MIDI	375
Трекерные модули.....	375
Импорт звука и работа с ним	375
Импорт звука	375
Использование звука в фильме	376
Правка звука средствами Flash	379
Работа с импортированными звуками.....	381
Задание параметров звука	383

Глава 18. Подготовка к экспорту.....	386
Оптимизация фильма.....	387
Профилировщик загрузки.....	387
Оптимизация фильмов	391
Доступность фильма	392
Технология "чтение с экрана".....	392
Поддержка "чтения с экрана" Flash	393
Дополнительные параметры доступности.....	394
Полезные советы по созданию доступных фильмов.....	396
 Глава 19. Публикация и экспорт анимации	 398
Публикация фильма.....	399
Выбор формата публикации	399
Публикация фильма.....	400
Shockwave/Flash	400
HTML	402
QuickTime	407
GIF.....	409
Экспорт фильма.....	409
Форматы экспорта, поддерживаемые Flash	409
Экспорт анимации	410
Shockwave/Flash и FutureSplash	411
AVI	411
QuickTime	413
GIF (анимированный).....	414
WAV (только звук)	415
Создание фильмов, предназначенных для печати	415
 ЧАСТЬ IV. ПРОГРАММИРОВАНИЕ	 417
 Глава 20. Основы программирования	 419
Зачем нужны сценарии.....	420
Создание сценариев и работа с ними	421
Знакомство с панелью <i>Actions</i>	421
Написание сценариев	425
Работа с панелью <i>Actions</i>	430
Работа в обычном режиме.....	430
Работа в профессиональном режиме	434
Использование внешнего текстового редактора	438
Поддержка сценариев Проводником Flash	439

Глава 21. Язык ActionScript	440
Начала ActionScript	440
Что такое сценарий	440
Типы данных	442
Строковый	442
Числовой	443
Логический	444
Объект, клип и функция	444
<i>null</i>	444
<i>undefined</i>	444
Константы	444
Переменные	444
Именованые переменных	445
Объявление и использование переменных	445
Видимость переменных	446
Системные переменные	447
Операторы	448
Арифметические	448
Объединения строк	449
Двоичные	449
Присваивания	450
Сравнения	451
Логические	452
Оператор <i>typeof</i>	453
Совместимость типов данных	453
Приоритет операторов	454
Действия	456
Комментарии	456
Сложные выражения	457
Блоки	457
Условные выражения	457
Выражения выбора	459
Циклы	460
Цикл со счетчиком	461
Цикл с постусловием	462
Цикл с предусловием	462
Прерывание цикла	463
Обработчики событий	463
Функции	464
Создание функций	465
Вызов функций	466
Рекурсия	467
Встроенные функции ActionScript	468

Массивы	468
Объекты	470
Понятия объекта и экземпляра	470
Работа с объектами	470
Несколько новых операторов и действий	472
Встроенные объекты ActionScript	473
<i>String</i>	474
<i>Number</i>	475
<i>Boolean</i>	475
<i>Date</i>	475
<i>Array</i>	476
<i>Function</i>	476
<i>Arguments</i>	477
<i>Math</i>	477
<i>Object</i>	478
Пользовательские объекты	478
Создание экземпляров объекта <i>Object</i>	478
Создание нового объекта	478
Создание методов и свойств	479
Создание динамических свойств	480
Наследование	481
Внешние объекты	483
Глава 22. Создание интерактивных фильмов	484
Объект <i>movieClip</i>	484
Зачем нужны встроенные клипы	485
Использование клипов	486
Имена и пути доступа к клипам	486
Накладываемые клипы. Уровни наложения	489
Простейшие манипуляции над клипами	490
Управление проигрыванием клипа	490
Загрузка и выгрузка клипов	492
Обеспечение безопасности при загрузке клипов	493
Использование обработчиков событий	494
Управление экземплярами-клипами	496
Создание и удаление экземпляров-клипов	496
Изменение параметров встроенных клипов	498
Drag'n'drop	500
Создание фигурного курсора мыши	502
Выявление совпадений	503
Работа с графикой	504
Рисование	505
Пример использования операций рисования	508

Работа с масками.....	509
Объект <i>Key</i>	509
Свойства и методы объекта <i>Key</i>	509
Обработка нажатий клавиш.....	511
Использование объектов-перехватчиков.....	512
Объект <i>Mouse</i>	514
Объект <i>Color</i>	514
Объект <i>Sound</i>	516
Свойства и методы объекта <i>Sound</i>	516
Создание органов управления звуком.....	518
Загрузка и выгрузка звуков.....	521
Объект <i>Stage</i>	522
Объект <i>System.capabilities</i>	523
Использование таймеров.....	524
Создание библиотек кода.....	527
Глава 23. Создание приложений Flash.....	529
Кнопки.....	530
Создание кнопок.....	530
Объект <i>Button</i>	533
Поля ввода и динамические текстовые блоки.....	534
Объект <i>TextField</i> и его использование.....	535
Пример приложения Flash, использующего поля ввода.....	539
Форматирование текста.....	540
Объект <i>Selection</i>	542
Элементы управления.....	543
Компоненты и работа с ними.....	544
Понятие о компонентах.....	544
Работа с компонентами в среде Flash.....	545
Встроенные элементы управления Flash.....	548
Флажок (<i>CheckBox</i>).....	548
Раскрывающийся список (<i>ComboBox</i>).....	549
Обычный список (<i>ListBox</i>).....	551
Кнопка (<i>PushButton</i>).....	552
Переключатель (<i>RadioButton</i>).....	553
Полоса прокрутки (<i>ScrollBar</i>).....	554
Панель с прокруткой (<i>ScrollPane</i>).....	555
Программирование пользовательского интерфейса.....	557
Работа с элементами управления из сценариев.....	557
Написание обработчиков событий.....	559
Пример приложения Flash, использующего элементы управления.....	560
Тонкая настройка компонентов.....	563
Настройка стилей компонентов.....	563

Изменение "шкур" компонентов.....	565
Создание пользовательского элемента управления	567
Создание образца-клипа	568
Написание сценариев ActionScript.....	569
Создание параметров компонента	570
Создание значка компонента	573
Создание описания компонента	574
Создание клипа "живого просмотра"	579
Создание панели параметров.....	581
Последние штрихи	585
Установка компонента во Flash.....	585
Глава 24. Работа с внешними приложениями	587
Управление внешними приложениями.....	588
Загрузка Web-страницы.....	589
Управление проигрывателем Flash.....	589
Взаимодействие со сценариями JavaScript	591
Использование внешних данных	594
Посылка запроса серверному приложению	594
Получение данных от серверного приложения	597
Использование объекта <i>LoadVars</i>	598
Использование данных XML	600
Вводный курс языка XML.....	600
Использование объекта XML	602
Использование объекта <i>XMLSocket</i>	605
Глава 25. Средства отладки сценариев ActionScript	608
Как выявить ошибки	609
Окно <i>Output</i>	611
Отладчик Flash.....	614
Использование отладчика Flash	614
Просмотр значений переменных и свойств и списка вызовов	616
Использование точек останова.....	618
Трассировка кода.....	620
Удаленная отладка фильмов Flash.....	621
Заключение.....	623
ЧАСТЬ V. ПРИЛОЖЕНИЯ.....	625
Приложение 1. Элементы языка ActionScript	627

Приложение 2. Внедрение фильмов Flash в Web-страницы	814
Как графика Flash помещается на Web-страницы	814
Теги <i><OBJECT></i> и <i><EMBED></i>	815
Тег <i><OBJECT></i>	815
Тег <i><EMBED></i>	816
Использование тегов <i><OBJECT></i> и <i><EMBED></i>	816
Атрибуты тегов <i><OBJECT></i> и <i><EMBED></i>	818
<i>ALIGN</i>	818
<i>CLASSID</i>	818
<i>CODEBASE</i>	818
<i>HEIGHT</i>	819
<i>PLUGINSPAGE</i>	819
<i>SRC</i>	819
<i>SWLIVECONNECT</i>	819
<i>WIDTH</i>	820
Параметры проигрывателя Flash, реализованного в виде компонента	
ActiveX	820
<i>BASE</i>	820
<i>BGCOLOR</i>	820
<i>DEVICEFONTS</i>	820
<i>LOOP</i>	820
<i>MENU</i>	821
<i>MOVIE</i>	821
<i>PLAY</i>	821
<i>QUALITY</i>	821
<i>SALIGN</i>	822
<i>SCALE</i>	822
<i>WMODE</i>	822
Шаблоны HTML	823
Макросы	823
Примеры использования макросов	826
Карты-изображения	826
Замещающие изображения	828
Текст фильма и список гиперссылок	828
Предметный указатель	829

Введение

Вы держите в руках книгу, посвященную последней на данный момент версии программного пакета Macromedia Flash — Flash MX, она же Flash 6. А это значит, что вы им заинтересовались. Так что же такое Macromedia Flash? И что он вам может дать?

Скажем сразу, что Flash — пакет компьютерной графики и формат сохранения ее в файле. Скажем больше: это пакет для создания и формат для сохранения двумерной анимированной компьютерной графики, предназначенной, в основном, для публикации в Интернете. Скажем проще: это средство создания мультиков, которые вы можете выложить в Сеть. Скажем чистую правду: именно Flash принес в Интернет высококачественную и компактную анимацию. Скажем еще кое-что: Flash породил целый вид искусства, известный как "Flash-анимация", и целую касту деятелей этого искусства, известную как "Flash-аниматоры". Скажем банальность: Flash — это современно. Скажем пошлость: Flash — это модно.

На сегодняшний момент существует множество Web-сайтов, построенных с использованием технологии Flash. Есть также довольно много программ, использующих для тех или иных целей Flash-графику. (Например, замечательный проигрыватель мультимедийных файлов J. RiverMedia Jukebox, с которым вы можете ознакомиться по адресу <http://www.musicex.com/mediajukebox>. Этот проигрыватель не только показывает Flash-фильмы, но и хранит некоторые части своего интерфейса в формате Flash.) Создано большое количество неплохих Flash-фильмов, которые вы можете увидеть на сайтах <http://www.mp4.com>, <http://www.hypnotic.com>, <http://atomfilms.shockwave.com> и др. Существует Дмитрий Дибров, показывающий в телепередаче "Ночная смена" потрясающие по своей невразумительности Flash-ролики. И, наконец, существует целое сообщество "флэшеров", в которое при желании можете влиться и вы.

Так для кого же эта книга? Для тех, кто хочет научиться рисовать в среде Flash. Только и всего.

Ниже мы подробнее рассмотрим все возможности, предлагаемые Macromedia Flash. Но, прежде всего, давайте узнаем побольше о современной компьютерной графике и выясним, зачем нужно было изобретать еще один графический формат. И, поскольку Flash, в первую очередь, предназначен для создания интернет-графики, особое внимание мы уделим компьютерным сетям.

Компьютерная графика: прошлое и настоящее

История компьютерной графики неотделима от истории персональных компьютеров. В самом деле, во времена больших ЭВМ компьютерная графика если и существовала, то носила чисто утилитарный характер. Например, нарисовать зелеными линиями на черном фоне простейший график, основанный на результатах каких-либо расчетов. Или вывести на экран только что спроектированную печатную плату. Как видите, тогдашние ЭВМ использовались только для дела. А, как поется в одной старой песне, "первым делом — самолеты"

Персональный компьютер (ПК) произвел настоящую революцию в мире вычислительной техники. Изначально подразумевалось, что это чудо может использоваться своим хозяином не только для дела, но и для потехи. (И делу время, и потехе час — так гласит пословица.) А для этого "персоналке" нужны хорошие возможности по выводу сложных графических изображений. Неудивительно, что даже на заре новой эпохи только самые дешевые ПК имели видеоадаптер, приспособленный исключительно для вывода текста (как говорят профессиональные компьютерщики, алфавитно-цифровой).

И первой такой потехой были компьютерные игры. Индустрия игр началась также вместе с ПК. На больших компьютерах игры просто не существовали (хотя, по слухам, первая компьютерная игра была написана именно для большой ЭВМ). Когда же появились и получили распространение "персоналки", появилась и игровая индустрия. И не просто появилась, а вскоре стала подлинным двигателем прогресса в этой области. Ведь зачем совершенствуются трехмерные ускорители, звуковые карты, зачем выпускаются умопомрачительные джойстики — неужели для Microsoft Word или 1С Бухгалтерии? Отнюдь! Лишь для того, чтобы любимая игрушка шла чуть-чуть быстрее, монстры выглядели чуть-чуть реалистичнее, а играть в нее было чуть-чуть удобнее.

Но, кажется, мы отвлеклись. Вернемся к нашей графике.

Компьютерная графика эволюционировала вместе с компьютерным "железом" и программным обеспечением. Сначала это были корявые картинки, выполненные в шестнадцать цветов огромными пикселями. С совершенствованием графических подсистем и мониторов изображение на них

стало выглядеть больше похожим на оригинал и меньше — на плохую мозаику. По мере совершенствования графических файловых форматов, алгоритмов сжатия и программ, обрабатывающих графику, качество изображения улучшалось, а размеры графического файла уменьшались. В результате компьютерные художники из немногочисленных сумасбродов превратились в настоящих профессионалов своего дела.

Сейчас все книги, журналы и газеты верстаются на компьютерах, в Интернете полно электронных репродукций картин и фотографий на любые вкусы (и на любое безвкусие), качество компьютерных игр приближается к качеству блокбастеров десятилетней давности, а сами блокбастеры помещаются на обычных компакт-дисках с надписью "DivX". Казалось бы, все замечательно. Так зачем нужен еще один графический формат — Macromedia Flash?

Для того чтобы ответить на этот вопрос, нам нужно обратиться к интернет-графике. Именно для Интернета и был создан Flash.

Что такое Интернет? Нет, не так... Что такое Интернет для большинства его пользователей? Электронная почта (отмечаем сразу, ибо не наш профиль), вирусы (давить!), хакеры (давить!) и Всемирная Паутина, или, как говорят еще, *World Wide Web*, она же *WWW* или просто *Web*. Вот на ней мы остановимся подробнее.

Что такое Всемирная Паутина? Это *Web-странички*. А что такое *Web-странички*? Это текст, который можно читать, графика, которую можно смотреть, музыка, которую можно слушать, видео, которое также можно смотреть, и файлы, которые можно скачать. Если вы не новичок в Интернете, то сами знаете, что такое Паутина. Она живет, взаимодействует с вами, реагирует на вас и ваши действия. То есть, она интерактивна, иначе говоря, общительна.

Почтовые Web-серверы запрашивают у вас имя и пароль и выдают вам именно вашу почту. Новостные сайты выводят список самых последних событий "бегущей строкой", позволяя вам щелкнуть по нужному заголовку и прочесть больше. Файловые архивы регистрируют каждое скачивание каждого хранящегося на них файла. А сайты с онлайн-играми? И "продвинутые" сайты с "продвинутыми" интерфейсными элементами, наподобие всплывающих меню или картинок, ползающих за курсором мыши? И концептуальные сайты с потрясающе красивыми заставками? Неужели вы не видели всего этого?!!

А ведь были времена, когда Интернет был полностью текстовым. Да-да, не удивляйтесь. Тим Бернерс-Ли, создавший в 1989 году язык *HTML* (HyperText Markup Language — язык гипертекстовой разметки), на котором и пишутся Web-страницы, не предусмотрел в нем поддержку графики. Потом, правда, под нажимом общественности консорциум *WWWC* (World Wide Web Consortium — консорциум всемирно протянутой паутины, ну

и название!..), занимающийся развитием языка HTML, внес в него некоторые изменения. Так в Интернет пришла графика.

Но другой недостаток HTML не преодолен до сих пор. Это его порочная неинтерактивность. В самом деле, если вы хотите сделать на своей Web-странице всплывающее меню со ссылками на другие страницы, одним лишь HTML вам не обойтись. Вам нужно будет встраивать в HTML-код страницы программы, написанные на специальном языке *JavaScript*. Такие программы называются *сценариями* и служат для управления поведением той или иной части Web-страницы в ответ на то или иное действие пользователя. Как видите, врожденный порок инвалида — HTML — пытаются преодолеть с помощью костылей — сценариев. И не всегда это получается.

Вся эта связка — HTML+JavaScript — работает из рук вон плохо. Дело в том, что различные программы *Web-обозревателей* (специальные программы для просмотра Web-страниц, называемые также Web-браузерами от английского browser — обозреватель) поддерживают JavaScript-сценарии (как и сам HTML) по-разному. Эта пресловутая несовместимость Web-обозревателей отравила жизнь многим Web-дизайнерам. И выхода — увы! — не предвидится...

Есть еще одна проблема, связанная с некоторыми особенностями JavaScript. Если вы собираетесь реализовать на этом языке какой-либо сложный алгоритм, являющийся ноу-хау, будьте готовы к тому, что его очень быстро украдут. Дело в том, что программу, написанную на JavaScript, можно просмотреть с помощью обычного Блокнота, поставляемого в составе Windows. А уж разобраться, что он делает, может любой школьник, знакомый с основными навыками JavaScript-программирования.

И вот тут на сцене появляется Macromedia Flash. Он может решить — и решает — все наши проблемы. Дадим же ему слово.

Что дает нам Flash

То, что дает нам Flash, можно выразить тремя словами: компактность, безопасность, интерактивность. Давайте рассмотрим подробно, что же скрывается за всем этим.

Компактность

Файлы, хранящие изображения Flash, на самом деле очень компактны. Это достигается одной особенностью, которую мы во всех подробностях рассмотрим позже. А пока проведем такую аналогию.

Возможно, вы писали какие-либо программы на одном из *компилируемых языков программирования* (Pascal, C++, Visual Basic и т. п.). При этом процесс написания программы выглядит следующим образом. Вы пишете исходный текст программы в виде текстового файла; при этом текст выглядит

так, что с ним удобно работать (конечно, удобно тому, кто знаком с этим языком программирования). После этого вы запускаете особую программу — *компилятор* — и передаете ей получившийся текстовый файл. Компилятор преобразует набор команд языка в набор инструкций процессора — выполняет его компиляцию — и сохраняет его в исполняемом файле с расширением *exe* (*com*, *dll*, *osx* и др.). Если вы откроете исполняемый файл в текстовом редакторе, вы ничего не поймете — этот файл представляет собой мешанину шестнадцатеричных цифр, в которой нет ничего от исходного текстового файла. Однако процессор компьютера прекрасно понимает эту мешанину — для него этот язык "родной".

Так же поступает и Flash. При публикации Flash-изображения оно подвергается такому же преобразованию. В результате создается исключительно компактный файл формата *Shockwave/Flash* с расширением *swf*, аналог исполняемого файла обычной программы. Этот файл впоследствии может быть загружен и просмотрен с помощью специального *проигрывателя* Flash, который можно рассматривать как аналог процессора компьютера.

Безопасность

Компилируемые языки программирования имеют еще одно неоспоримое достоинство. После того, как вы откомпилируете исходный текст своей программы, практически невозможно будет выполнить обратное преобразование — получить из откомпилированного кода исходный текст. По слухам, где-то якобы существуют программы для такого обратного преобразования, но никто их в глаза не видел, так что, скорее всего, это только слухи. Конечно, теоретическая возможность того, что кто-то сможет выяснить, как работает программа, существует, но это связано с такими трудностями, что вряд ли кто за это возьмется.

Таким образом, вы можете реализовать в коде программы какие-либо ноу-хау без боязни того, что кто-то их позаимствует. В отличие от языков типа JavaScript, где программа фактически хранится в виде исходного текста и перед каждым выполнением расшифровывается (*интерпретируемые языки программирования*). Как вы уже знаете, программу на JavaScript может просмотреть любой.

Файл Shockwave/Flash в этом смысле аналогичен откомпилированной программе. При экспорте вы можете задать различные параметры секретности. И если вы написали программу в среде Flash, то есть уверенность, что никто не доберется до ее исходного кода.

Flash может быть использован для публикации текстов, которые не могут быть скопированы. Как вы знаете, любой пользователь Web-обозревателя имеет возможность выделить отображаемый в нем текст, скопировать его в буфер обмена Windows, вставить в текстовый редактор и сохранить. Если же вы набираете какой-либо текст во Flash, вы можете запретить его копировать.

Интерактивность

Большинство современной интернет-графики удручающе неинтерактивно. Вы не можете выбрать какое-либо место на изображении и заставить его менять цвет при наведении на него курсора мыши. То есть в принципе, можете, но для этого вам придется приложить очень много усилий, изучить, кроме любимой программы графического редактора, еще HTML и JavaScript, и некоторое время поколдовать с целым ворохом дополнительных файлов. И ведь изучают, и колдуют, иной раз уподобляясь шаману с бубном.

Было бы неплохо, если бы в каком-либо пакете имелась изначальная возможность создания анимированной интерактивной графики. В таком случае работы у Web-дизайнеров уменьшилось бы на порядок. А сколько нервных клеток бы сохранилось! Ведь связка HTML+JavaScript, как вы знаете, работает из рук вон плохо; разные программы Web-обозревателей обрабатывают ее по-своему, и добиться единообразия очень и очень трудно.

Долгое время эти мечты оставались лишь мечтами. Но несколько лет назад на горизонте появился Macromedia Flash и сделал их реальностью.

Работая в среде Flash, вы можете писать программы на встроенном языке программирования ActionScript, аналогичном JavaScript, для описания поведения тех или иных элементов изображения. Эти программы при публикации изображения будут откомпилированы. Язык ActionScript достаточно богат, чтобы писать на нем весьма сложные программы. В частности, автору встречались настоящие системные утилиты, интерфейсы к сложным Web-сайтам и даже трехмерные игры, написанные на Macromedia Flash. (Все это вы можете посмотреть на сайте <http://www.flasher.ru>.) Таким образом, ActionScript позволяет строить полноценный графический интерфейс, аналогичный по богатству возможностей интерфейсу Windows-приложений.

Вот такие возможности предоставляет нам Flash. Но как их реализовать? Это и объясняет настоящая книга.

Но стоит ли использовать Flash? Не лучше ли подождать, пока появится что-нибудь получше? Давайте поговорим на эту тему.

Что еще?

Речь пойдет о конкурентах технологии Macromedia Flash. И о том, что они могут нам дать (желательно, такого, что не может дать сам Flash).

Главнейшими конкурентами Flash являются уже существующие технологии и стандарты. Это, прежде всего, языки HTML и JavaScript. Относительная простота создания документов (в том числе, интернет-документов) с помощью этих языков, ориентированность на тексты, обилие различных программ, поддерживающих эти языки, обеспечивают HTML и JavaScript

большую популярность, которой Flash пока что похвастаться не может. О недостатках мы уже говорили: "открытость" кода документов и программ и различия в поддержке этих языков разными Web-обозревателями.

Если HTML и JavaScript стоят несколько в стороне, обеспечивая не столько представление графики, сколько распространение текстов и описание поведения различных фрагментов HTML-документов, то существующие графические форматы занимают ту же нишу, что Flash. Таких форматов три: BMP (BitMaP — битовый массив), GIF (Graphic Interchange Format — формат обмена графикой) и JPEG (Joint Picture Encoding Group — группа кодирования неподвижных изображений). Первый формат используется, в основном, для распространения "обоев" рабочего стола Windows, вторые два — вообще для распространения графики, в том числе, и в Интернете.

К достоинствам перечисленных выше графических форматов можно отнести широкую поддержку их со стороны разнообразного программного обеспечения. Да и они, в конце концов, справляются со своими задачами. Файлы этих форматов получаются достаточно компактными (за исключением файлов BMP) при хорошем качестве изображения. Недостатки — зависимость размеров файла от геометрических размеров изображения и плохие возможности по масштабированию (увеличению и уменьшению) и любой другой обработке графики при ее выводе.

Современные форматы сохранения фильмов также имеют свою нишу. В настоящее время популярны форматы AVI (Audio and Video Interlaced — чередующиеся аудио и видео), Apple QuickTime и MPEG (Motion Picture Encoding Group — группа кодирования движущихся изображений) версий I, II и IV. Они также широко поддерживаются программным обеспечением и обеспечивают приемлемое качество графики и звука. К тому же, к настоящему времени накоплено множество фильмов, записанных в этих форматах. К их недостаткам можно отнести большие размеры файлов, получаемые при сохранении фильмов хорошего качества.

Уже знакомый вам формат GIF также позволяет сохранять фильмы. Но его возможностей хватает только на короткие видеоролики, используемые, в основном, в рекламных целях.

В последнее время появились технологии, позволяющие внедрять в Web-страницы небольшие программы, которые можно использовать, в том числе, для отображения различной информации и создания пользовательских интерфейсов. Существует две разновидности таких программ: *апплеты Java* и *компоненты ActiveX*. Апплеты (от английского applet — "приложеньице") Java пишутся на межплатформенном языке программирования Java, разработанном фирмой Sun. Компоненты ActiveX были разработаны фирмой Microsoft и пишутся на любом языке программирования, поддерживающим их создание. К достоинству этих технологий можно отнести возможность реализации логики любой сложности, к недостатку — трудность создания.

И, наконец, самая "горячая" новинка современных интернет-технологий. Это язык описания документов и, одновременно, программирования Curl, разработанный фирмой Curl Corporation (<http://www.curl.com>). Curl позиционируется как "могильщик" языков HTML, JavaScript, апплетов Java, компонентов ActiveX и практически всех графических форматов, используемых в Интернете, в том числе и Flash. Достоинством этого языка выступает простота написания документов и программирования любых сценариев, а недостатком — пока что очень малая распространенность.

Как видите, все вышеперечисленные технологии имеют серьезные недостатки, препятствующие их распространению. Только Flash сочетает такие особенности, как широкая распространенность, простота создания графики и реализация программной логики пользовательского интерфейса, высокое качество графики, богатые возможности по ее обработке и компактность получаемого файла. Неудивительно, что 95% пользователей Интернета имеют на своих компьютерах установленный модуль расширения Web-обозревателя, позволяющий просматривать графику Flash.

Итак, чем хорош Flash, мы выяснили. Осталось выяснить, зачем его обычно применяют.

Что делают на Flash

Давайте перечислим, какие виды графики обычно создаются с использованием Macromedia Flash.

- ❑ Несложные статические изображения, в основном, элементы оформления Web-страниц. Для создания сложной графики все-таки лучше применить более мощные графические редакторы.
- ❑ Рекламные баннеры, как правило, анимированные.
- ❑ Небольшие фильмы для помещения на Web-страницы. Это могут быть учебные, рекламные, развлекательные или полноценные художественные ленты. Такие фильмы вы можете увидеть на сайтах, распространяющих интернет-фильмы, и в передаче "Ночная смена".
- ❑ Небольшие программы для помещения на Web-страницы. Чаше всего встречаются онлайн-игры или различные "приколы". (Автору однажды встретилась Flash-программа, помогающая придумать название для рок-группы.) Но иногда можно найти весьма полезные утилиты, написанные на Flash.
- ❑ Интерфейсы для различных интернет-сервисов, например, почтовых серверов, интернет-магазинов или справочных баз данных.
- ❑ Полноценные Web-сайты. В основном, на Flash делаются развлекательные или "продвинутые" сайты многочисленных исполнителей. (Автор может привести в качестве примеров "стильный" сайт английской группы

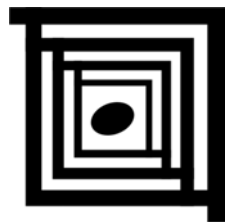
"Death in June" и потрясающий по красоте портал, посвященный владивостокскому року. К сожалению, ссылки на эти сайты потеряны).

Осталось привести список всего того, что не делается в формате Flash.

- ❑ Обычный текст. Его проще всего оформить в формате HTML и выложить в Сеть для всеобщего обозрения. Если же вы хотите, чтобы ваши тексты прочитали только избранные, распространяйте их по электронной почте.
- ❑ Обычная графика: схемы, фотографии, картины и т. п. Их распространяют в форматах GIF и JPEG. Штриховые рисунки — схемы, гравюры, карты и т. п. — лучше распространять в формате GIF, а полутоновые — картины и фотографии — в формате JPEG.
- ❑ Любые более-менее сложные программы. Их проще написать на любом известном вам языке программирования, откомпилировать и выложить в виде файлового архива. Также вы можете оформить вашу программу как апплет Java или компонент ActiveX, в крайнем случае — как сценарий JavaScript.
- ❑ Документы, предназначенные для печати на бумаге. Ну, это совсем глупо!

Вот и все. Теперь вы знаете, зачем нужен и что может дать вам программный пакет Macromedia Flash. Начальную информацию вы получили. Чтобы узнать все его возможности во всех подробностях, читайте эту книгу!

Часть I

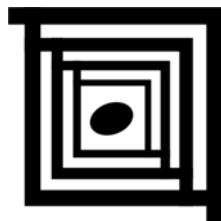


Основные принципы работы с Flash

Глава 1. Основы пользовательского интерфейса Flash

Глава 2. Типовые функции Flash

Глава 3. Настройка Flash



Глава 1

Основы пользовательского интерфейса Flash

В этой главе мы изучим базовые принципы работы с пакетом Macromedia Flash. А именно, узнаем такие вещи, которые обязательно пригодятся в дальнейшей работе. Мы перечислим их здесь, чтобы далее, при описании той или иной возможности или особенности Flash не описывать самые элементарные понятия. Мало того, что это непродуктивно, так еще и сильно раздражает по прошествии времени.

Что будет рассмотрено в этой главе?

Во-первых, самые основные принципы работы с пакетом. На какие этапы делится работа над изображением. Как оно создается. Как готовится к публикации. Почему все это делается так, а не иначе. Каких ошибок следует при этом избегать.

Во-вторых, внешний вид окна программы Flash. Как вы знаете, всякая программа, работающая в среде Windows, имеет окно, в котором, собственно, и протекает работа с документом. Мы узнаем, что находится в этом окне, и как всем этим пользоваться. Мы научимся работать с меню, панелями инструментов, панелями свойств и самим рабочим листом Flash. И даже создадим первое, самое примитивное изображение.

В-третьих, вспомогательные, но отнюдь не второстепенные возможности. Например, направляющие линейки, шкалы и координатная сетка, которые могут сильно помочь вам в работе. Мы узнаем, как ими пользоваться с максимальной для себя (простите за тавтологию) пользой.

Как видите, скучать нам не придется. В этой, самой первой, главе книги нам предстоит узнать много нового. Так давайте же не будем тратить время на пустые разговоры! Вперед!

Как работать в среде Flash

Сначала поговорим о самых основных принципах работы с Flash. На данном этапе Flash, да и сам компьютер нам не понадобятся. Вы можете закрыть

Flash и даже выключить компьютер, чтобы сэкономить электроэнергию. Просто поговорим, как работать в среде Flash.

Итак, вам нужно создать изображение. Статичное или анимированное (то есть, фильм), простое или интерактивное (фактически, программу) — неважно. Что делать?

Прежде всего, четко представлять, что вы хотите сделать. Это означает, что вам нужно предварительно спланировать свое пока еще не существующее изображение. Лучше всего, если вы нарисуете его на бумаге, хотя это необязательно. Важно просто хорошо представлять себе окончательный результат. Это нужно хотя бы для того, чтобы в процессе работы не забыть, что вы хотите сделать. Ведь если вы что-то забудете, переделывать уже сделанное будет гораздо труднее, чем делать это сразу, правильно.

Второй этап — собственно рисование. Если в наши планы входит создание анимации, то сначала подготовим неподвижную часть графики. Потом займемся созданием самой анимации. Повторяю, здесь мы создаем саму графику, не затрагивая интерактивные возможности, если они есть. Нарисуем все, что нужно, даже те элементы, которые необходимы только для реализации интерактивных возможностей. Это нужно, чтобы проверить саму композицию нашего изображения и при необходимости вовремя исправить ее.

Интерактивными возможностями займемся на третьем этапе. Именно здесь мы создадим все элементы, которые будут отвечать только за интерактивность. И, конечно, напишем сценарии на языке ActionScript, которые и будут реализовывать эту интерактивность. Здесь же мы выполним отладку и исправим ошибки в этих сценариях.

Самый последний, завершающий этап — это публикация готового изображения в один из форматов, пригодных для распространения. Что подразумевается под публикацией, стоит рассмотреть более подробно.

Когда вы работаете в среде Flash, вам нужно сохранять промежуточные и окончательные результаты работы. Для этого Flash, как и все программы, предоставляет возможность создания файлов своего собственного формата, называемого *документом Flash*. Это файл с расширением fla, довольно больших размеров, в котором хранится вся графика и все сценарии ActionScript, привязанные к этой графике. Кроме того, в данном файле хранится вся информация, необходимая для того, чтобы вы могли в любой момент исправить как графику, так и сценарии. Подобная информация зачастую весьма объемиста, поэтому файл документа Flash имеет такие большие размеры — до сотен килобайт.

А теперь давайте подумаем. Если мы собираемся распространять наше изображение для просмотра, но никак не для изменения, то зачем нам нужно хранить в файле всю эту дополнительную информацию? Когда мы ее исключим, то не только уменьшим размеры файла, но и защитим наше изо-

бражение от шаловливых ручонок тех, кто захочет немного изменить его и выдать за свое. Но как это сделать?

Вот поэтому Flash поддерживает два графических формата. Один из них служит для сохранения результатов работы в самой среде Flash — это формат документов Flash, записываемых в файлы с расширением fla. Второй формат служит только для распространения графики и включает лишь ту информацию, которая нужна для отображения этой графики и придания ей интерактивности. Эта информация сильно оптимизирована, чтобы уменьшить размер файла и ускорить его обработку. Такие файлы называются *распространяемыми файлами Shockwave/Flash*. Не перепутайте эти два формата данных — они совершенно разные и служат разным целям, хотя и тот, и другой предназначены для сохранения графики.

Вы спросите, что такое Shockwave. Это среда разработки мультимедийных приложений, разработанная фирмой Macromedia достаточно давно, гораздо раньше появления Flash. И Shockwave, и Flash используют один и тот же формат сохранения распространяемой графики. Только возможности, разумеется, у них совсем разные, поэтому файл Flash не может быть просмотрен средствами Shockwave.

Конечно, особенности вашего изображения могут внести поправки в эту схему. Если ваше изображение содержит только статичную графику, то вам не нужно создавать ни анимацию, ни сценарии ActionScript. Если вы создаете во Flash приложение, то основной упор, конечно же, сделайте на интерактивные возможности, а графика будет стоять на втором, а то и третьем месте. Приведенный выше порядок действий — просто основа, которой следует придерживаться, но отнюдь не догма, которую нужно возводить в абсолют.

Формально последним этапом создания Flash-изображения является его распространение. Но мы не будем рассматривать, как это осуществляется. В конце концов, это проблемы уже не Flash. Здесь мы опишем только, как и с помощью какого программного обеспечения можно просмотреть готовое изображение.

Просмотр графики Flash

Скажем сразу, что изображение Flash можно экспортировать в другом формате, например, Apple QuickTime, AVI или тех же GIF или JPEG. Правда, кое-какая информация при этом потеряется, в частности, сценарии ActionScript. Поэтому в другие, альтернативные Shockwave/Flash, форматы стоит экспортировать только ту графику, которая не имеет никаких интерактивных возможностей.

В отличие от изображений GIF и JPEG, поддерживаемых всеми существующими на данный момент программами Web-обозревателей, изображение Shockwave/Flash для просмотра требует специальной программы — *проигры-*

вателя Shockwave/Flash. Этот проигрыватель может быть встроен в Web-обозреватель в качестве модуля расширения или существовать в виде программы, запускаемой отдельно. В этом формат Shockwave/Flash схож с форматами Apple QuickTime и AVI, для которых также нужны отдельные проигрыватели, в первом случае — одноименный проигрыватель фирмы Apple, во втором — Универсальный проигрыватель фирмы Microsoft, поставляемый в составе Windows.

Очень странно, что формат Shockwave/Flash не поддерживается Web-обозревателями непосредственно. Ведь он существует уже достаточно давно — несколько лет — и уже стал в Сети стандартом де-факто. За это время производители программного обеспечения для Интернета могли бы и подустеиться. Пока же поддержкой формата Shockwave/Flash занимается только его разработчик — фирма Macromedia.

Итак, как же можно просмотреть изображение, сохраненное в формате Shockwave/Flash? Вы уже знаете ответ: с помощью проигрывателя Shockwave/Flash. Этот проигрыватель един во многих лицах. Давайте их все перечислим.

- ❑ Модуль расширения Web-обозревателя. Модули расширения поддерживают все современные программы Web-обозревателей, за исключением разве уж самых примитивных и маломощных. Две популярнейшие на данный момент программы — Microsoft Internet Explorer и Netscape Navigator — имеют этот модуль в составе своей поставки. Пользователям других программ Web-обозревателей следует загрузить и установить его самим. Это происходит автоматически, когда вы заходите на специальную страницу Web-сайта Macromedia. А перейти на эту страницу можно по особой ссылке, которая находится практически на любой Web-странице, использующей Flash-графику. Вы также можете установить этот модуль расширения вручную, запустив файл Install Flash Player 6.exe (модуль расширения) или Install Flash Player 6 AX.exe (компонент ActiveX). Оба этих файла находятся в подкаталоге Players/Release, находящемся в каталоге, где установлен сам Flash.
- ❑ Отдельная программа. Эта программа поставляется в составе Flash и при его установке записывается в подкаталог Players. Исполняемый файл этой программы называется SAFlashPlayer.exe.
- ❑ Полностью автономный исполняемый файл, содержащий и проигрыватель Shockwave/Flash, и созданный вами фильм Flash. Для просмотра такого файла не нужно иметь на своем компьютере ни модуля расширения Web-обозревателя, ни отдельного проигрывателя Shockwave/Flash. Вы просто запускаете этот исполняемый файл и смотрите изображение.

Как видите, имеются фактически три возможности распространения Flash-графики. Вы можете поместить свое изображение на Web-странице, тогда все желающие смогут просмотреть его, используя модуль расширения Web-

обозревателя. Вы можете распространять файл Shockwave/Flash другим способом, чтобы все желающие просмотрели его, воспользовавшись отдельной программой проигрывателя. И, наконец, если вы хотите продемонстрировать свое творчество тем, у кого нет ни отдельного проигрывателя Shockwave/Flash, ни соответствующего модуля расширения, вы можете создать автономный файл. Помните только, что в последнем случае размер вашего файла очень сильно возрастет (на целых 800 килобайт — это размер исполняемого файла проигрывателя Shockwave/Flash).

Но с проблемами распространения своего творчества вы столкнетесь еще не скоро. Пока что вы не знакомы с самой средой Flash. Познакомимся же с ней поближе.

Среда Flash

Здесь мы изучим саму среду Flash и все инструменты, которые она предоставляет в наше распоряжение.

Как запустить программу в среде Windows, вы знаете. Так запустите же Flash, если он еще не запущен.

Главное окно программы

Главное (или основное) окно программы Macromedia Flash показано на рис. 1.1. Рассмотрим его подробнее.



Рис. 1.1. Главное окно Flash

Скажем сразу, что Flash — *приложение с многодокументным интерфейсом* или просто многодокументное приложение. Это значит, что вы можете открыть в одном и том же рабочем окне программы сразу несколько документов. В этом случае окна, содержащие открытые документы, открываются внутри большого окна самой программы. К многодокументным приложениям также относятся Microsoft Word и Adobe Photoshop. Этим они отличаются от приложений с однодокументным интерфейсом (однодокументных приложений), в которых можно открыть только один документ, а чтобы открыть второй, нужно запускать вторую копию приложения. Примерами однодокументных приложений являются, в частности, текстовый редактор Microsoft WordPad и графический редактор Microsoft Paint, поставляемые в составе Windows.

Окно документа Flash занимает большую часть окна программы. Его заголовок совпадает с именем открытого в нем файла. Вы можете перемещать, свертывать и разворачивать это окно и изменять его размеры, в общем, продолжая с ним те же манипуляции, что и с любым другим окном Windows. Единственное исключение: вы не можете "вытащить" это окно за пределы окна программы (так называемого *родительского окна*).

При первом запуске Flash MX выводит на экран еще одно небольшое окно, показанное на рис. 1.2. Это так называемое *приглашение*, содержащее текст, предлагающий пользователю прочитать некоторые справочные данные или запустить интерактивные презентации, объясняющие, как работать во Flash. Вы можете просмотреть их или сразу же закрыть это окно. При последующих запусках окно-приглашение появляться больше не будет.

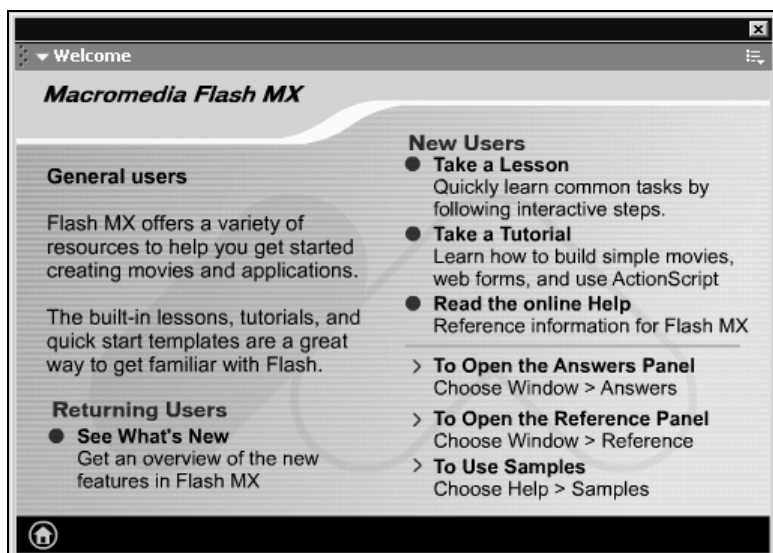


Рис. 1.2. Окно-приглашение

Левее окна документа находится *главный инструментарий* Flash или просто инструментарий. (Его также называют панелью инструментов, но мы не будем употреблять этот термин, чтобы избежать путаницы.) Эта небольшая серая панель, вытянутая по вертикали, содержит в себе набор кнопок. Нажимая кнопки, вы можете выбирать те или иные *инструменты*, предлагаемые Flash для рисования или правки графики.



Рис. 1.3. "Ручка" для "переноса"

Если вы "захватите" инструментарий мышью за особую "ручку", отображаемую в виде набора точек на темно-сером фоне в ее верхней части (рис. 1.3), то он "отклеится" от края родительского окна и превратится в независимое окно (рис. 1.4). Таким образом вы можете увеличить площадь окна документа Flash. Можно перемещать это окно по экрану так же, как любое другое окно. Есть даже возможность "вытащить" его за пределы окна программы, что недоступно для окон документов. Однако менять размеры окна, содержащего инструментарий, вы не можете — они всегда постоянны.

Если же вы не хотите, чтобы инструментарий маячил постоянно перед глазами, вновь "приклейте" его к краю окна. Для этого "поднесите" его мышью к левому или правому краю родительского окна и оставьте там. Отследить момент "приклеивания" и "отклеивания" очень просто: если при перетаскивании инструментарий меняет толстый контур на тонкий, то при отпуске кнопки мыши он будет "приклеен". И наоборот, если тонкий контур меняется на толстый, инструментарий будет "отклеен". Чтобы инструментарий ни в коем случае не "приклеивался" к краю окна, при его перетаскивании удерживайте нажатой клавишу <Ctrl>.

Если вы хорошенько присмотритесь к инструментарию, то увидите, что он разделен на четыре *области*. Перечислим их сверху вниз.

1. Область основных инструментов (заголовок **Tools**). Здесь находятся кнопки, предоставляющие доступ ко всем инструментам, что предусмотрены во Flash для рисования и правки уже нарисованного.



Рис. 1.4. Инструментарий, "отклеенный" от края родительского окна

2. Область вспомогательных инструментов (заголовок **View**). Здесь находятся всего две кнопки, которые мы рассмотрим в этой главе.
3. Область задания цвета (заголовок **Colors**). Здесь находятся элементы управления, позволяющие вам задавать цвет.
4. Область модификаторов (заголовок **Options**). Здесь находятся кнопки, предоставляющие доступ к *модификаторам* — дополнительным режимам, предусмотренным в том или ином выбранном в данный момент инструменте.

Не всегда в данный момент времени в инструментарии присутствуют все четыре области. Область модификаторов в некоторые моменты может быть пуста.

Правее окна документа, одна над другой, выстроились другие *панели*. ("Другие" — потому что инструментарий тоже относится к панелям. В дальнейшем, когда мы будем говорить о панелях, будут иметься в виду также и инструментарии.) Они предназначены для самых различных целей; подробнее мы рассмотрим их в следующих разделах книги. В верхней части каждой панели имеется ее *заголовок* — темно-серая полоса, на которой написано название панели.

Изначально все панели, имеющиеся на экране, "приклеены" к краю родительского окна программы, но не к левому, как инструментарий, а к правому. Разработчики из фирмы Macromedia считают, что так будет удобно большинству пользователей Flash, и, похоже, они правы. Однако вы можете придерживаться иного мнения. Поэтому и здесь существует возможность "отклеить" какую-либо панель от края родительского окна и поместить ее в независимое окно (рис. 1.5). Для этого каждая панель имеет "ручку" для ее "переноски" (см. рис. 1.3). Конечно же, вы всегда можете "приклеить" ее обратно, чтобы она не загромождала документ.

Панель может быть перемещена в любое место экрана, даже за пределы главного окна программы. Перетаскивать панель можно как за ее "ручку", так и за заголовок ее окна. Кроме того, вы можете изменять размеры окон панелей, что недоступно для инструментария.

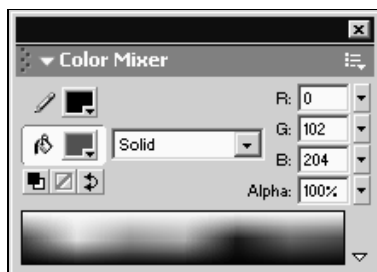


Рис. 1.5. Панель, "отклеенная" от края родительского окна

Внимание!

Если на вкладке **General** диалогового окна **Preferences** был включен флажок **Disable Panel Docking**, вы не сможете "приклеить" ни одну панель к краю окна документа.

Если вы перетащите одну панель на другую, эти панели будут объединены в общую *группу*, занимающую одно окно (рис. 1.6). Подобные группы можно также "приклеивать" к краю главного окна программы. Если вы посмотрите на рис. 1.1, то увидите, что все панели, "приклеенные" к краю главного окна, образуют такую группу. Чтобы вынести какую-либо панель из группы, "ухватите" ее за заголовок и "вытащите" прочь.

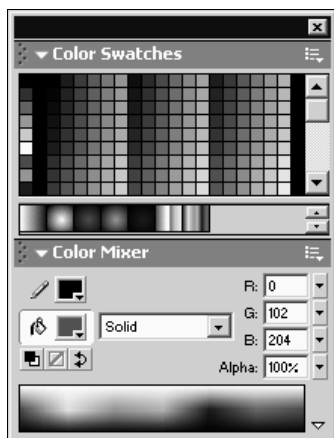


Рис. 1.6. Панели, объединенные в группу

Есть еще один способ уменьшить площадь, занимаемую панелью — "сжать" ее так, чтобы на экране остался только заголовок (рис. 1.7). Для этого дважды щелкните по заголовку окна панели. Чтобы "развернуть" панель до обычного состояния, снова дважды щелкните по заголовку ее окна.

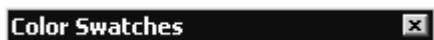


Рис. 1.7. "Сжатое" окно панели

Чтобы сжать панель, вы также можете один раз щелкнуть по ее заголовку (а не по заголовку ее окна). В этом случае панель примет такой вид, как на рис. 1.8. Конечно, в таком случае панель займет на экране больше места, зато этот прием работает не только в отдельном окне, но и когда панель "приклеена" к краю главного окна или объединена в группу с другими панелями. Посмотрите на рис. 1.1, и вы увидите, что некоторые панели в группе сжаты, а другие — развернуты. Таким образом, вы можете выборочно скрывать ненужные панели и раскрывать нужные.

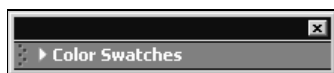


Рис. 1.8. "Сжатая" панель

Большинство панелей имеют так называемое *дополнительное меню*. Оно открывается при щелчке мышью по небольшой кнопке, расположенной в правом верхнем углу панели и имеющей изображение списка из трех позиций и небольшой стрелки, направленной вниз (рис. 1.9). В дополнительном меню находятся пункты, выполняющие редко используемые команды.

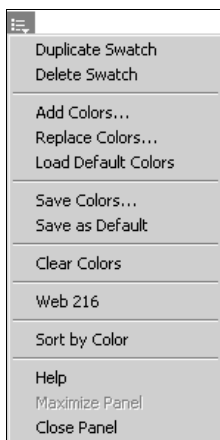


Рис. 1.9. Дополнительное меню панели (открыто)

Иногда бывает так, что панель имеет слишком маленький размер, чтобы в ней поместилось все ее содержимое. В этом случае в правом нижнем углу такой панели появляется небольшая кнопка, имеющая вид стрелки, направленной вниз (рис. 1.10). Это *кнопка раскрытия панели*, при нажатии на нее панель увеличивается в размерах так, чтобы все ее содержимое в ней поместилось. После этого стрелка, изображенная на кнопке раскрытия, переворачивается (указывает вверх), и при щелчке на нее панель уменьшится до первоначальных размеров.



Рис. 1.10. Стрелка раскрытия панели

Одна из панелей Flash стоит несколько особняком. Это так называемый *редактор свойств*, с помощью которого вы можете задавать различные параметры уже нарисованной графики, кадров фильма или всего документа Flash. Редактор свойств изначально "приклеен" к нижнему краю главного окна и показан на рис. 1.11. Подробнее мы опишем его далее.

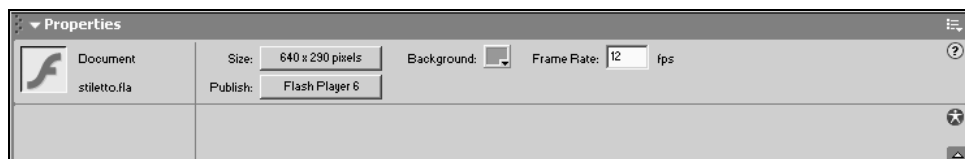


Рис. 1.11. Редактор свойств

И панели, и инструментарий всегда находятся выше окна документа, даже если в данный момент неактивны. Это сделано для того, чтобы вы могли в любой момент получить к ним доступ, вне зависимости от того, какое окно сейчас активно. Если же вы хотите убрать какую-либо из этих панелей, "вынесите" ее за пределы окна документа или вообще закройте, щелкнув мышью по *кнопке закрытия*, расположенной в правой части заголовка панели. В дальнейшем вы сможете открыть нужную панель, выбрав соответствующий пункт меню программы.

Кроме того, Flash может отображать в своем окне три *вспомогательных инструментария*. Эти инструментарии содержат кнопки, дублирующие наиболее часто используемые пункты меню программы: открытие файла, вырезание и копирование в буфер обмена Windows, вставка из буфера обмена и т. п. Как правило, Flash предоставляет средства для доступа к часто используемым командам меню из панелей и главного инструментария, а также с помощью комбинаций клавиш. Поэтому вспомогательные инструментарии изначально скрыты. Но, если хотите, вы можете вывести их на экран; как это сделать, описано ниже. Как и главный инструментарий, вспомогательные могут перемещаться по экрану и "приклеиваться" к краю окна: горизонтальные — к горизонтальному, вертикальные — к вертикальному.

Когда окно программы Flash перестает быть активным (например, когда пользователь переключается в другую программу), все панели и инструментарии временно скрываются, если они не "приклеены". При активизации окна программы они опять появляются на экране.

Управление окнами и панелями Flash

Как видите, Flash может вывести на экран сразу множество разнообразнейших окон. Как разобраться во всем этом многообразии?

Прежде всего, нужно знать пункты меню, с помощью которых осуществляется управление этими окнами. Все эти пункты находятся в подменю **Window**. Рассмотрим их подробнее.

Если вы открыли несколько документов в одной программе Flash, разобраться в них может быть очень трудно. Окна перекрывают друг друга, и добраться до нужного удастся далеко не сразу. Откройте подменю **Window** и посмотрите в самый его низ. Там будут находиться пункты, имеющие имена вида:

<Порядковый номер> <Имя файла документа Flash без расширения>

Для того чтобы переключиться в окно, где открыт нужный файл, просто выберите соответствующий пункт. Flash тотчас выведет это окно на первый план, т. е. активизирует его.

Если вам нужно держать на виду сразу два или больше окон, воспользуйтесь пунктами **Cascade** и **Tile** меню **Window**. Первый из них выкладывает все открытые окна документов в виде "стопки" в окне программы так, что вы можете видеть их заголовки и часть содержимого. Второй пункт "выкладывает" в окне программы "мозаику" из окон документов так, чтобы они не перекрывались.

Иногда нужно держать перед глазами несколько частей очень большого изображения. Для этого случая предусмотрен пункт **New Window** (здесь речь идет о подменю **Window**) и эквивалентная ему комбинация клавиш <Ctrl>+<Alt>+<N>. При этом Flash открывает еще одно окно, в котором показывает тот же файл, что был открыт в активном окне. Вы можете воспользоваться пунктом **Tile**, чтобы держать оба этих окна перед глазами.

Пункт **Tools** служит для вывода на экран или скрытия главного инструментария. Если слева от имени этого пункта стоит галочка, это значит, что инструментарий выведен на экран (или, как еще говорят, что соответствующий пункт меню "включен"). Чтобы убрать его, снова выберите этот пункт; инструментарий исчезнет вместе с галочкой. Если впоследствии вы еще раз выберете этот пункт, инструментарий и галочка снова появятся. Такие пункты меню, меняющие свое состояние на противоположное при выборе, называют *выключателями*. Вместо выбора этого пункта вы можете нажать "горячую" комбинацию клавиш <Ctrl>+<F2>.

Большую часть меню **Window** занимает набор аналогичных пунктов-выключателей, служащих для вывода на экран или скрытия различных панелей. Далее в книге мы рассмотрим эти пункты вместе с рассмотрением той или иной панели.

Если вам для какой-то цели нужно закрыть все панели и инструментарии, например, чтобы просмотреть без помехи все изображение, выберите пункт **Close All Panels**. Все панели будут тут же закрыты. Но имейте в виду, что открывать вам их придется вручную.

Если же вам нужно не закрыть безвозвратно, а просто скрыть на время все панели и инструментарии, выберите пункт **Hide Panels** в меню **View**. Этот пункт работает как выключатель, т. е. при первом выборе он скрывает все панели, а при втором — снова выводит их на экран. Вы также можете нажать клавиши <Tab> или <F4> — это проще и быстрее, чем лезть в меню.

Есть еще один весьма удобный способ организовать свое рабочее место: пользовательские *раскладки панелей*. В двух словах это можно объяснить следующим образом. Вы открываете нужные вам в данный момент панели, располагаете на экране так, как вам удобно, и сохраняете их расположение в настройках Flash. После этого вы можете закрыть их совсем или перемес-

тить на другие места. Но чтобы вернуться к прежнему состоянию, вам достаточно будет выбрать сохраненную ранее раскладку, и все панели выстроятся на экране так, как они были выстроены при сохранении выбранной раскладки. Таких сохраненных раскладок может быть сколько угодно.

Чтобы сохранить раскладку панелей, выберите пункт **Save Panel Layout**. На экране появится диалоговое окно **Save Panel Layout** (рис. 1.12). Введите в поле ввода **Name** имя сохраняемой раскладки и нажмите кнопку **OK**. Если вы передумали сохранять раскладку, нажмите кнопку **Cancel**.

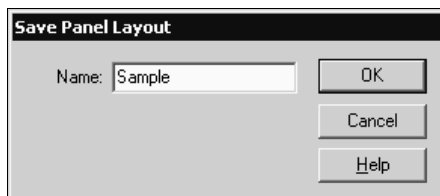


Рис. 1.12. Диалоговое окно **Save Panel Layout**

Чтобы выбрать сохраненную раскладку, откройте подменю **Panel Sets** меню **Window** и выберите в нем пункт, соответствующий нужной раскладке. Пункт **Default Layout** позволит вам выбрать раскладку по умолчанию, эта раскладка создается при первом запуске программы.

Чтобы удалить ненужную или ошибочно сохраненную раскладку, вам придется воспользоваться Проводником Windows или аналогичной программой. Если вы работаете в Windows 95/98/Me, то откройте подкаталог Application Data/Macromedia/Flash MX/Configuration/Panel Sets, расположенный в каталоге Windows. Если же вы работаете в Windows NT/2000/XP, то откройте тот же подкаталог Application Data/Macromedia/Flash MX/Configuration/Panel Sets, расположенный в каталоге вашего пользовательского профиля. Отыщите файл, одноименный с нужной раскладкой, и удалите его. Когда вы снова откроете подменю **Panel Sets** меню **Window**, удаленной раскладки в нем уже не будет.

С Flash MX уже поставляется несколько готовых раскладок панелей, которые могут вам пригодиться. Все они перечислены в табл. 1.1 и также доступны в подменю **Panel Sets** меню **Window**.

Таблица 1.1. Готовые раскладки панелей, поставляемые в составе Flash MX

Название раскладки	Разрешение экрана, на которое рассчитана раскладка	Для кого предназначена
Designer [1024x768]	1024×768	Для художников

Таблица 1.1 (окончание)

Название раскладки	Разрешение экрана, на которое рассчитана раскладка	Для кого предназначена
Designer [1280x1024]	1280×1024	Для художников
Designer [1600x1200]	1600×1200	
Developer [1024x768]	1024×768	Для разработчиков интернет-приложений
Developer [1280x1024]	1280×1024	
Developer [1600x1200]	1600×1200	

Вы уже знаете, что Flash может отображать на экране вспомогательные инструменты. Чтобы вывести их на экран, выберите соответствующий пункт-выключатель подменю **Toolbars**. Ниже мы опишем все вспомогательные инструменты, доступные во Flash.

- ❑ *Вспомогательный инструментальный общего назначения.* За этим длинным названием прячется набор кнопок, предоставляющих доступ к наиболее часто используемым командам меню. Это команды открытия файла, его печати, предварительного просмотра, работы с буфером обмена Windows и некоторые другие. На наш взгляд, проще получить доступ ко многим из этих команд или из панелей, или с помощью комбинаций "горячих клавиш". Выводится этот инструментальный путем выбора пункта-выключателя **Main**.
- ❑ *Строка статуса*, принадлежащая родительскому окну программы. В ней выводится краткая подсказка, разъясняющая функции кнопки или пункта меню, над которым находится курсор мыши. Также в ней отображается состояние клавиш <Caps Lock> и <NumLock> клавиатуры. Вероятно, на первых этапах освоения Flash эту строку стоит вывести на экран, что выполняется выбором пункта-выключателя **Status**.
- ❑ *Инструментальный управления проигрыванием* анимации. Содержит кнопки запуска, приостановки и остановки проигрывания, а также некоторые другие. Самый, на наш взгляд, полезный вспомогательный инструментальный из всех трех. За него "отвечает" пункт-выключатель **Controller**.

Если вы щелкнете правой кнопкой мыши по заголовку любой панели (но не заголовку ее окна), на экране появится контекстное меню этой панели. Пользуясь этим меню, вы можете выполнить над данной панелью различные манипуляции:

- ❑ закрыть панель, выбрав пункт **Close Panel**;
- ❑ увеличить размеры панели так, чтобы ее содержимое поместилось в нее полностью, выбрав пункт **Maximize Panel**;
- ❑ получить справку по этой панели, выбрав пункт **Help**.

Работа с окном документа

Теперь рассмотрим работу в окне документа — самом главном окне Flash. Ведь, если подумать, все эти инструментарии и панели нужны только для обслуживания окна документа, точнее, самого документа, открытого в этом окне. Можно сказать, что окно документа — сердце Flash.

Для управления окном документа Flash предоставляет достаточно много инструментов. Как "раскладывать" эти окна в главном окне программы, вы уже знаете. Теперь мы научимся управлять представлением документа в окне, т. е. займемся самим окном.

Но сначала посмотрим, из каких частей состоит окно документа.

Окно документа

Окно документа Flash показано на рис. 1.13. Рассмотрим его подробнее.

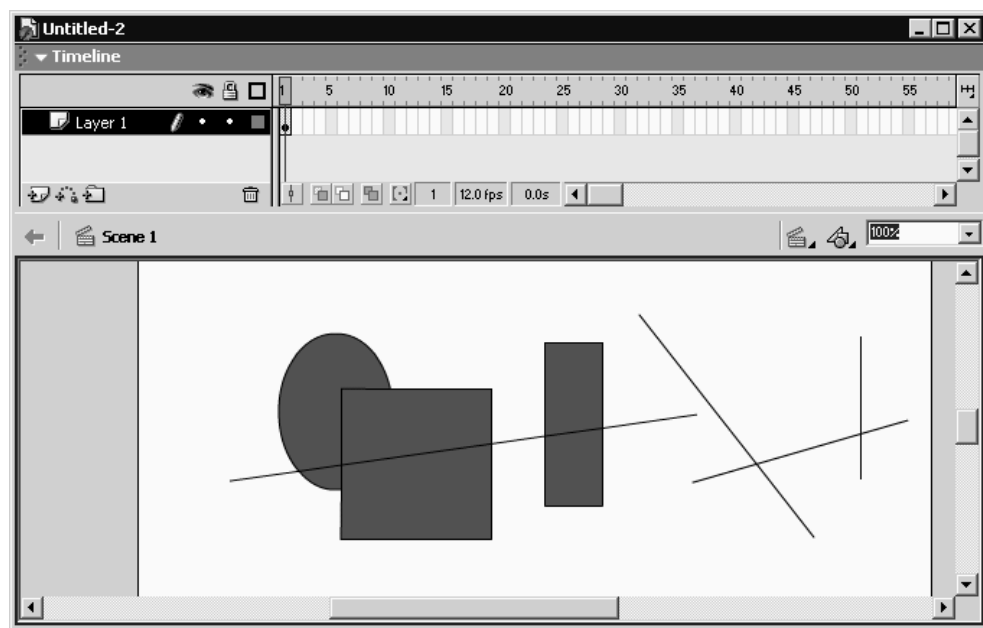


Рис. 1.13. Окно документа Flash

Как видите, окно документа Flash разделено на две неравные части. Разделены они тонкой серой линией, которую вы можете перетаскивать мышью, меняя размеры этих частей. Ниже этой серой линии, практически сливаясь с ней, находится также небольшая панель с кнопками, которую невозможно скрыть. Эта панель принадлежит окну документа, а не главному окну программы — запомните это.

Теперь обратите внимание на нижнюю, большую часть окна. В ней на сером фоне находится белый прямоугольник, ограниченный тонкими черными линиями. Небольшое изображение, нарисованное нами для примера, находится как раз внутри этого прямоугольника. Этот прямоугольник схематически представляет наш рисунок, и размеры его совпадают с размерами этого изображения, заданного при его создании. (О создании изображения и задании его размеров см. главу 2.) То, что не входит в пределы этого прямоугольника, не попадает в наше изображение и отбрасывается Flash при сохранении в формате Shockwave/Flash. Назовем этот прямоугольник *рабочим листом* или просто *листом*, а всю серую область, где рисуется графика, — *рабочей областью*.

Как вы уже поняли, изображение рисуется в рабочей области. При этом все, что не попадает на рабочий лист (то есть, находится на окружающем его сером поле), не войдет в окончательное изображение, сохраненное в файле Shockwave/Flash. Однако в файле документа Flash сохраняется все: и попадающее на рабочий лист, и находящееся на сером поле. Этим можно пользоваться, размещая некоторые части изображения на сером поле и перетаскивая их на лист, когда в них появится нужда.

В дальнейшем, говоря о рабочем листе или просто о листе, мы фактически будем подразумевать рабочую область. Если нам нужно будет однозначно отделить рабочий лист от рабочей области, мы специально предупредим вас.

У правого края панели с кнопками, расположенной выше рабочей области, находится небольшой раскрывающийся список, позволяющий задать масштаб отображения содержимого рабочего листа (рис. 1.14). Этот список содержит ряд пунктов, позволяющих задать тот или иной масштаб; подробно описывать их нет нужды. Есть только два пункта, которые нужно описать. Пункт **Show Frame** показывает в окне документа весь рабочий лист, автоматически выбирая нужный масштаб. А пункт **Show All** показывает в окне документа только нарисованное на рабочем листе изображение.

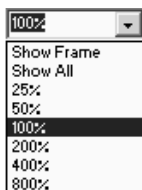


Рис. 1.14. Раскрывающийся список выбора масштаба

Обратимся теперь к верхней части окна документа. Вы видите, что в ней находится некий гибрид списка и шкалы. Это *временная шкала* или *шкала анимации*, служащая для создания анимации и проградуированная в кадрах. (Также временная шкала содержит список слоев, созданных в документе.)

Вы можете "отклеить" ее и "вытянуть", поместив в отдельное окно, аналогично панелям. Подробнее о слоях и анимации вы можете прочитать в соответствующих главах *части 3* этой книги.

Внимание!

Если на закладке **General** диалогового окна **Preferences** был включен флажок **Disable Timeline Docking**, вы не сможете "приклеить" временную шкалу к краю окна документа.

Принципы работы с графикой Flash

И теперь самое время дать некоторые базовые понятия о создании графики в среде Flash. Они очень помогут вам в дальнейшем.

Любое графическое изображение Flash состоит из небольших простейших фрагментов, так называемых *графических примитивов*. Это могут быть отрезки линий, геометрические фигуры или текстовые надписи. Вы создаете эти примитивы с помощью соответствующих инструментов Flash, кнопки для выбора этих инструментов находятся в области **Tools** инструментария. Как только вы нажмете нужную кнопку, она останется нажатой — это значит, что данный инструмент выбран.

Кнопки инструментария имеют "говорящие" картинки, позволяющие точно узнать, какой инструмент скрывается за той или иной кнопкой. В частности, инструмент "прямоугольник" скрывается за кнопкой с изображением прямоугольника, инструмент "эллипс" — за кнопкой с изображением эллипса и т. п. Если же вы не уверены, что за данной кнопкой скрывается тот инструмент, который вам нужен, задержите над ней курсор мыши. При этом над кнопкой появится всплывающая подсказка, кратко описывающая этот инструмент. Если вы вывели на экран строку статуса, принадлежащую окну программы, в ней появится развернутое описание инструмента.

Многие простейшие примитивы рисуются следующим образом. Вы помещаете курсор мыши в некоторую точку рабочего листа, нажимаете левую кнопку, протаскиваете мышь и отпускаете кнопку. Так, в частности, рисуются прямые линии, прямоугольники и эллипсы. Вы можете попробовать сами что-либо нарисовать. Например, попробуйте повторить изображение, нарисованное в окне документа, такое как на рис. 1.13.

Щелкая мышью по различным частям изображения, вы можете выделять их и впоследствии выполнять над ними какие-либо действия. Например, пользуясь редактором свойств, задавать для выбранного фрагмента изображения какие-либо параметры, перемещать их или выполнять некоторые преобразования.

Во всех подробностях работа с неподвижной, статичной графикой в среде Flash описана в *части 2* этой книги. Здесь мы дадим только самые базовые понятия, чтобы ввести вас в курс дела.

Теперь условимся о специальных терминах, которые будем применять в дальнейшем в этой книге. Если вы запомните их сразу, вам будет проще усваивать дальнейший материал.

Итак, вы уже знаете, что такое графический примитив. Это простейшая часть графического изображения Flash: линия, прямоугольник, эллипс, фрагмент текста и др. Из примитивов состоят более крупные части изображения: графические фрагменты и элементы. *Графическим фрагментом* будем называть более или менее сложный фрагмент изображения, но, тем не менее, не являющийся независимой ее частью. А *графическим элементом* — независимую часть изображения, тем не менее, входящую в его состав.

О создании анимации и интерактивных возможностей будет во всех подробностях рассказано в *частях 3 и 4*.

Управление окном документа

А пока что расскажем, как можно управлять видом изображения в рабочей области.

Прежде всего, скажем, что вы можете отключить отображение в окне документа временной шкалы. Когда вы будете рисовать неподвижные изображения, она будет только мешать вам. Чтобы избавиться от временной шкалы, отключите пункт-выключатель **Timeline** в меню **Views** (или нажмите комбинацию клавиш <Ctrl>+<Alt>+<T>). Впоследствии вы всегда можете его включить.

Если вы хотите скрыть всю графику, что выходит за пределы рабочего листа и оказывается на сером поле, выключите пункт-выключатель **Work Area** меню **View** или нажмите комбинацию клавиш <Ctrl>+<Shift>+<W>. Вся эта графика так и останется на своих местах, просто она не будет отображаться.

Вы уже знаете, как увеличить или уменьшить масштаб изображения, воспользовавшись раскрывающимся списком масштабов, показанным на рис. 1.14. Однако есть еще несколько способов управления масштабом, предоставляемых меню программы и инструментарием. В последнем случае вы можете воспользоваться инструментом "лупа".

Чтобы выбрать инструмент "лупа", щелкните кнопку, показанную на рис. 1.15, в области **View** инструментария или нажмите клавишу <M> или <Z> на клавиатуре. Курсор мыши примет вид небольшой лупы.



Рис. 1.15. Кнопка выбора инструмента "лупа"

Для увеличения масштаба изображения на рабочем листе просто щелкните по этому изображению. Если же вы хотите уменьшить масштаб, щелкните

по изображению, удерживая нажатой клавишу <Alt>. Когда вы увеличиваете изображение, внутри небольшой лупы, форму которой примет курсор мыши, отображается небольшой знак "плюс", когда уменьшаете — знак "минус".

Чтобы менять режимы увеличения и уменьшения, вы также можете воспользоваться кнопками модификаторов "увеличение" и "уменьшение", показанными на рис. 1.16. Это кнопки с зависимой фиксацией или кнопки-переключатели: выбор одной кнопки отключает другую.



Рис. 1.16. Кнопки модификаторов "увеличение" (слева; нажата) и "уменьшение" (справа)

Если вы хотите выбрать какой-либо фрагмент изображения, чтобы рассмотреть его поближе, сделайте следующее. Поместите курсор мыши в какую-либо точку на листе Flash, нажмите левую кнопку и, не отпуская ее, протаскивайте мышью так, чтобы захватить в прямоугольник выделения нужный вам фрагмент графики. После этого Flash автоматически рассчитает требуемый масштаб и так позиционирует рабочий лист в окне, чтобы показать вам выбранный фрагмент.

Для увеличения и уменьшения масштаба вы также можете воспользоваться пунктами **Zoom In** и **Zoom Out** меню **View** (или комбинациями клавиш <Ctrl>+<=> и <Ctrl>+<->). При этом увеличение и уменьшение масштаба производится ступенчато, по пунктам списка масштабов (см. рис. 1.14).

Еще одна возможность изменения масштаба изображения — подменю **Magnification** меню **View**. Его пункты задают различные масштабы изображения, аналогично неоднократно упомянутому списку масштабов. Также это подменю содержит пункты **Show Frame** и **Show All**. Комбинация клавиш <Ctrl>+<1> быстро устанавливает масштаб 1:1 (100%), <Ctrl>+<2> показывает весь рабочий лист целиком (аналогично пункту **Show Frame**), а <Ctrl>+<3> — только нарисованное на нем изображение (как пункт **Show All**).

Если в окне документа помещается не весь рабочий лист, а только небольшой его фрагмент, вы можете воспользоваться инструментом "рука", чтобы перемещать лист для просмотра всей графики. Инструмент "рука" включается кнопкой, показанной на рис. 1.17, в области **View** инструментария или клавишей <H>. После его включения курсор мыши принимает вид руки.



Рис. 1.17. Кнопка выбора инструмента "рука"

Чтобы переместить лист, "захватите" его мышью и перетащите на другое место. Как видите, все это очень просто.

Если вы хотите временно переключиться на другой инструмент, когда включена "рука", нажмите на клавиатуре клавишу пробела и, удерживая ее нажатой, щелкните нужную кнопку инструментария. Выбранный вами инструмент будет активным, пока вы не отпустите клавишу пробела, как только вы это сделаете, Flash автоматически выберет инструмент "рука".

Вы также можете задать качество отображения графики в окне документа. По умолчанию она выводится с самым высоким качеством. Однако, если у вас маломощный компьютер, не справляющийся с выводом высококачественной графики, или вы создаете очень сложные изображения, вы можете уменьшить качество вывода графики. Для этого служит набор из четырех пунктов-переключателей меню **View**. При выборе каждого из них левее его названия появляется большая черная точка, говорящая о том, что он включен, бывший включенным до этого пункт отключается. Перечислим по порядку все эти пункты и опишем режимы вывода изображения, за которые они "отвечают", в порядке улучшения качества графики.

- ❑ Режим вывода контуров. Включается выбором пункта **Outlines** или комбинацией клавиш <Ctrl>+<Shift>+<Alt>+<O>. Все графические примитивы выводятся в виде тонких контуров без всякого сглаживания и без заливки. Это может быть полезно, если вы хотите изменить только контуры графики.
- ❑ "Быстрый" режим. Включается выбором пункта **Fast** или комбинацией клавиш <Ctrl>+<Shift>+<Alt>+<F>. Если выбран этот режим, Flash выводит графику полностью, но не выполняет сглаживание контуров. Это значит, что линии могут выглядеть очень грубыми, зернистыми. Если у вас маломощный компьютер, выводящий на экран одновременно максимум 256 цветов, используйте этот режим.
- ❑ Режим сглаживания графики. Включается выбором пункта **Antialias** или комбинацией клавиш <Ctrl>+<Shift>+<Alt>+<A>. Если выбран этот режим, Flash выполняет сглаживание контуров графики (но не текста). Качество вывода графики очень высоко, но символы текста получаются грубыми, зернистыми. Используйте этот режим, если видеоподсистема вашего компьютера может работать в режимах HiColor и TrueColor, в противном случае переключитесь на "быстрый" режим.
- ❑ Режим сглаживания и графики, и текста. Включается выбором пункта **Antialias Text** или комбинацией клавиш <Ctrl>+<Shift>+<Alt>+<T>. Включен по умолчанию. Качество вывода графики и текста очень высокое. Всегда используйте этот режим, если имеете достаточно мощный компьютер.

Имейте в виду, что заданное вами качество изображения затрагивает только его отображение в окне документа Flash. При его публикации в формат

Shockwave/Flash вы сможете задать качество результирующего изображения в одном из поддерживаемых Flash графических форматов. Само изображение при смене качества его вывода не меняется, и качество его не искажается.

Средства позиционирования

Средства позиционирования помогут вам точно разместить графические фрагменты на листе. Давайте их рассмотрим.

Самое простое из этих средств — *координатные линейки*. Они отображаются вдоль верхней и левой сторон окна документа Flash, позволяя точно определить соответственно горизонтальную (X) и вертикальную (Y) координаты. Начало координат находится в верхнем левом углу рабочего листа. По умолчанию линейки проградуированы в пикселах, но вы можете задать другую единицу измерения; как это сделать, описано в *главе 2*.

Чтобы вывести на экран координатные линейки, выберите пункт-выключатель **Rulers** в меню **View** или, если у вас очень гибкие пальцы, нажмите комбинацию клавиш <Ctrl>+<Alt>+<Shift>+<R>. Результат этих действий показан на рис. 1.18.

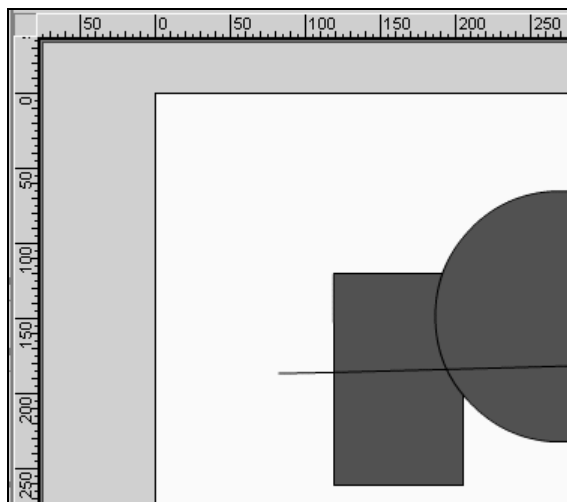


Рис. 1.18. Координатные линейки

Если вы вывели на экран линейки, то можете воспользоваться такой полезной возможностью как *направляющие*. Это тонкие линии, которые удобно использовать, например, для точного выравнивания графики на листе или для создания и соблюдения границ вокруг изображения. Эти линии не сохраняются в результирующем изображении Shockwave/Flash, так что вы можете использовать их без боязни что-то испортить.

Чтобы создать направляющую, вам нужно, прежде всего, вывести на экран координатные линейки. Как это сделать, вы уже знаете. Затем поместите курсор мыши на горизонтальную или вертикальную линейку, в зависимости от того, горизонтальную или вертикальную направляющую вы хотите создать. Далее нажмите левую кнопку мыши и, не отпуская ее, "вытащите" направляющую на рабочий лист, после чего отпустите кнопку. Созданная вами направляющая показана на рис. 1.19, она имеет вид тонкой линии.

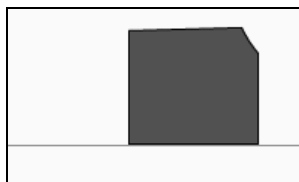


Рис. 1.19. Направляющая (тонкая линия внизу)

Вы можете перемещать направляющие мышью и удалять их. Чтобы удалить направляющую, перетащите ее обратно на линейку. Если вы хотите сделать направляющие перемещаемыми, можно их на время заблокировать. Для этого выберите пункт-выключатель **Lock Guides** в подменю **Guides** меню **View** или нажмите комбинацию клавиш <Ctrl>+<Alt>+<;>. Существует также возможность временно скрыть все направляющие, отключив пункт-выключатель **Show Guides** в подменю **Guides** меню **View** или нажав комбинацию клавиш <Ctrl>+<;>.

Когда вы рисуете или правите графику, графические фрагменты будут "приклеиваться" к направляющим, если их "поднести" слишком близко. Это предусмотрено для вашего удобства, но иногда, когда нужно особенно точно позиционировать графику, мешает. Чтобы временно отключить "приклеивание", отключите пункт-выключатель **Snap to Guides** в подменю **Guides** меню **View** или нажмите комбинацию клавиш <Ctrl>+<Shift>+<;>.

Выбор пункта **Edit Guides** подменю **Guides** меню **View** или комбинация клавиш <Ctrl>+<Shift>+<Alt>+<G> выводят на экране диалоговое окно **Guides**. С помощью этого диалогового окна вы можете настраивать некоторые параметры направляющих и выполнять над ними различные действия. Диалоговое окно **Guides** показано на рис. 1.20.

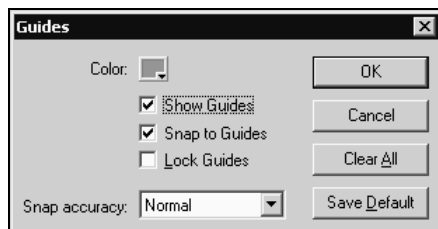


Рис. 1.20. Диалоговое окно **Guides**

В верхней части этого диалогового окна находится небольшой цветной квадратик. Этот квадратик называется *селектором цвета*. С его помощью вы можете задать цвет направляющих линий.

Чтобы выбрать цвет с помощью селектора цвета, нужно щелкнуть по нему мышью. После этого на экране появится небольшое окошко, содержащее набор цветов, доступных для выбора (рис. 1.21). Щелкните мышью по нужному цвету — и окошко селектора закроется. Вы также можете закрыть его, снова щелкнув по цветному квадрату или нажав клавишу <Esc>.

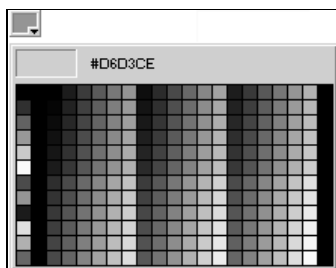


Рис. 1.21. Открытый селектор цвета

Флажок **Show Guides** включает или выключает вывод направляющих на экран. По своей функции он аналогичен пункту-выключателю **Show Guides** в подменю **Guides** меню **View**.

Флажок **Snap to Guides** включает или выключает "приклеивание" графики к направляющим. Он аналогичен пункту-выключателю **Snap to Guides** в подменю **Guides** меню **View**.

Флажок **Lock Guides** включает или выключает блокировку направляющих, чтобы их нельзя было ни переместить, ни удалить. Его "обязанности" такие же, как и у пункта-выключателя **Lock Guides** в подменю **Guides** меню **View**.

Раскрывающийся список **Snap accuracy** позволяет задать, как близко графический фрагмент должен быть помещен к направляющей, чтобы быть к ней "приклеенным". В этом списке доступны три пункта: **Must be close** (должен быть близко), **Normal** (значение по умолчанию) и **Can be distant** (может быть достаточно далеко). Поэкспериментируйте с этими установками, чтобы подобрать самую, на ваш взгляд, удачную. (Хотя, часто лучше оставить значение по умолчанию.)

Кнопка **Clear All** позволяет вам удалить все направляющие разом.

Задав нужные параметры, нажмите кнопку **OK**, чтобы применить их. Если вы передумали, нажмите кнопку **Cancel**. Если же вы хотите, чтобы заданные вами параметры использовались и в дальнейшем, когда вы станете создавать направляющие в других изображениях, т. е. стали значениями по умолчанию, нажмите кнопку **Save Default**.

Еще одно весьма полезное средство позиционирования, предлагаемое Flash, — это *координатная сетка*. Вы можете рассматривать ее как набор направляющих, выводимых самой программой через равные промежутки. Использовать координатную сетку можно как вместе с координатными линиями, так и отдельно от них.

Чтобы вывести на экран координатную сетку, выберите пункт-выключатель **Show Grid** в подменю **Grid** меню **View** или нажмите комбинацию клавиш <Ctrl>+<’>. Сама координатная сетка показана на рис. 1.22, она имеет вид множества тонких линий, наложенных на рабочий лист.

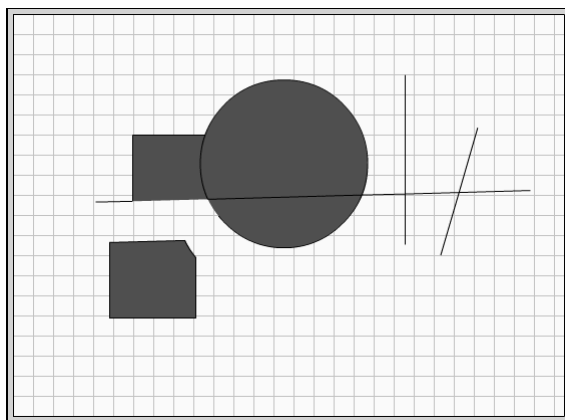


Рис. 1.22. Координатная сетка

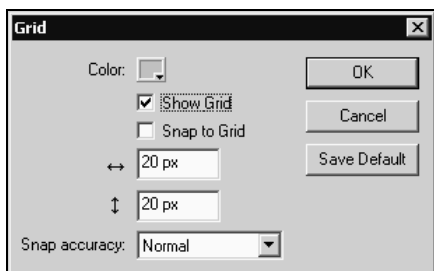
Когда вы рисуете или правите графику, графические фрагменты будут "приклеиваться" к линиям координатной сетки, так же, как и к направляющим. Чтобы временно отключить "приклеивание", отключите пункт-выключатель **Snap to Grid** в подменю **Grid** меню **View** или нажмите комбинацию клавиш <Ctrl>+<Shift>+<’>.

Выбор пункта **Edit Grid** подменю **Grid** меню **View** или комбинация клавиш <Ctrl>+<Alt>+<G> выводят на экране диалоговое окно **Grid**. С помощью этого диалогового окна вы можете настраивать некоторые параметры координатной сетки. Диалоговое окно **Grid** показано на рис. 1.23.

В верхней части окна находится уже знакомый вам селектор цвета **Color**. С его помощью вы можете задать цвет линий координатной сетки.

Флажок **Show Grid** включает или выключает вывод на экран координатной сетки. По своей функции он аналогичен пункту-выключателю **Show Grid** в подменю **Grid** меню **View**.

Флажок **Snap to Grid** включает или выключает "приклеивание" графики к линиям координатной сетки. Он аналогичен пункту-выключателю **Snap to Grid** в подменю **Grid** меню **View**.

Рис. 1.23. Диалоговое окно **Grid**

Следующие два поля ввода служат для установки горизонтального и вертикального шага линий координатной сетки. Эти значения задаются в той единице измерения, которая была выбрана при настройке параметров документа Flash. (По умолчанию, эта единица измерения — пиксели.)

Раскрывающийся список **Snap accuracy** позволяет задать, как близко графический фрагмент должен быть помещен к линии координатной сетки, чтобы быть к ней "приклеенным". В этом списке доступны четыре пункта: **Must be close** (должен быть близко), **Normal** (значение по умолчанию), **Can be distant** (может быть достаточно далеко) и **Always snap** ("приклеивается" независимо от расстояния). Поэкспериментируйте с этими установками, чтобы подобрать самую подходящую, или, если хотите, оставьте значение по умолчанию.

Выбрав нужные параметры, нажмите кнопку **OK**, чтобы применить их. Если вы передумали, нажмите кнопку **Cancel**. Если же вы хотите, чтобы заданные вами параметры использовались и в дальнейшем, когда вы станете работать с координатной сеткой в других изображениях, т. е. стали значениями по умолчанию, нажмите кнопку **Save Default**.

Вот и все о средствах позиционирования. Как и об интерфейсе пользователя пакета Flash MX.

Глава 2



Типовые функции Flash

Разобравшись с пользовательским интерфейсом Flash и предоставляемыми им возможностями, перейдем к работе с файлами. В этой главе мы расскажем о файловых операциях Flash, точнее, об их особенностях по сравнению с другими Windows-приложениями, создающими файлы документов.

Когда вы набираете текст в любимом Microsoft Word, вы сохраняете его на жестком диске в виде файла. Файл — это массив информации, записанный на дисковом устройстве и имеющий уникальное имя, по которому его можно однозначно распознать. Кроме имени, файл имеет также набор атрибутов: признак "только для чтения", даты создания и последнего изменения, комментариев и т. п. Но, как правило, опознается файл по имени.

Документы Microsoft Word хранятся в файлах. Документы Macromedia Flash также хранятся в файлах, как и готовые к распространению изображения формата Shockwave/Flash. Сама программа Flash хранится в виде огромного многомегабайтного файла. А уж сколько файлов занимает интерактивное руководство по Flash — страшно представить!

Любое приложение, создающее какие-либо документы, должно сохранять их в файлах. А для этого оно должно поддерживать так называемые файловые операции: создание, открытие, закрытие, сохранение, сохранение под другим именем (пересохранение). Таким образом, файловые операции относятся к типовым операциям, которые должны поддерживаться всеми подобными программами. Также к типовым операциям относятся печать документа и предварительный просмотр его перед печатью.

В приложениях, написанных для Windows, выполнение типовых операций стандартизировано. Windows предоставляет стандартные диалоговые окна открытия, сохранения файлов, печати и настройки принтера. (Исключение составляют разве только ученические и совсем уж хитроумные программы.) Обычные диалоговые окна вам, конечно, знакомы, поэтому мы не будем их описывать. Сосредоточимся только на особенностях, присущих именно Flash.

Однако современные Windows-приложения не только работают с файлами. Они всегда готовы прийти на помощь неопытному пользователю. А именно, предоставляют удобную в использовании электронную справочную систему, включающую в себя полное интерактивное руководство по программе, описывающее практически все ее возможности. Только самые примитивные или, как иногда говорят, "интуитивно понятные" из программ обходятся без справки, да и то, если у разработчиков нет времени или желания ее писать.

В процессе работы с Flash вам тоже может понадобиться — и наверняка понадобится! — помощь. Как и все серьезные Windows-приложения, Flash снабжен мощной справочной системой. И мы также расскажем вам, как ей пользоваться.

Пожалуй, читая одну только справочную систему, можно изучить возможности всей программы. Правда, такой подход не даст никакого практического опыта, но все же имеет право на жизнь. Некоторые народные умельцы-самоучки так и поступают: прилежно изучают теорию по электронным справочным руководствам, а потом очень долго не решаются приступить к практической работе...

Файловые операции

Когда вы хотите создать новый документ, то даете приложению команду на создание нового файла. Когда вы хотите вернуться к уже сохраненному документу, указываете приложению открыть файл, в котором сохранен этот документ. Как видите, файловые операции очень важны. Именно поэтому мы рассмотрим их в первую очередь.

Создание нового документа

Прежде, чем начать работать с документом, его нужно создать. Давайте же выясним, как создать новый документ (точнее, новый файл документа) в среде Flash.

Создать сам документ очень просто. Для этого выберите в меню **File** пункт **New** или нажмите комбинацию клавиш <Ctrl>+<N>. Внутри окна программы Flash откроется новое окно документа с пустым рабочим листом и пустой временной линией. На этом этапе никаких новых диалоговых окон, запрашивающих дополнительную информацию, на экране не появляется.

Собственно, когда вы запускаете программу Flash щелчком по ее ярлыку на Рабочем столе, в меню **Start (Пуск)** или на самом исполняемом файле, новый документ создается автоматически сразу после ее запуска. Прибегать к пункту **New** меню **File** вам нужно только тогда, когда вы хотите создать еще один новый документ в процессе работы с программой.

Выше уже говорилось, что при создании нового документа Flash вы не сможете задать его параметры, например, размеры. Чтобы сделать это, вам придется выбрать пункт **Document** в меню **Modify**, выбрать пункт **Document Properties** в контекстном меню листа или нажать комбинацию клавиш <Ctrl>+<J>. На экране появится диалоговое окно **Document Properties** (рис. 2.1).

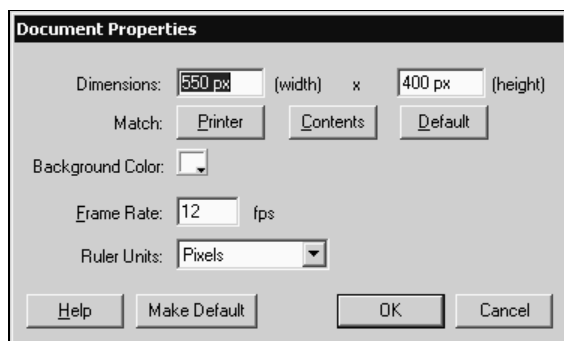


Рис. 2.1. Диалоговое окно **Document Properties**

Поля ввода **Width** и **Height**, находящиеся в группе **Dimensions**, служат для указания соответственно ширины и высоты изображения. Они задаются в текущих единицах измерения, по умолчанию это пиксели. Размеры изображения по умолчанию — 550×400, минимальный размер — 1×1, максимальный — 2880×2880 пикселей.

Ниже находится группа кнопок **Match**, позволяющая быстро задать размеры фильма. Кнопка **Printer** делает размер изображения равным текущему размеру бумаги, заданному в настройках текущего принтера. Кнопка **Contents** позволит вам задать такой размер изображения, чтобы вся нарисованная графика помещалась в нем, оставляя минимум пустого места. Кнопка **Default** задает размер изображения по умолчанию.

Селектор цвета **Background Color** позволяет вам задать цвет фона изображения. По умолчанию он белый. Рекомендуется его таким и оставить, так как цветные фоны смотрятся очень плохо. В конце концов, человечество уже много лет пишет черными чернилами по белой бумаге.

В поле ввода **Frame Rate** задается частота кадров создаваемого фильма. (Конечно, этот параметр имеет значение только в том случае, если вы создаете анимированное изображение, то есть, фильм.) Для фильмов, публикуемых в Интернете, обычно задается частота от 8 до 12 кадров в секунду. Значение по умолчанию — 12 кадров в секунду; рекомендуется его оставить, если только вы не хотите сделать свой фильм компактнее в ущерб качеству.

Раскрывающийся список **Ruler Units** позволяет задать текущую единицу измерения. Эта единица будет потом использоваться везде, где вам будет нуж-

но задать какие-либо размеры или расстояния, например, при настройке шага линий координатной сетки. Этот список предоставляет для выбора шесть пунктов:

- ☐ **Inches** — дюймы;
- ☐ **Inches (decimal)** — десятичные дюймы;
- ☐ **Points** — пункты;
- ☐ **Centimeters** — сантиметры;
- ☐ **Millimeters** — миллиметры;
- ☐ **Pixels** — пиксели (значение по умолчанию).

Задав параметры фильма, нажмите кнопку **ОК**, чтобы применить их. Если вы передумали, нажмите кнопку **Cancel**. Если же вы хотите, чтобы заданные вами параметры использовались по умолчанию при создании новых документов Flash, нажмите кнопку **Save Default**.

Задать параметры изображения вы также можете, используя редактор свойств. Щелкните по пустому пространству рабочего листа — и редактор свойств примет вид, показанный на рис. 2.2. При нажатии кнопки **Size** на экране появится уже знакомое вам диалоговое окно **Document Properties** (см. рис. 2.1), где вы сможете задать параметры изображения. Установленные вами размеры изображения написаны на самой кнопке **Size**, так что вы всегда сможете их узнать. Селектор цвета **Background** и поле ввода **Frame Rate** выполняют те же функции, что и их "коллеги" из окна **Document Properties**.

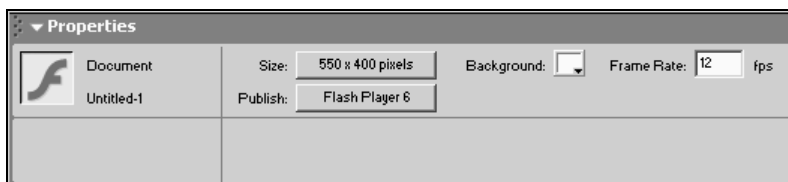


Рис. 2.2. Редактор свойств в режиме задания параметров изображения

Создание нового документа на основе шаблона

Наконец-то, после многих лет ориентации на профессионалов и "профессионалов" интернет-графики, фирма Macromedia обратила внимание на начинающих и неопытных пользователей. Теперь пакет Flash позволяет создавать новый документ на основе некой заготовки, созданной опытными дизайнерами по заказу Macromedia, — *шаблона*. Такой шаблон уже имеет в своем составе некоторые фрагменты изображений, вам останется только добавить свою графику и текст, а также задать ряд параметров.

Создать новый документ на основе шаблона так же просто, как и "пустой" документ. Для этого выберите пункт **New From Template** в меню **File**. На экране появится диалоговое окно **New Document**, показанное на рис. 2.3. В этом окне вы сможете выбрать нужный шаблон.

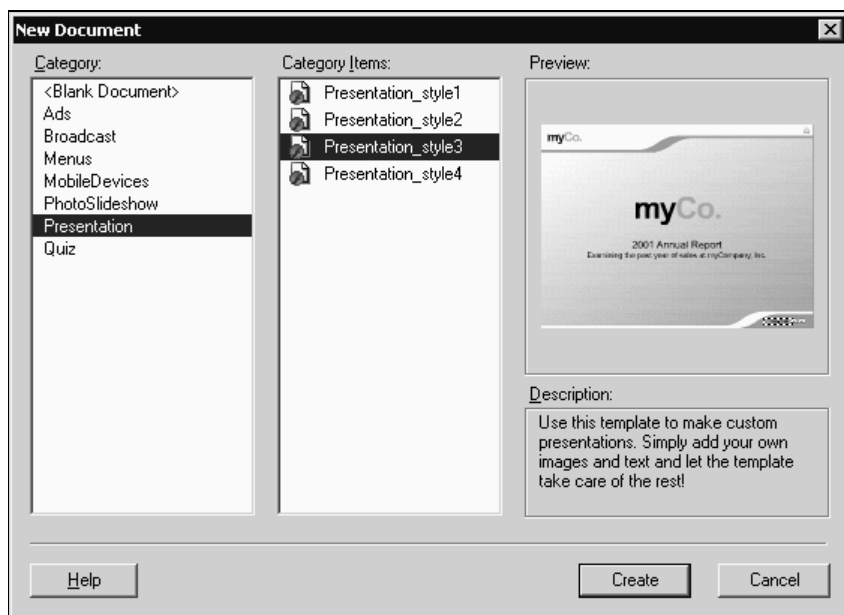


Рис. 2.3. Диалоговое окно **New Document**

Все шаблоны Flash организованы в категории. Всего таких категорий семь. Им соответствуют следующие пункты списка **Category**:

- ☐ **Ads** — рекламные баннеры различных размеров;
- ☐ **Broadcast** — фильмы, предназначенные для вещания в Интернете;
- ☐ **Menus** — два шаблона, позволяющие создать соответственно строку меню и набор закладок, аналогичные используемым в обычных Windows-приложениях;
- ☐ **MobileDevices** — изображения и фильмы, предназначенные для просмотра на карманных компьютерах Nokia Communicator и PocketPC;
- ☐ **PhotoSlideshow** — фотогалерея, в которой одно изображение плавно сменяется другим (слайд-шоу);
- ☐ **Presentation** — презентации;
- ☐ **Quiz** — анкеты, опросные листы и викторины.

После того, как вы выберете нужную категорию из списка **Category**, в списке **Category Items** появится перечень собственно шаблонов, относящихся к

этой категории. Например, для категории **Ads** появится список различных форматов баннеров, а для категории **Presentation** — форматов презентаций. Выберите нужный шаблон, после чего в панели просмотра **Preview** появится изображение этого шаблона, а в текстовом поле **Description** — его краткое описание.

Чтобы создать новый документ на основе выбранного шаблона, нажмите кнопку **Create**. Чтобы отказаться от создания нового документа, нажмите кнопку **Cancel**.

Работа с документами

Теперь рассмотрим остальные файловые операции, предусматриваемые Flash.

Открыть файл документа можно, выбрав пункт **Open** в меню **File** или нажав комбинацию клавиш <Ctrl>+<O>. После этого на экране появится стандартное диалоговое окно открытия файла **Windows**, в котором вы можете выбрать нужный файл.

Сохранение документа в файл выполняется после выбора пункта **Save** меню **File** или нажатия комбинации клавиш <Ctrl>+<S>. Если сохранение выполняется первый раз, т. е. документ еще не был сохранен в файл, то на экране появится стандартное диалоговое окно сохранения файла **Windows**, где вы должны будете задать имя файла. Впоследствии Flash будет просто сохранять документ, не спрашивая имени файла.

Flash MX также предоставляет вам возможность сохранить документ в формате Flash 5 (предыдущей версии Flash). Для этого просто выберите в списке типов файла стандартного диалогового окна сохранения пункт **Flash 5 Document**.

Если вам нужно пересохранить документ в другом файле под другим именем, выберите пункт **Save As** в меню **File** или нажмите комбинацию клавиш <Ctrl>+<Shift>+<S>. На экране появится стандартное диалоговое окно сохранения файла **Windows**, где вы должны будете задать имя нового файла.

Если вы после множества изменений хотите загрузить последнюю сохраненную версию документа, выберите пункт **Revert** в меню **File**. Учтите только, что этот пункт недоступен, если вы ни разу не сохранили текущий документ или ни разу не изменяли его после загрузки.

Чтобы закрыть открытый файл, активизируйте окно документа, в котором он открыт, и выберите пункт **Close** меню **File** или нажмите комбинацию клавиш <Ctrl>+<W>. Хотя проще всего будет закрыть это окно (имеется в виду окно документа, а не окно программы).

Кроме того, Flash предоставляет возможность отправки открытого в его среде документа по электронной почте. Для этого выберите пункт **Send** в меню **File**. На экране появится окно программы почтового клиента, установлен-

ного в системе в качестве клиента по умолчанию. В этом окне будет сформировано готовое письмо с вложенным в него файлом, содержащим отправляемый документ. Вам останется только вписать адрес, тему и, возможно, текст письма и запустить отправку почты.

Создание нового шаблона

Шаблоны, конечно, неплохая идея. Но она была бы еще лучше, если бы набор шаблонов был расширяемым. Вероятно, такими соображениями руководствовались разработчики Flash MX, когда добавляли в этот программный продукт возможность сохранения документа в виде шаблона. В самом деле, это позволяет вам самим создавать собственные шаблоны для своих нужд.

Чтобы создать свой шаблон на основе активного документа, выберите в меню **File** пункт **Save As Template**. На экране появится диалоговое окно **Save As Template** (рис. 2.4).

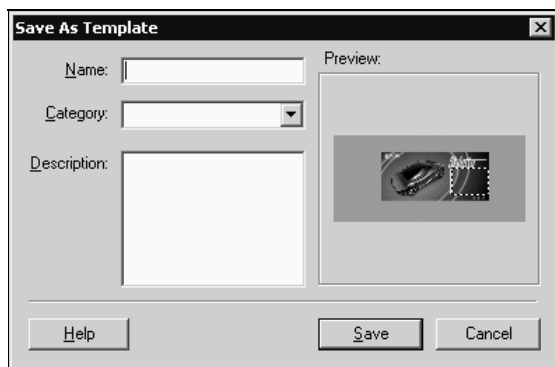


Рис. 2.4. Диалоговое окно **Save As Template**

В поле ввода **Name** задается имя шаблона, отображаемое в списке **Category Items** диалогового окна **New Document** (см. рис. 2.3). В раскрывающемся списке **Category** выбирается или вводится вручную название категории, где будет находиться создаваемый шаблон. В области редактирования **Description** вводится текст описания шаблона. После этого останется только нажать кнопку **Save** для сохранения шаблона или **Cancel** для отказа от этого.

Созданные пользователем шаблоны сохраняются в виде файлов с расширением fla в одном из подкаталогов, соответствующем выбранной в окне **Save As Template** категории и расположенном в подкаталоге Application Data/Macromedia/Flash MX/Configuration/Templates. Если вы работаете в Windows 95/98/Me, то сможете найти подкаталог Application Data/Macromedia/Flash MX/Configuration/Templates в каталоге Windows. Если же вы работаете в Windows NT/2000/XP, то найдете его в каталоге вашего пользовательского профиля. Таким образом, чтобы удалить ненужный шаблон, просто удалите соответствующий ему файл.

Печать

Печать документов, созданных в среде Flash, несколько отличается от печати документов в других программах. Сейчас мы рассмотрим эти отличия. И заодно перечислим все операции, связанные с печатью документов.

Собственно печать документа Flash осуществляется обычным для Windows-приложения способом. Вы выбираете пункт **Print** в меню **File** или нажимаете комбинацию клавиш <Ctrl>+<P>. После этого на экране появляется стандартное диалоговое окно печати Windows, в котором вы можете выбрать принтер, задать количество копий и выбрать страницы, которые должны быть напечатаны. Flash для печати использует стандартное окно без всяких изменений.

Как и многие Windows-приложения, работающие с документами, Flash предоставляет возможность просмотра документа в том виде, в котором он будет распечатан. Для этого выберите пункт **Print Preview** в меню **File**. На экране появится окно предварительного просмотра, где будет показан документ (рис. 2.5).

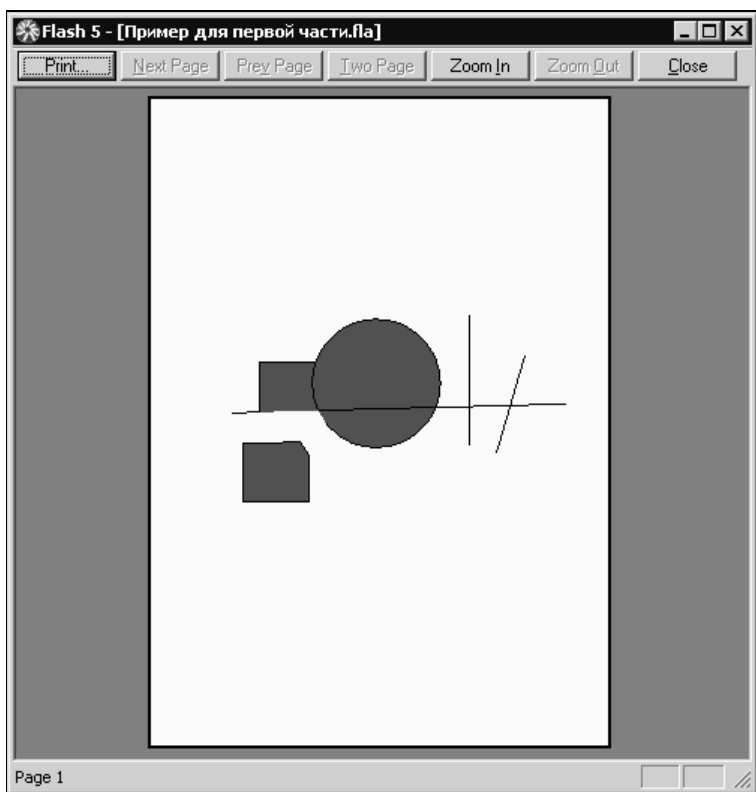


Рис. 2.5. Окно предварительного просмотра документа перед печатью

В верхней части этого окна находится набор кнопок, с помощью которых над показанным в нем документом выполняются различные манипуляции. Всего этих кнопок семь:

- ☐ **Print** — запуск печати документа;
- ☐ **Next Page** — переход на следующую страницу многостраничного документа, если она есть;
- ☐ **Prev Page** — переход на предыдущую страницу многостраничного документа, если она есть;
- ☐ **One Page** или **Two Page** — одна и та же кнопка, переключающая режимы отображения одной или двух страниц документа одновременно;
- ☐ **Zoom In** — увеличение изображения;
- ☐ **Zoom Out** — уменьшение изображения;
- ☐ **Close** — закрытие окна предварительного просмотра.

Опять же, как и многие программы, предусматривающие печать своих документов, Flash предоставляет возможность задать параметры печати. Эти параметры включают задание размера бумаги, отступов, масштаба и центрирования изображения. Чтобы получить доступ к этим настройкам, выберите пункт **Page Setup** в меню **File**. На экране появится диалоговое окно **Page Setup** (рис. 2.6).

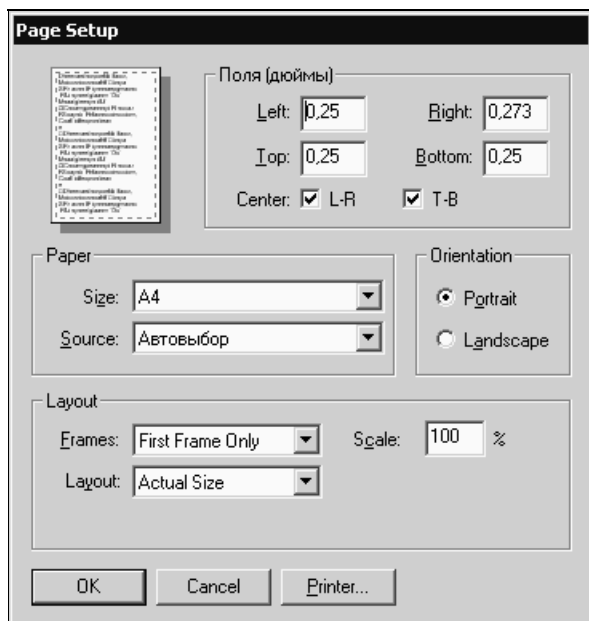


Рис. 2.6. Диалоговое окно **Page Setup**

В группе **Paper** находятся элементы управления для задания размера бумаги и способа ее подачи. Размер бумаги задается с помощью раскрывающегося списка **Size**, а способ подачи — **Source**.

Набор переключателей **Orientation** служит для выбора расположения листа бумаги. Переключатель **Portrait** включает портретное (вертикальное) расположение, а переключатель **Landscape** — ландшафтное (горизонтальное).

В группе **Layout** находятся элементы управления для задания расположения печатаемого изображения на листе. Рассмотрим их подробнее.

Раскрывающийся список **Frames** задает, какой кадр фильма будет напечатан. Если выбран пункт **First Frame Only**, будет напечатан только первый кадр. Если же выбрать пункт **All Frames**, будут напечатаны все кадры. Конечно, эта настройка имеет смысл только для фильма, но никак не для статичного изображения. (О создании анимации см. *часть 3*.)

С помощью раскрывающегося списка **Layout** задается расположение кадров фильма на листе бумаги. Этот список имеет пять пунктов:

- ☐ **Actual Size** — на одну страницу выводится один кадр фильма;
- ☐ **Fit On One Page** — на одну страницу выводится один кадр фильма, причем выполняется автоматическое масштабирование, чтобы он заполнил страницу целиком;
- ☐ **Storyboard — Boxes** — на одну страницу рядами выводятся несколько кадров фильма, причем каждый кадр располагается в рамке (рис. 2.7), так называемая многокадровая печать;
- ☐ **Storyboard — Grid** — на одну страницу рядами выводятся несколько кадров фильма, причем разделяются они друг от друга линиями;
- ☐ **Storyboard — Blank** — на одну страницу рядами выводятся несколько кадров фильма без рамок и разделяющих линий.

Если выбран пункт **Actual Size**, становится доступным поле ввода **Scale**. В этом поле задается масштаб изображения (кадра), выводимого на страницу. Это может помочь, если вы хотите распечатать очень мелкое или, наоборот, очень крупное изображение.

Если выбран один из пунктов **Storyboard — Boxes**, **Storyboard — Grid** или **Storyboard — Blank**, становятся доступными флажок **Label frames** и поля ввода **Frames across** и **Frame margin**. Если включить флажок **Label frames**, то под каждым кадром на странице Flash будет подставляться его метка. В поле ввода **Frames across** введите количество кадров, располагаемых в один горизонтальный ряд; косвенно этот параметр влияет на количество кадров, помещаемых на одну страницу, и, соответственно, на масштаб кадров. А в поле ввода **Frame margin** указывается расстояние между отдельными кадрами, располагаемыми на странице, это расстояние задается в текущей единице измерения.

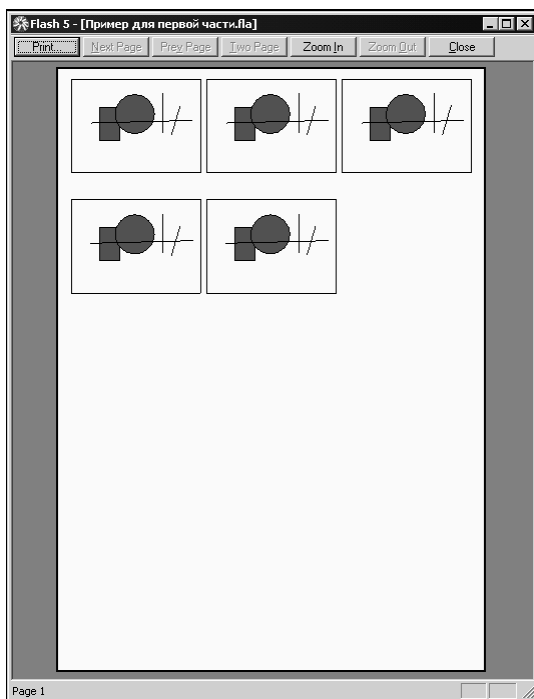


Рис. 2.7. Фильм в окне предварительного просмотра (включена многокадровая печать)

Интерактивная справка Flash

Для вызова интерактивной справки вам нужно просто нажать клавишу <F1> или выбрать пункт **Using Flash** в меню **Help**. После этого на экране появится окно встроенной справочной системы, показанное на рис. 2.8. Как видите, справочная система Flash использует для работы Web-обозреватель, установленный в системе по умолчанию.

В левой части окна справки расположен древовидный список статей. Вы можете щелкнуть мышью по названию статьи, чтобы просмотреть ее содержание, развернуть или свернуть ветвь "дерева" статей. Содержание выбранной статьи отображается справа.

Вы можете искать определения различных терминов, встретившихся в руководстве. Выполняется такой поиск следующим образом. Над списком тем находится небольшое меню из трех пунктов. Пункт **Contents** выводит собственный список тем. Пункт **Index** отображает набор букв алфавита (латинского, естественно). Выберите нужную букву, и на экране появится список терминов, начинающихся на эту букву. Вам останется только щелкнуть нужный термин, чтобы просмотреть его разъяснение.

Также предусмотрен поиск статей по ключевым словам. Для этого щелкните мышью по пункту **Search** списка тем, и на экране появится окно **Search** (рис. 2.9). Введите в поле ввода появившегося на экране окна нужное ключевое слово и нажмите кнопку **List Topics**. После этого в списке, занимающем нижнюю половину окна поиска, появятся заголовки найденных статей. Вам остается только выбрать нужную и нажать кнопку **Display**. Выбранная статья отобразится в главном окне справки. Чтобы закрыть окно **Search**, щелкните кнопку **Cancel**.

Выбор пункта **Using Flash** в меню **Help** выводит собственно руководство по Flash. Кроме того, в меню **Help** есть еще шесть аналогичных пунктов:

- ☐ **ActionScript Dictionary** — выводит на экран справочник по языку ActionScript;
- ☐ **Welcome** — выводит на экран окно приглашения (см. рис. 1.2);
- ☐ **What's New** — выводит на экран окно с описанием новых возможностей, появившихся во Flash MX;
- ☐ **Lessons** — выводит на экран окно со списком интерактивных уроков Flash MX;

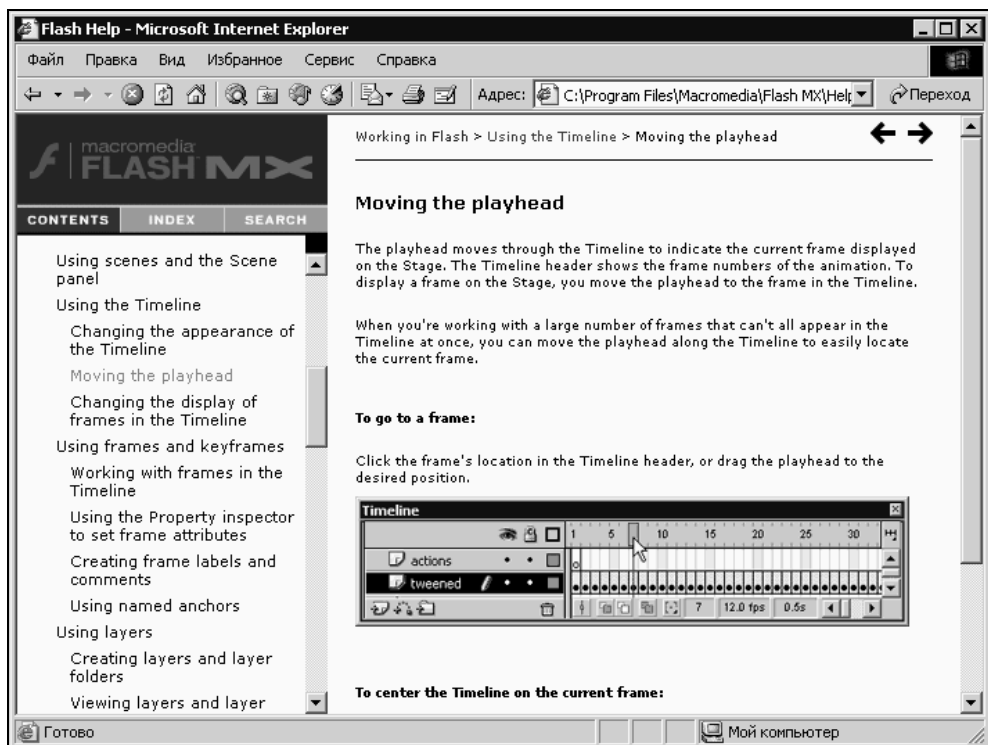


Рис. 2.8. Справочная система Flash

- ❑ **Tutorials** — выводит на экран окно со списком руководств для начинающих пользователей, рассказывающих о создании графики во Flash и программировании сценариев на языке ActionScript;
- ❑ **Samples** — выводит Web-страницу со списком примеров, поставляемых с Flash.

Очень и очень многие диалоговые окна Flash имеют кнопку **Help**. Эта кнопка вызывает на экран интерактивную справку и открывает ее на той статье, где описывается данное окно. Также некоторые панели имеют в своем составе кнопку вызова справки (рис. 2.10), расположенную в правом верхнем углу. При нажатии на нее также открывается соответствующая статья справки.

Фирма Macromedia включила во Flash MX доступ к очень интересному интернет-сервису, предназначенному для пользователей этого программного пакета. Это панель **Answers**, в которой отображаются различные документы, — новости, советы, инструкции — загружаемые с Web-сайта Macromedia. Эта панель показана на рис. 2.11, изначально она всегда присутствует на экране. Конечно, вы можете скрыть ее, а затем снова вызвать, выбрав пункт-выключатель **Answers** в меню **Window** или нажав комбинацию клавиш <Alt>+<F1>.

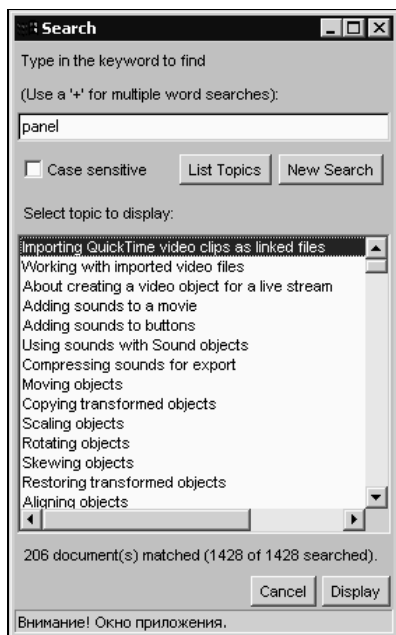


Рис. 2.9. Окно **Search**



Рис. 2.10. Кнопка вызова справки, расположенная на панели

Рис. 2.11. Панель **Answers**

Как вы поняли, панель **Answers** работает по принципу Web-обозревателя, т. е. загружает и отображает специально подготовленные Web-страницы. Чтобы загрузить самую свежую Web-страницу, нажмите кнопку **Update** в нижней части этой панели.

Меню **Help** с помощью пункта **Flash Support Center** предоставляет доступ еще к одному сервису, предназначенному для пользователей пакета. Это центр поддержки Flash, где можно получить ответы на вопросы, связанные с функционированием Flash.

Ну и еще немного об интерактивной справке Flash. Выбрав пункт-выключатель **Reference** в меню **Window** или нажав комбинацию клавиш <Shift>+<F1>, вы сможете вызвать на экран панель **Reference** (рис. 2.12). Данная панель представляет исчерпывающий справочник по языку ActionScript и будет очень полезна для программистов.

Как видите, панель **Reference** состоит из иерархического списка тем и области, где отображается текст выбранной темы. Вы можете перетаскивать мышью серую линию, разделяющую список и область просмотра текста, меняя их относительные размеры. А если вы щелкнете мышью небольшую кнопку, находящуюся на этой линии, то иерархический список тем вообще исчезнет, и область просмотра развернется на всю панель. Чтобы вернуть список, еще раз щелкните эту кнопку.

Вы можете выделять текст в области просмотра и копировать его в буфер обмена Windows. Для этого выберите пункт **Copy** дополнительного меню или нажмите комбинацию клавиш <Ctrl>+<C>. А пункт **Print** позволит вам распечатать текст выбранной темы.

С помощью набора пунктов-переключателей, находящихся в верхней части дополнительного меню, вы можете менять шрифт, которым набран текст выбранной темы. Доступны три пункта: **Large Font** (крупный шрифт), **Mediun Font** (выбран по умолчанию) и **Small Font** (мелкий шрифт).

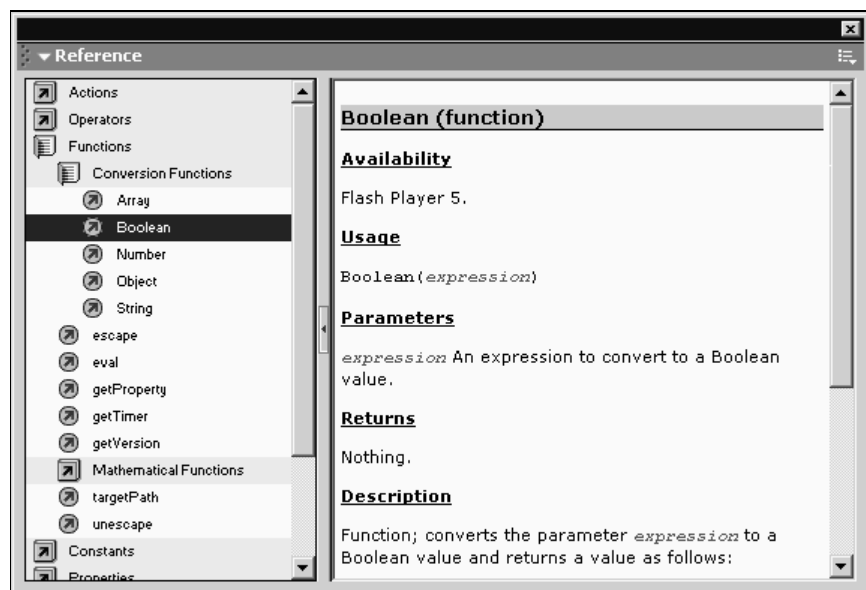
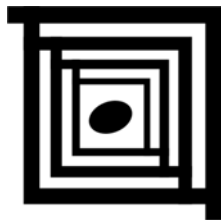


Рис. 2.12. Панель Reference

Глава 3



Настройка Flash

Разработчики практически каждого программного продукта стремятся сделать свое детище популярным, распространить его на максимальное количество пользователей. Любой программист, даже если он написал всего одну примитивнейшую бесплатную утилиту, в своих самых сладких снах видит себя королем программного обеспечения, а свое детище — установленным на каждом компьютере мира. И это нормально. Тщеславие свойственно человеческой природе так же, как и любопытство.

И, наряду с любопытством, оно правит этим миром. Более того, оно движет его вперед, объединяет людей, делает их единым целым — человечеством. В самом деле, представьте на минутку, что было бы, если каждый станет творить только для себя любимого, забыв об остальном человечестве? Мир бы раскололся на множество индивидуальных мирков; в чем-то эти мирки, несомненно, будут богаче большого мира, но в остальном — несравнимо беднее. "Настоящий творец творит только для себя", — сказал какой-то высоколбый мизантроп, но заприте этого умника в уютный карцер, изолируйте его от остального мира и посмотрите — много ли он натворит.

В этой главе мы поговорим о тщеславии, человеколюбии и пользовательских настройках Flash. Как это связано между собой, мы сейчас объясним.

Итак, если программист не относится к числу "настоящих творцов", он стремится сделать свой программный продукт доступным максимальному количеству пользователей. То есть, как можно более популярным. Но, поскольку люди и их привычки очень разные, возникает проблема: как угодить всем, как сделать, чтобы всем было удобно? Существует два пути решения этой проблемы.

Путь первый: переписывание программы под каждого конкретного пользователя (как правило, за отдельную плату). Такого подхода придерживаются разработчики бухгалтерских, складских, учетных, расчетных и прочих "деловых" программ. Хорошо известно, что бухгалтерия и складской учет в разных странах существенно отличаются, поэтому каждую такую программу приходится иной раз писать заново для каждого конкретного заказчика. А поскольку стандартизацией учета и контроля здесь и не пахнет, разработ-

чики деловых программ всегда будут иметь свой кусок хлеба (и свою головную боль).

Путь второй: создание гибких, настраиваемых программ. Такой подход выгоден для универсальных программ, которыми пользуются все: текстовых редакторов, обработчиков электронных таблиц, систем управления базами данных (СУБД), графических программ, системных утилит, проигрывателей видеофильмов и т. п. Разработчик оснащает свою программу возможностями ее настройки пользователем, эти возможности могут быть более или менее широкими, в зависимости от программы. В самом деле, не будет же он переписывать весь текстовый редактор полностью, чтобы изменить набор кнопок на инструментальной панели, да еще требовать за это деньги! (Мягко говоря, его не поймут.)

Macromedia Flash относится как раз ко второй категории программ, имеющих возможности настройки пользователем. Эти возможности довольно широки, мы рассмотрим их в отдельной, третьей главе книги, в ее первом разделе. И рассмотрим очень подробно.

Вы уже знаете, что многим пунктам меню Flash присвоены клавишные комбинации. Таким образом, вы можете вызвать команду, просто нажав эту комбинацию, а не обращаясь к меню — так гораздо быстрее. Но Flash позволяет вам также перенастроить эти комбинации клавиш, если стандартные вас почему-то не удовлетворяют. Настройке клавиатурных комбинаций посвящен второй раздел этой главы.

Настройка программы

Все пользовательские настройки Flash осуществляются в диалоговом окне настроек **Preferences**. Чтобы вызвать его на экран, выберите пункт **Preferences** в меню **Edit** (или нажмите комбинацию клавиш <Ctrl>+<U>). Само это диалоговое окно, точнее, вкладка **General**, где настраиваются основные параметры программы, показано на рис. 3.1.

В поле ввода **Undo Levels** задается количество операций пользователя, сведения о которых Flash хранит в памяти. Это нужно для выполнения операции так называемого *отката*, т. е. отмены результата последней операции пользователя. (Подробнее об откате см. главу 5.) В большинстве случаев лучше оставить значение по умолчанию (100). Если вы хотите сэкономить оперативную память компьютера для других нужд, можете уменьшить это значение вплоть до нуля. Если же, наоборот, вы считаете, что не уверены в себе, то можете увеличить его до 200.

Включение флажка **Disable PostScript** позволит вам отключить PostScript-вывод при печати на принтере, поддерживающем PostScript. Это может замедлить печать, так что включайте этот флажок только при наличии проблем с выводом изображения на принтер.

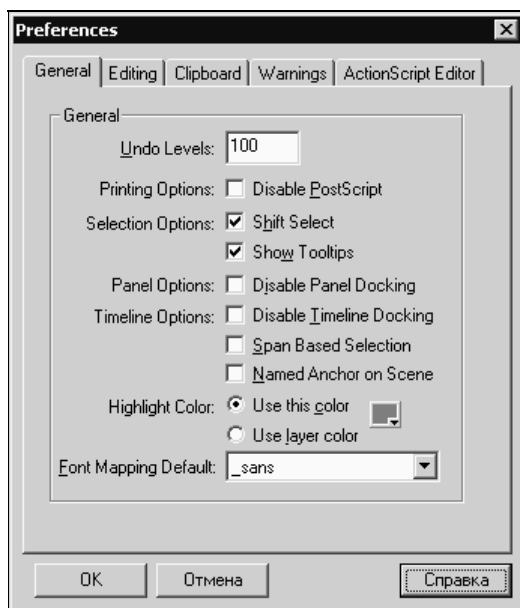


Рис. 3.1. Диалоговое окно **Preferences** (вкладка **General**)

Если флажок **Shift Select** включен (а он включен по умолчанию), то для выделения нескольких графических фрагментов на листе вам придется удерживать нажатой клавишу <Shift>. Если же он выключен, вам не нужно удерживать клавишу <Shift> — Flash будет выделять фрагменты при последовательных щелчках мышью. Включать или не включать этот флажок — на наш взгляд, дело вкуса.

Флажок **Show Tooltips** включает или отключает вывод всплывающих подсказок при наведении курсора мыши на кнопки и другие элементы управления панелей и инструментариев. Если этот флажок отключен, то в строку статуса окна программы также не будет выводиться справочная информация по пунктам меню.

Включение флажка **Disable Panel Docking** отменяет возможность "приклеивания" панелей к краям окна программы. Включение флажка **Disable Timeline Docking** отменяет ту же самую возможность для временной шкалы. О временной шкале и анимации вообще см. *часть 3*.

Если включен флажок **Span Based Selection**, то при щелчке на растянутом кадре будет выделен весь кадр. Если этот флажок выключен, то при щелчке будет выделен только тот промежуточный кадр, по которому щелкнули мышью. Фактически, флажок **Span Based Selection** заставляет Flash MX вести себя как предыдущая версия этого пакета Flash 5. Изначально он отключен, включите его, если вы переходите на Flash MX с предыдущей версии.

Если включен флажок **Named Anchor on Scene**, Flash делает первый кадр каждой сцены именованным "якорем". О сценах и "якорях" см. *главу 13*.

С помощью переключателей **Highlight Color** можно задать цвет, которым будут выделяться экземпляры образцов и группы графических фрагментов. (О группах см. *главу 5*, а об экземплярах — *главу 10*.) Если выбран переключатель **Use this color**, вы сможете задать цвет в расположенном правее этого переключателя селекторе цвета. Если же выбран переключатель **Use layer color**, то Flash будет использовать текущий цвет линии.

Раскрывающийся список **Font Mapping Default** позволяет установить шрифт, который будет подставляться вместо отсутствующих в системе шрифтов. Выберите любой шрифт, который вам нравится.

Вкладка **Editing** диалогового окна **Preferences** служит для настройки параметров рисования и правки графики на рабочем листе. Содержимое этой вкладки показано на рис. 3.2.



Рис. 3.2. Диалоговое окно **Preferences** (вкладка **Editing**)

Скажем сразу: будет лучше, если вы перед тем, как начнете изменять эти параметры, ознакомитесь с *главой 5* этой книги и, главное, приобретете некоторый практический опыт. Установки, заданные по умолчанию, как правило, подходят практически в любом случае, и, чтобы менять их, нужно представлять, к чему это может привести.

Группа флажков **Pen Tool** позволяет задать параметры инструмента "перо". Давайте их рассмотрим.

Флажок **Show Pen Preview** включает показ линий, рисуемых с помощью инструмента "перо". (С помощью инструмента "перо" рисуются прямые и кривые линии.) При выключенном флажке во время рисования, перед созданием конечной точки получающаяся линия не показывается. Если же вы включите этот флажок, Flash будет показывать "резиновую" линию, тянущуюся от начальной точки до курсора мыши. Это может помочь при рисовании, но отнимает много процессорного времени, поэтому на медленных компьютерах флажок **Show Pen Preview** лучше отключить.

Флажок **Show Solid Points** задает режим отображения ключевых точек — угловых и точек искривления. Если он включен (а он включен по умолчанию), невыбранные ключевые точки показываются в виде сплошных кружков и прямоугольников, а выбранные — в виде полых. Если же он выключен, то невыбранные точки, наоборот, показываются в виде полых кружков и прямоугольников, а выбранные — в виде сплошных. Включить или отключить этот флажок — на наш взгляд, дело вкуса и привычки.

Флажок **Show Precise Cursors** изменяет вид курсора мыши при выбранном инструменте "перо". Если он выключен, курсор мыши имеет вид чертежного рейсфедера. Если же этот флажок включить, курсор мыши примет вид небольшого прицела, что, по мнению разработчиков Flash, должно повысить точность его позиционирования. Включите этот флажок, если хотите "бить без промаха".

Группа флажков **Vertical Text** служит для задания специальных параметров текста, написанных на неевропейских языках. Все эти флажки по умолчанию отключены, включите их только в том случае, если разрабатываете графику для азиатских стран.

Если включить флажок **Default Text Orientation**, то текст по умолчанию будет писаться сверху вниз. Это необходимо, если требуется написать текст на одном из азиатских языков.

Включение флажка **Right to Left Text Flow** заставляет Flash писать текст справа налево. Это также нужно для некоторых азиатских языков.

Включение флажка **No Kerning** отменяет кернинг для вертикального текста. (О кернинге и вообще о работе с текстом см. главу 7.) Но если вы хотите включить этот флажок (отключить кернинг), то учтите, что внешний вид символов текста может при этом значительно ухудшиться.

Теперь рассмотрим группу элементов управления **Drawing Settings**. Она позволяет задать параметры рисования, распознавания и сглаживания линий.

Раскрывающийся список **Connect lines** устанавливает, как близко рисуемая линия должна находиться от уже нарисованной, чтобы Flash соединил их. Предоставляет для выбора три пункта: **Must be close** (рисуемая линия долж-

на быть рядом с нарисованной), **Normal** (обычное поведение; выбран по умолчанию) и **Can be distant** (может быть удалена). Этот флажок также задает, как близко от горизонтали или вертикали должна проходить рисуемая линия, чтобы Flash сделал ее, соответственно, горизонтальной или вертикальной. Если включен модификатор "притягивание", он еще позволяет задать, насколько близко один графический фрагмент должен находиться от другого, чтобы Flash объединил их.

Раскрывающийся список **Smooth curves** задает параметры сглаживания кривых, рисуемых с помощью "карандаша" при включенных режимах **Straighten** или **Smooth**. (Режимы "карандаша" выбираются с помощью модификатора "режим карандаша".) Имеет четыре пункта: **Off** (нет сглаживания), **Rough** ("грубые" кривые), **Normal** (обычное поведение; выбран по умолчанию) и **Smooth** (гладкие кривые).

Раскрывающийся список **Recognize lines** устанавливает, насколько линии, рисуемые с помощью "карандаша" при включенных режимах **Straighten** или **Smooth**, должны быть близки к прямым, чтобы Flash сделал их прямыми. (Режимы "карандаша" выбираются с помощью модификатора "режим карандаша".) Имеет четыре пункта: **Off** (линии не спрямляются), **Strict** (линии должны быть строго прямыми с минимальными отклонениями), **Normal** (обычное поведение; выбран по умолчанию) и **Tolerant** (отклонения от "прямызны" могут быть достаточно велики).

Раскрывающийся список **Recognize shapes** задает, насколько фигуры, рисуемые с помощью "карандаша" при включенных режимах **Straighten** или **Smooth**, должны быть правильными, чтобы Flash распознал и перерисовал их. (Режимы "карандаша" выбираются с помощью модификатора "режим карандаша".) Имеет четыре пункта: **Off** (распознавание отключено), **Strict** (фигуры должны быть правильными с минимальными отклонениями), **Normal** (обычное поведение; выбран по умолчанию) и **Tolerant** (отклонения могут быть достаточно велики).

Раскрывающийся список **Click accuracy** определяет, насколько близко курсор мыши (при выбранной "стрелке выделения") должен находиться от графического фрагмента, чтобы Flash выбрал его. Имеет три пункта: **Strict** (курсор мыши должен находиться точно над фрагментом), **Normal** (обычное поведение; выбран по умолчанию) и **Tolerant** (отклонение может быть достаточно велико).

Вкладка **Clipboard** диалогового окна **Preferences** позволяет настроить параметры операций, связанных с буфером обмена Windows. Ее содержимое показано на рис. 3.3.

Группа элементов управления **Bitmaps** позволяет задать параметры изображения, копируемого в буфер обмена в растровом формате (о растровой графике см. главу 4).

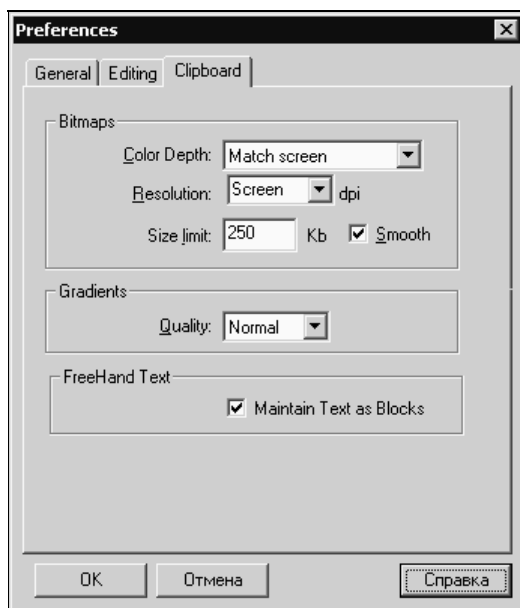


Рис. 3.3. Диалоговое окно **Preferences** (вкладка **Clipboard**)

Раскрывающийся список **Color Depth** служит для выбора цветового режима копируемого в буфер обмена изображения. В нем доступно семь пунктов:

- ☐ **None** — черно-белое изображение;
- ☐ **Match screen** — цветовое разрешение такое же, как у текущего видеорежима (этот пункт выбран по умолчанию);
- ☐ **4 bit color** — четырехбитный цвет, 16 доступных цветов;
- ☐ **8 bit color** — восьмибитный цвет, 256 доступных цветов;
- ☐ **16 bit color** — шестнадцатибитный цвет, 65536 доступных цветов (режим HiColor);
- ☐ **24 bit color** — 24-битный цвет, 16777216 доступных цветов (режим TrueColor);
- ☐ **32 bit color w/ alpha** — 24-битный цвет с каналом прозрачности (альфа-каналом).

Раскрывающийся список **Resolution** служит для задания разрешения растрового изображения. Выбранный по умолчанию пункт **Screen** задает то же самое разрешение, что и у экрана монитора. Остальные три пункта позволяют задать разрешение в 72, 150 и 300 точек на дюйм. Вы также можете ввести значение разрешения вручную прямо в этот список.

В поле **Size limit** вводится максимальный размер памяти, отводимой под растровое изображение, в килобайтах. Чем больше этот размер, тем большее

графическое изображение может быть помещено в буфер обмена. Доступны для ввода значения от 20 до 5000 килобайт.

Флажок **Smooth** включает или отключает сглаживание полученного растрового изображения. Рекомендуется оставлять его включенным, однако помните, что сглаживание может в некоторых случаях ухудшить качество изображения, особенно если оно имеет множество мелких деталей.

Группа элементов управления **Gradients** позволяет задать параметры изображения, копируемого в буфер обмена в векторном формате. В этой группе находится единственный элемент управления — раскрывающийся список **Quality**, задающий качество градиентных заливок. (О векторной графике см. главу 4, о градиентных заливках — главу 6.) Четыре пункта этого списка задают качество градиентных заливок: **None** (градиентные заливки отсутствуют), **Fast** (низкое качество), **Normal** (значение по умолчанию) и **Best** (самое высокое качество). Чем выше качество градиентных заливок, тем больше памяти они отнимают и тем дольше отображаются на экране при выводе векторного изображения.

Единственный элемент управления в группе **FreeHand Text** — флажок **Maintain Text as Blocks** — включает или отключает копирование текста в виде текстовых блоков. Если потом изображение, содержащее текстовые блоки, вставить в Macromedia Freehand, то содержащийся в блоках текст можно редактировать. Поэтому рекомендуется оставить этот флажок включенным.

Вкладка **Warnings** диалогового окна **Preferences** позволяет включить или отключить выдачу различных предупреждений. Содержимое этой вкладки показано на рис. 3.4.

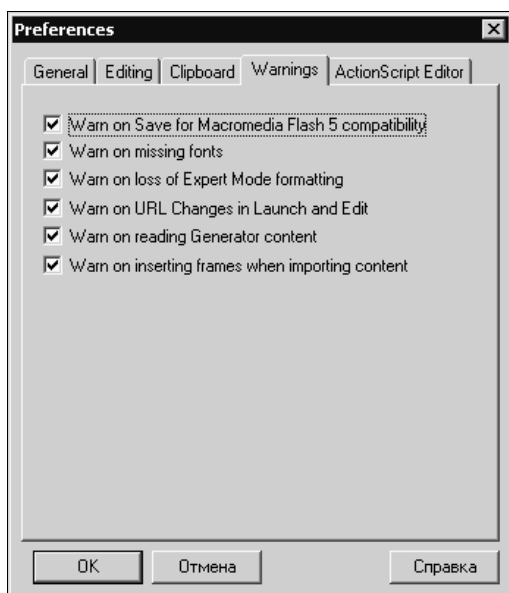
На этой вкладке расположена большая группа флажков. Все флажки, входящие в неё, перечислены в табл. 3.1. И все они по умолчанию включены.

Таблица 3.1. Флажки вкладки **Warnings** диалогового окна **Preferences**

Флажок	Назначение
Warn on Save for Macromedia Flash 5 compatibility	Включает или отключает выдачу предупреждения, выдаваемого при сохранении в формате Flash 5 документа, который не может быть правильно сохранен в этом формате
Warn on missing fonts	Включает или отключает выдачу предупреждения, выдаваемого при открытии документа, содержащего текст, написанный шрифтами, не установленными на компьютере

Таблица 3.1 (окончание)

Флажок	Назначение
Warn on loss of Expert Mode formatting	Включает или отключает выдачу предупреждения, выдаваемого при переключении панели Actions из профессионального режима в обычный о том, что форматирование текста сценария будет при этом потеряно
Warn on reading Generator content	Если включен, Flash помечает каждый объект Macromedia Generator красным крестиком
Warn on inserting frames when importing content	Включает или отключает выдачу предупреждения, выдаваемого при импорте аудио- и видеофайлов и добавлении при этом в фильм дополнительных кадров

Рис. 3.4. Диалоговое окно **Preferences** (вкладка **Warnings**)

Вкладка **ActionScript Editor** диалогового окна **Preferences** позволяет настроить параметры панели **Actions**. Ее содержимое показано на рис. 3.5.

Флажок **Automatic Indentation** включает или выключает создание автоматических отступов строк кода. При этом размер отступа задается в поле ввода **Tab Size**. По умолчанию флажок **Automatic Indentation** включен, а величина отступа равна четырем символам.

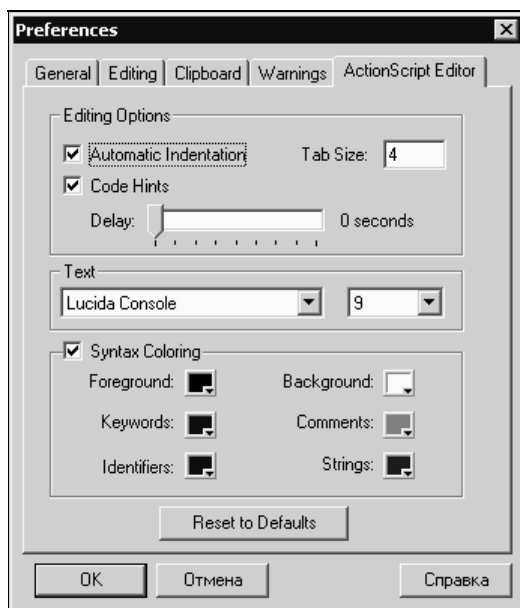


Рис. 3.5. Диалоговое окно **Preferences**
(вкладка **ActionScript Editor**)

Флажок **Code Hints** включает или отключает выдачу подсказок по синтаксису. Задержка перед появлением подсказки устанавливается с помощью движкового регулятора **Delay**. По умолчанию флажок **Code Hints** включен, а величина задержки равна нулю (нет задержки).

С помощью двух раскрывающихся списков, расположенных в группе **Text**, задаются параметры текста. В левом списке задается шрифт текста, а в правом — его размер. Для отображения текста программы рекомендуется выбирать моноширинные шрифты, например, Courier New или Lucida Console, — набранный ими текст программы будет выглядеть лучше.

Флажок **Syntax Coloring** позволяет включить или отключить цветовую подсветку синтаксиса текста программы. Это очень полезно для начинающих пользователей, поэтому данный флажок изначально включен. Ниже флажка **Syntax Coloring** расположены шесть селекторов цвета, позволяющих задать цвет следующих элементов:

- ☐ **Foreground** — основного текста программы;
- ☐ **Keywords** — ключевых слов языка программирования ActionScript;
- ☐ **Identifiers** — встроенных идентификаторов;
- ☐ **Background** — фона текста программы;
- ☐ **Comments** — комментариев;
- ☐ **Strings** — строковых констант.

Кроме того, на вкладке **ActionScript Editor** расположена кнопка **Reset to Defaults**, позволяющая быстро установить фабричные настройки редактора кода.

Задав нужные настройки, нажмите кнопку **OK** для их сохранения. Если вы передумали менять их, нажмите кнопку **Cancel**.

Настройка клавиатурных комбинаций

Настройки клавиатурных комбинаций Flash осуществляются в диалоговом окне **Keyboard Shortcuts**. Чтобы вызвать его на экран, выберите пункт **Keyboard Shortcuts** в меню **Edit**. Само это диалоговое окно показано на рис. 3.6.

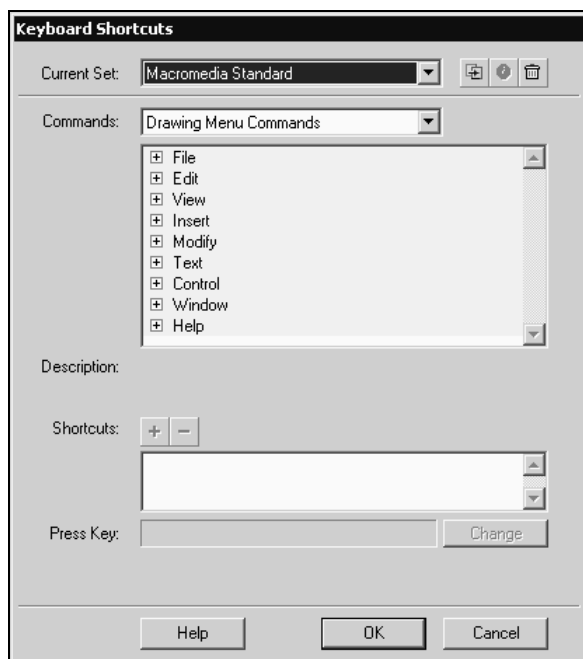


Рис. 3.6. Диалоговое окно **Keyboard Shortcuts**

Скажем сразу, что Macromedia Flash MX поставляется с шестью предопределенными наборами клавиатурных комбинаций. Каждый из этих наборов объединяет клавиатурные комбинации, применяемые в пяти популярных графических программах. Выбирается нужный набор с помощью раскрывающегося списка **Current Set**, в котором доступны шесть пунктов:

- ☐ **Macromedia Standard** — стандартный набор клавиатурных комбинаций, используемый по умолчанию;
- ☐ **Fireworks 4** — клавиатурные комбинации Macromedia Fireworks 4;

- ❑ **Flash 5** — клавиатурные комбинации, используемые в Macromedia Flash 5;
- ❑ **Freehand 10** — клавиатурные комбинации Macromedia Freehand 10;
- ❑ **Illustrator 10** — клавиатурные комбинации Adobe Illustrator 10;
- ❑ **Photoshop 6** — клавиатурные комбинации Adobe PhotoShop 6.

Если вас не удовлетворяет ни один из этих predetermined наборов, вы можете создать свой на базе одного из них. Для этого выберите в списке **Current Set** набор, наиболее близкий к вашим потребностям, и нажмите кнопку **Duplicate Set** (рис. 3.7), расположенную правее этого списка. На экране появится диалоговое окно **Duplicate** (рис. 3.8). Введите в единственное поле ввода **Duplicate Name** имя нового набора клавиатурных комбинаций и нажмите кнопку **OK**. Если же вы передумали создавать новый набор, нажмите кнопку **Cancel**.



Рис. 3.7. Кнопка **Duplicate Set**

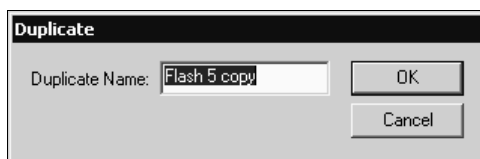


Рис. 3.8. Диалоговое окно **Duplicate**

Сразу после создания нового набора его имя появится в раскрывающемся списке **Current Set**. Теперь вы можете изменять клавиатурные комбинации, входящие в этот набор, как вам заблагорассудится.

Прежде всего, обратите внимание на раскрывающийся список **Commands**, расположенный чуть ниже списка **Current Set**. С помощью трех пунктов этого списка вы можете выбрать, какие именно клавиатурные комбинации вы будете изменять:

- ❑ **Drawing Menu Commands** — команды меню программы, отображаемые в обычном режиме;
- ❑ **Drawing Tools** — инструменты, доступные в главном инструментарии;
- ❑ **Test Movie Menu Commands** — команды меню программы, отображаемые в режиме тестирования фильма (о тестировании фильма см. главу 13);
- ❑ **Action Panel Commands** — команды, доступные в панели действий.

Когда вы выберете один из пунктов списка **Commands**, соответствующий набор команд меню или инструментов появится в иерархическом списке, расположенном ниже. Если были выбраны пункты **Drawing Menu Commands**

или **Test Movie Menu Commands**, то отображается иерархический список, "деревьями" которого являются меню, "поддеревьями" — подменю, а пунктами — соответствующие пункты меню программы. Если же были выбраны пункты **Drawing Tools** или **Action Panel Commands**, отображается обычный список, содержащий названия инструментов или команд. Кроме названия собственно пункта меню, инструмента или команды, пункт списка также содержит обозначения всех клавиатурных комбинаций, привязанных к этому пункту меню, инструменту или команде. Так что вы сразу можете увидеть, подходит вам эта клавиатурная комбинация или нет.

Когда вы выбираете любой из пунктов иерархического списка, ниже его появляется краткое описание пункта меню, инструмента или команды. А еще ниже, в списке **Shortcuts**, появляются все привязанные к нему клавиатурные комбинации.

Чтобы изменить какую-либо клавиатурную комбинацию, выберите ее в списке **Shortcuts**. Выбранная комбинация клавиш появится в поле ввода **Press Key** ниже списка **Shortcuts**. Выделите все содержимое поля ввода **Press Key** и нажмите нужную клавиатурную комбинацию. Учтите, что Flash требует, чтобы клавиатурные комбинации, задаваемые пользователем, обязательно включали в себя клавишу <Ctrl>. После этого нажмите кнопку **Change**, и новая комбинация клавиш заменит старую в списке **Shortcuts**.

Чтобы добавить новую клавиатурную комбинацию, нажмите кнопку со знаком "плюс", расположенную над списком **Shortcuts**. В поле ввода **Press Key** появится надпись "empty". Так же, как и при изменении комбинации клавиш, выделите содержимое поля ввода **Press Key** и нажмите нужную клавиатурную комбинацию. После этого нажмите кнопку **Change**, и новая комбинация клавиш добавится в список **Shortcuts**.

Чтобы удалить ненужную или ошибочно введенную комбинацию клавиш, выберите ее в списке **Shortcuts**. Далее нажмите кнопку со знаком "минус", расположенную над списком **Shortcuts**. Удаленная клавиатурная комбинация тотчас исчезнет из списка **Shortcuts**.

Вы также имеете возможность переименовать созданные вами наборы клавиатурных комбинаций. Для этого выберите его в списке **Current Set** и нажмите кнопку **Rename Set** (рис. 3.9), расположенную правее этого списка. На экране появится диалоговое окно **Rename**, аналогичное окну **Duplicate** (см. рис. 3.8). Введите в единственное поле ввода **New Name** новое имя набора и нажмите кнопку **OK**; если вы передумали, нажмите кнопку **Cancel**.



Рис. 3.9. Кнопка **Rename Set**

Также Flash предоставляет возможность удалить ненужный или ошибочно созданный набор клавиатурных комбинаций. Для этого выберите его в списке **Current Set** и нажмите кнопку **Delete Set** (рис. 3.10), расположенную правее этого списка. На экране появится диалоговое окно **Delete Set** (рис. 3.11). Выберите в списке, занимающем большую часть этого окна, нужный набор и нажмите кнопку **Delete**; если вы передумали, нажмите кнопку **Cancel**.



Рис. 3.10. Кнопка **Delete Set**

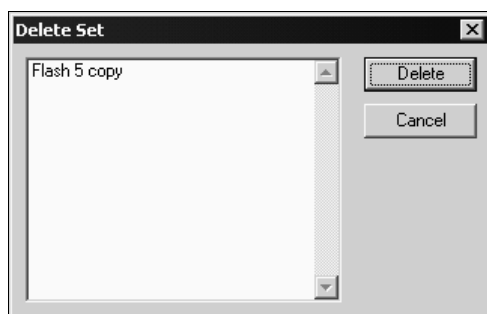


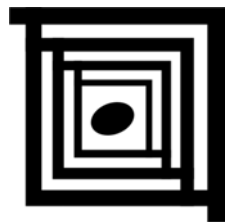
Рис. 3.11. Диалоговое окно **Delete Set**

Внимание!

Вы не можете изменять, переименовывать и удалять встроенные во Flash наборы клавиатурных комбинаций. Если вы хотите изменить один из этих наборов, вам придется создать его копию. Однако вы можете просматривать содержимое встроенных наборов, пользуясь иерархическим списком пунктов меню и инструментов и списком **Shortcuts**.

Создав новые или внося изменения в старые наборы клавиатурных комбинаций, нажмите кнопку **ОК** для их сохранения. Чтобы отказаться от изменений и вернуть предыдущие настройки, нажмите кнопку **Cancel**.

Часть II



Работа со статичной графикой

Глава 4. Форматы статичной графики

Глава 5. Рисование

Глава 6. Работа с цветом

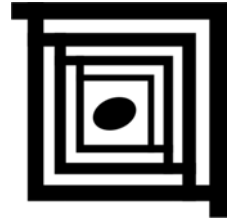
Глава 7. Работа с текстом

Глава 8. Импорт графики

Глава 9. Работа с графическими фрагментами

Глава 10. Образцы и библиотеки. Проводник Flash

Глава 11. Публикация и экспорт статичной графики



Глава 4

Форматы статичной графики

Итак, с пользовательским интерфейсом Macromedia Flash мы разобрались. Как выполняются типичные для всех Windows-приложений задачи, выяснили. Теперь самое время приступить к работе над графикой.

Да, мы так и поступим. Сначала рассмотрим создание статичной графики, потом перейдем к анимации. Ну, а уж после того, как мы научимся делать анимированные картинки, можно рассмотреть и программирование во Flash: интерактивные фильмы, пользовательские интерфейсы интернет-программ и т. п. Таким образом, к концу этой книги вы узнаете все, что может Flash, и сможете использовать это "все" в своих целях.

Но не торопитесь. Сначала немного поговорим о самой компьютерной графике.

Вы скажете: мы уже говорили о ней во введении. Мы узнали, какое место в компьютерном мире вообще и в Интернете в частности она занимает сейчас, даже познакомились с краткой ее историей. Да, кое-что вы уже знаете, но этого для настоящей работы недостаточно. Современный компьютерный художник должен знать значительно больше.

Чтобы полноценно работать с компьютерной графикой, нужно знать, в каких форматах она хранится, что они собой представляют, и какой формат в каком случае используется. *Формат* хранения графики — это способ записи графической информации в файле на диске. Чтобы корректно отобразить тот или иной графический файл, программное обеспечение должно поддерживать формат, в котором он записан.

Форматов хранения графической информации трудолюбивое человечество наплодило великое множество. Основная их масса была создана достаточно давно, вместе с программами для обработки графики, т. е. они являлись фирменным, "родным" форматом для той или иной программы. Очень немногие графические форматы были задуманы специально для обмена графическими файлами между разными программами (межпрограммные или программно-независимые форматы). К первым форматам можно отнести известный формат BMP, созданный специально для хранения графики в Windows-

программах, а ко вторым — популярнейший формат GIF. В дальнейшем многие форматы графики исчезли (зачастую вместе с программами, для которых они были созданы), и лишь малое число их дожило до нашего времени.

В живом мире действует жестокий закон эволюции. Это же справедливо и для мира компьютеров. Любое устройство, программа, любая технология нежизнеспособны, если они не найдут для себя "экологическую нишу", т. е. будут использоваться людьми. Так же как и любой графический формат. Те из них, что остались "живы", более или менее активно используются и покрывают практически все нужды современных компьютерных художников. Их-то и поддерживает большая часть современного графического программного обеспечения.

Их-то мы и рассмотрим в этой главе.

Растровая и векторная графика

Но, прежде всего, нужно рассказать о двух принципиально разных подходах к созданию и хранению графики. Собственно, с этого следовало бы и начать.

Растровая графика

Если взять обычную газетную иллюстрацию и хорошенько ее рассмотреть, желательно под лупой, то можно увидеть, что она на самом деле состоит из множества точек. Эти точки могут быть как жирно-черными (в тех местах, где на иллюстрации виден глубокий черный фон), так и более или менее серыми (где присутствуют полутона) или вообще быть почти незаметными (на белых местах). Если рассматривать такую иллюстрацию на некотором расстоянии, а не вплотную, как сделали мы, отдельные точки сливаются в единое изображение. Можно сказать, что обычная газетная иллюстрация — классический пример *растровой* графики.

Растровая графика впервые была создана полиграфистами для того, чтобы с минимальным расходом краски печатать на бумаге иллюстрации. При этом на исходное изображение словно бы накладывается тонкая сетка, называемая *растром*, и изображение оказывается разбитым на множество точек (пикселей). После этого остается только вычислить интенсивность черного цвета в каждой точке раstra или, для цветного изображения, цвет точки и, возможно, ее прозрачность. Этот процесс называется *растеризацией*. Полученные в результате растеризации данные либо сразу же переносятся на типографские пленки, либо записываются в файл для дальнейшей обработки.

На рис. 4.1 показан небольшой пример растрового изображения — литера А, как она отображается на экране компьютера. Как видите, она состоит из множества разноцветных — белых, серых и черных — точек, складывающихся в ее изображение. Для вашего удобства на этот рисунок наложена растровая сетка.

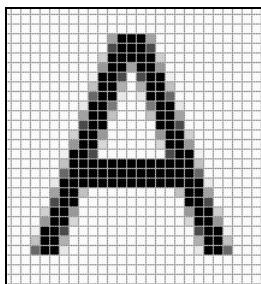


Рис. 4.1. Пример растровой графики — литера А

Вообще, все, что отображается на экране компьютера — суть растровая графика. Дело в том, что компьютерный экран сам представляет собой не что иное, как растр, т. е. набор точек, цвет которых можно изменять программно. Поэтому самыми первыми компьютерными графическими форматами были именно растровые.

В случае растровой графики в графическом файле, как вы поняли, сохраняется упорядоченный набор (опытные компьютерщики говорят — массив) значений цветов точек растра. Разумеется, где-то в начале файла, в его заголовке, должен быть записан размер растра, например, 320x200 точек, иначе программное обеспечение не сможет правильно обработать файл. Также иногда в файл записываются дополнительные данные: сведения о создателе, данные, добавленные программой, в которой редактировался файл, и пр.

Кстати о цвете. Как вы знаете, современные компьютеры могут выводить на экран одновременно множество различных цветов. На самом деле, количество одновременно отображаемых цветов зависит от текущего цветового видеорежима, точнее, от количества бит, выделяемых на обозначение цвета одного экранного пиксела (табл. 4.1). Чем больше выделяется бит на кодирование одного пиксела, тем большее количество цветов может отобразить компьютер одновременно.

Таблица 4.1. Цветовые видеорежимы, доступные в современных компьютерах

Количество бит на пиксел	Количество одновременно отображаемых на экране цветов	Другое название цветового видеорежима
1	2 (черно-белое изображение)	MONO
2	4	—
4	16	CGA
8	256	—
16	65536	HiColor
24 (32)	16776576	TrueColor

Та же ситуация наблюдается и с самими графическими изображениями. На каждую точку изображения отводится определенное количество бит. Поэтому изображение может содержать только ограниченный набор цветов. Другое дело, что этот набор может быть очень велик, например, 65536 цветов (HiColor) или все 16776576 (TrueColor). Поэтому в последних двух случаях говорят о *фотореалистичной* или *полноцветной* графике.

Растровая графика имеет как достоинства, так и недостатки. Перечислим их, начав, разумеется, с достоинств.

- ❑ Простота вывода. В самом деле, чтобы вывести растровое изображение на экран монитора или принтер, не требуются сверхсложные вычисления. Отображение растровой графики не "нагружает" слишком сильно процессор компьютера, а значит, вывод изображения происходит очень быстро. Какая-либо дополнительная обработка при этом отсутствует, за исключением, может быть, подстройки цветов.
- ❑ Размер массива точек, а значит, графического растрового файла, зависит от геометрических размеров самого изображения и количества цветов, использованных в нем, точнее, количества бит на точку. Размер растрового изображения не зависит от его сложности. Это означает, что маленькие черно-белые изображения занимают меньше места, чем большие полноцветные, выполненные в формате TrueColor. Данной особенностью растровой графики часто пользуются Web-дизайнеры.
- ❑ Высокая точность и достоверность передачи полутоновых изображений, например, сканированных картин и фотографий. В самом деле, если использовать достаточно большое разрешение и цветовой режим TrueColor, то цифровая копия визуально не будет отличаться от оригинала.

Теперь рассмотрим недостатки растровой графики.

- ❑ Размер массива точек зависит от геометрических размеров самого изображения и количества цветов, использованных в нем. Как часто происходит, недостаток является продолжением достоинства. А именно, если мы сохраним в растровом формате простенькое, но полноцветное (цветовой режим TrueColor) и, вдобавок, огромного размера изображение, оно займет многие мегабайты, а то и десятки мегабайт. Профессиональные полиграфисты, работающие с фотореалистичной графикой, часто имеют дело с такими большими изображениями.
- ❑ Растровая графика зависит от разрешения устройства вывода: монитора или принтера. В самом деле, если вывести изображение размером 640×480 точек на экран монитора с таким же разрешением, то этот рисунок займет весь экран целиком. Если же его вывести при разрешении 1024×768, то на экране отобразится только часть рисунка. Как видите, в этих двух случаях размеры нашего рисунка будут сильно различаться, что не всегда приемлемо.

- ❑ Растровая графика не масштабируется, т. е. при изменении ее размеров сильно падает ее качество.

Последний пункт нужно пояснить особо. Предположим, что мы имеем небольшое растровое изображение. И возникло у нас желание его увеличить. Мы открываем его в программе графического редактора, выполняем увеличение. Результат показан на рис. 4.2.

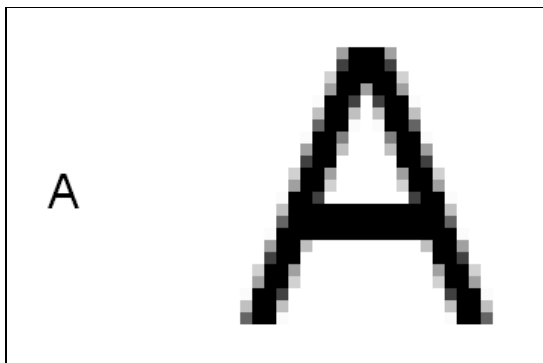


Рис. 4.2. Результат увеличения растрового изображения

Слева на этом рисунке показано исходное изображение, то, которое мы увеличивали. Справа — результат увеличения. Видно, что каждая точка правого изображения увеличилась до огромного "кирпича", в результате чего правое изображение выглядит абсолютно не эстетично. Вывод: растровые изображения нельзя увеличивать без потери качества, т. е. растровая графика не масштабируема. Что и требовалось доказать.

Как можно преодолеть этот недостаток?

Во-первых, по мере возможности не масштабировать растровые изображения. Лучше всего создавать их именно такого размера, какой нужен. (В крайнем случае, их можно уменьшить или совсем немного увеличить, чтобы не было видно точечной структуры.) Впрочем, современное программное обеспечение, в частности Microsoft Internet Explorer 6.0, может более-менее корректно масштабировать растровую графику.

Во-вторых, использовать достаточно мощные графические пакеты, например, последние версии Adobe Photoshop, для масштабирования графики. Реализованные в них алгоритмы масштабирования графики позволяют менять размеры изображений почти без потери качества. Поставляемый в составе Microsoft Windows простейший графический редактор Paint этого не может.

Что касается второго, точнее, первого недостатка растровой графики — прямой зависимости размера графического файла от геометрических размеров изображения — то он также практически решен. Дело в том, что подавляющее большинство графических форматов предоставляют возможность

сжатия массива данных. При этом используются специальные алгоритмы, оптимизированные именно для сжатия графики. В результате сжатия размер файла сильно уменьшается. Недостатками такого подхода являются повышенные затраты процессорного времени на распаковку изображения и возможная потеря данных при использовании слишком сильного сжатия.

Вот и все о растровой графике. Предоставим теперь слово конкурирующей стороне...

Векторная графика

Рассказ о векторной графике мы начнем с небольшого допущения. Предположим, что любое, даже очень сложное графическое изображение можно разбить на простейшие элементы: прямые, кривые, эллипсы, прямоугольники и т. п. Эти простейшие элементы, называемые графическими примитивами, можно описать определенными формулами. В результате получится набор параметров, используя которые, можно точно воссоздать исходный набор графических примитивов, а значит, и исходное изображение.

Такая графика, состоящая из отдельных примитивов, называется *векторной* графикой. Именно в таком виде сохраняются изображения Flash.

В качестве примера приведу все ту же литеру А в том виде, в котором она хранится в памяти компьютера (а не выводится на экран). Если вы посмотрите на рис. 4.3, то увидите, что она состоит из трех примитивов — прямых. Мы специально немного отделили эти прямые друг от друга, чтобы вам было понятнее.

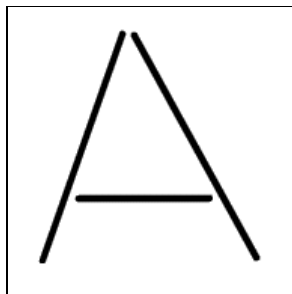


Рис. 4.3. Векторное представление литеры А

Кроме уже знакомых нам графических примитивов — прямых и кривых линий, прямоугольников и эллипсов — векторное изображение может содержать также текст и растровые изображения. Таким образом, можно сказать о рождении фактически нового вида графики — *гибридной* — о котором мы поговорим чуть ниже.

Но, спросите вы, как же компьютер выводит векторные изображения на экран? Ведь экран компьютера — это растр, и он должен сначала преобразо-

вать изображение в набор точек, т. е. растривать его. И вы будете правы. Да, компьютер растрирует векторную графику, для чего дополнительно тратятся системные ресурсы компьютера. Затраты системных ресурсов на растрезирование — один из главных недостатков векторной графики, но неоспоримые достоинства с лихвой его перевешивают.

Это становится ясно при перечислении достоинств векторной графики.

- ❑ Размер файла векторного изображения не зависит от геометрических размеров самого рисунка. Ведь в этом случае в файл записывается отнюдь не огромный массив цветовых значений точек, а только тип и параметры всех задействованных примитивов. Как видите, объемы сохраняемых данных несопоставимы. Поэтому даже при больших размерах векторное изображение может быть записано в файл совсем маленького размера.
- ❑ Прекрасная масштабируемость. В самом деле, чтобы изменить размеры изображения, нужно всего лишь изменить один-единственный параметр всех формул примитивов и перерисовать изображение. Посмотрите на рис. 4.4 — векторное изображение в любом масштабе выглядит идеально.
- ❑ Как следствие масштабируемости, независимость от разрешения устройства вывода: монитора или принтера.
- ❑ Исключительные возможности по обработке изображений. Векторные изображения можно поворачивать, искажать, отображать зеркально, перекрашивать, делать полупрозрачными и т. п. Примеры этого представлены на рис. 4.5. С растровыми изображениями эти операции вряд ли возможно выполнить, а если и можно, они отнимут слишком много системных ресурсов.

Кстати, знаете ли вы, что обычные компьютерные шрифты, которыми вы пользуетесь при наборе документов, хранятся в своих файлах в векторном виде? Это так называемые шрифты формата TrueType (файлы с расширением ttf). Благодаря векторному представлению, они исключительно хорошо масштабируются до любых размеров. Однако системные шрифты, используемые для вывода надписей, заголовков окон, пунктов меню, хранятся все же в растровом виде, чтобы зря не расходовать при выводе системные ресурсы.



Рис. 4.4. Векторное изображение идеально масштабируется



Рис. 4.5. Векторное изображение можно вращать и искажать

Теперь перечислим недостатки векторной графики и укажем пути их преодоления.

- ❑ Размер файла векторного изображения зависит от его сложности. В самом деле, чем сложнее изображение, чем больше примитивов включает оно в себя, тем больше потребуется сохранить данных.
- ❑ Вывод векторной графики (а именно, ее растеризация) отнимает много времени и системных ресурсов. В этом смысле растровая графика "работает" быстрее.
- ❑ Невозможность преобразования полутонового растрового изображения в векторное (*векторизации*) без больших потерь в качестве.

Первые два недостатка вполне можно преодолеть. Во-первых, не нужно создавать слишком уж сложных векторных изображений. Во-вторых, по возможности, нужно комбинировать векторную и растровую графику — современные графические пакеты предоставляют такую возможность. В-третьих, сложную векторную графику для распространения потребителям (или поклонникам) обязательно следует перевести в растровую.

К сожалению, третий недостаток преодолеть практически невозможно. Все изобразительное искусство, созданное людьми до появления компьютера и векторной графики, в большинстве случаев — растровое. Если попытаться превратить его в векторное с помощью программы векторизации, наступит момент, когда изображение окажется слишком сложным для этой программы. И качество результирующего векторного изображения будет очень низким. Так что пусть имеющаяся у вас цифровая репродукция Веласкеса (или Бориса Вальехо) остается в растровом формате.

Вместе с этим, векторная графика неплохо подходит для сохранения штриховых изображений. Так что если у вас случайно завалялись чертежи или гравюры Гойи — почему бы не попробовать!

Все, о векторной графике нам больше сказать нечего. Давайте еще раз сравним эти два вида графики и выясним, какой из них и в каких случаях лучше.

Гибридная графика

Собственно, гибридная графика — это разновидность векторной графики, содержащей внутри себя растровые изображения. Благодаря этому часто удается преодолеть главнейшие недостатки и растровой, и векторной графики: большой размер (у растровой графики) и невозможность точной передачи полутоновых изображений (векторной графикой).

Гибридное изображение разбито на фрагменты, часть из которых выполнена в растровом, а часть — в векторном виде. Обычно все полутоновые фрагменты изображения (картины, фотографии на коллажах) создаются в растровом виде. Векторными делают штриховые фрагменты: надписи, линейки,

выноски, схемы, карты и т. п. Таким образом достигается компромисс между качеством изображения и размером файла.

Все современные редакторы векторной графики, в том числе Flash, предоставляют возможность создания гибридной графики. Фактически, их можно даже назвать редакторами гибридной графики. Также гибридную графику создают и программы настольных издательств.

Применение разных видов графики

Если на свете существует два вида компьютерной графики, значит, это кому-то нужно. Давайте же выясним, кому и, главное, в каких случаях это нужно.

Главный козырь растровой графики — точность передачи сканированных изображений. При этом растровая графика занимает тем больший объем, чем больше само изображение, и не предоставляет никаких возможностей по его обработке (масштабированию, искажению, повороту, перекрашиванию). Главный козырь векторной графики — исключительные возможности обработки изображения. Недостаток — невозможность сохранения в векторном виде полутоновых изображений в близком к оригиналу виде. Исходя из этого, можно определить область применения для каждого вида компьютерной графики.

Итак, растровая графика применяется для:

- сохранения полутоновых изображений (сканированные или изначально нарисованные на компьютере картины, фотографии);
- создания небольших по размеру изображений для оформления программ или Web-страниц. В этом случае, как правило, критичны скорость вывода на экран и размер "ответственного" за вывод программного кода, а отнюдь не размер изображения.

Векторная графика лучше всего подойдет, если нужно:

- сохранить штриховые изображения (карты, чертежи, рисунки карандашом, гравюры) в электронном виде;
- создать небольшие изображения, которые в дальнейшем будут всячески обрабатываться при выводе. Хороший пример таких изображений — шрифты формата TrueType, которые при выводе на экран не только масштабируются, но и раскрашиваются в разные цвета, поворачиваются и т. п.

В остальных случаях вы можете использовать как векторную, так и растровую графику. Помните только об ограничениях, присущих обоим этим видам графики, и, разумеется, знайте об их преимуществах.

Вот и все о двух видах графики: растровой и векторной. Конкретно различные форматы представления и растровой, и векторной графики мы рас-

смотрим ниже, там же поговорим об их достоинствах и недостатках и опишем область применения каждого формата.

Осталось только напомнить, что Flash — формат векторной графики.

Форматы графических файлов

Теперь поговорим о самых распространенных на сегодняшний день форматах графических файлов. Как правило, все эти форматы поддерживаются Flash, если при описании формата не будет сказано противоположное.

Растровые форматы

Большинство распространенных графических форматов — растровые. Сначала их и рассмотрим.

BMP

Формат BMP (BitMaP — битовая матрица) — простейший формат записи растровых изображений. Разработан фирмой Microsoft для сохранения графики в операционной системе Windows и совместимых с ней программах. Для этого поддержка формата BMP была встроена непосредственно в ядро системы Windows. Также известен под названием DIB (Device Independent Bitmap — битовая матрица, не зависящая от устройства вывода).

Графика сохраняется в файлах с расширением bmp или dib (встречается очень редко). Поддерживает все цветовые видеорежимы. Графические данные могут быть сжаты с использованием простейшего алгоритма RLE (Run Length Encoding — кодирование с переменной длиной строки). Если сжатие не используется, размер графического файла может быть очень велик.

В настоящее время — один из самых распространенных графических форматов. Поддерживается практически всеми графическими программами. Из-за своей простоты требует для вывода очень мало системных ресурсов, поэтому основное его предназначение — хранение изображений, используемых как элементы пользовательского интерфейса операционной системы. В частности, именно в формате BMP хранятся системные "обои", заставки, иконки и т. п.

PCX

Формат PCX — один из самых старых графических форматов. Он был разработан в начале восьмидесятых годов не существующей ныне фирмой Z-Soft для собственного растрового графического редактора PC Paintbrush, работавшего в среде MS-DOS. Также поддерживался множеством других программ, работавших в среде DOS и Windows, и продолжает поддерживаться и поныне, хотя и является устаревшим.

Графика сохраняется в файлах с расширением `psx`. Поддерживает все цветовые видеорежимы, за исключением `TrueColor` и черно-белого. Всегда используется сжатие графических данных по алгоритму `ROB`.

Во времена господства `MS-DOS` в формате `PCX` создавалось подавляющее большинство компьютерной графики. Также этот формат часто использовался в играх для хранения графической информации. Сейчас же он устарел, а если и поддерживается графическими программами, то только исключительно ради совместимости.

Внимание!

Этот формат `Flash` не поддерживается. Чтобы импортировать графическое изображение в формате `PCX`, вам придется преобразовать его в другой формат, например, `BMP`, используя другую графическую программу.

GIF

Формат `GIF` (`Graphic Interchange Format` — формат обмена графикой) был разработан фирмой `CompuServe` в 1987 году для использования в собственной одноименной компьютерной сети. Получил огромное распространение в компьютерных сетях, в частности, в Интернете. Пожалуй, в настоящее время большинство компьютерной графики, использующейся в `Web-дизайне` и вообще для распространения изображений в Интернет, сохранено в этом формате.

Графика хранится в файлах с расширением `gif`. Поддерживаются цветовые видеорежимы до 256 цветов включительно. Для сжатия графики используется алгоритм `LZW`, разработанный израильскими математиками Лемпелом и Зивом. Графика может быть сохранена с *чередованием* (по-английски — `interleaving`) строк; в этом случае изображение как бы постепенно "проявляется" строка за строкой по мере загрузки файла.

В 1989 году формат `GIF` был расширен; новая версия стандарта получила название `GIF89A`. Во-первых, была введена поддержка прозрачности или "прозрачного" цвета, т. е. один цвет из всех доступных на изображении мог быть помечен как прозрачный, и сквозь него будет "просвечивать" фон изображения. (Профессиональные графики часто называют "прозрачный" цвет *альфа-каналом*, по-английски — `alpha channel` или просто `alpha`.) Во-вторых, появилась возможность сохранять в одном файле несколько изображений, которые могут демонстрироваться как фильм (так называемые "анимированные `GIF`-файлы"); о них мы поговорим в *главе 12*).

Формат `GIF` прекрасно подходит для сохранения изображений с резкими цветовыми переходами. В частности, `Web-дизайнеры` создают в этом формате элементы оформления и рекламные баннеры для своих страниц, а поддержка прозрачности и анимации им только на руку. Иногда в этом формате создаются начальные заставки и графические элементы интерфейса программ.

Ну и грех будет умолчать о том, что Microsoft Windows 98 и более поздние ее версии содержат в своем составе так называемый Активный рабочий стол. Одна из интересных возможностей, предоставляемых им, — отображение в качестве "обоев" рабочего стола файлов в формате GIF и JPEG (об этом формате см. ниже). К сожалению, Активный рабочий стол потребляет много ресурсов, поэтому используется довольно редко даже по его прямому назначению.

К несчастью, будущее этого формата весьма туманно. Еще в середине девяностых годов фирма CompuServe хотела получать авторские отчисления с продажи каждой программы, поддерживающей формат GIF, но тогда сетевому сообществу удалось отстоять его бесплатность. Теперь же фирма Unisys, владеющая патентом на алгоритм LZW, тоже хочет получать авторские отчисления, ведь именно этот алгоритм применен в формате GIF для сжатия графики. Судя по всему, этот формат в покое не оставят. Кроме того, 256 цветов по сегодняшним временам — очень мало. Поэтому ему уже пророчат преемника — формат PNG, описанный ниже.

В настоящее время формат GIF поддерживается практически всеми графическими программами. И, несмотря на все перипетии с авторскими правами, все еще активно используется.

PNG

Формат PNG (Portable Network Graphic — переносимая сетевая графика) разработан сообществом независимых программистов в качестве замены устаревшего и переходящего в разряд коммерческих продуктов формата GIF. Хотя он и поддерживается в настоящее время многими графическими пакетами, однако большой популярности в Интернете пока не снискал, хотя автору встречались сайты с PNG-графикой. Также используется для хранения графики, разрабатываемой в пакете интернет-графики Macromedia Fireworks.

Графика хранится в файлах с расширением png. Поддерживает все цветовые видеорежимы. Для сжатия графики применяется очень мощный алгоритм Deflate (буквально — "усыхание"), обеспечивающий более сильное сжатие по сравнению с LZW. Графика может быть сохранена с чередованием не только строк, но и столбцов, таким образом, изображение будет "проявляться" и по строкам, и по столбцам. Также поддерживаются 256 градаций прозрачности (альфа-канала) против всего двух у GIF и автоматическая коррекция яркости.

Однако, по сравнению с GIF, формат PNG имеет и недостатки. Первый огромный недостаток — PNG не поддерживает анимацию. Правда, вряд ли это так уж актуально в связи с повсеместным переходом Web-аниматоров на Flash. Второй недостаток — файлы формата PNG больше, чем GIF, примерно на один килобайт из-за того, что в заголовке файла хранится гораздо

больше информации. На мой взгляд, это тоже не очень значительный недостаток: получился файл чуть больше — ну и что? Так что, в принципе, вышеприведенные недостатки не повлияют на распространение формата PNG в Интернете.

Пока что формат PNG используется для хранения графики, разработанной в Fireworks, и в некоторых "продвинутых" сайтах. Однако, как уже говорилось, его поддерживают практически все графические пакеты.

JPEG

Формат JPEG (Joint Picture Encoding Group — группа кодировки неподвижных изображений) разработан одноименной группой программистов специально для распространения высококачественной графики в компьютерных сетях. Именно для этого он и используется в настоящее время. JPEG — второй по распространенности формат графики в Интернете.

Графика сохраняется в файлах с расширениями jpeg, jpe или jpg. Поддерживается только цветовой видеорежим TrueColor (24-битный цвет). Для сжатия графики используется исключительно мощный алгоритм под названием JPEG, фактически включающий в себя несколько алгоритмов сжатия для разных случаев. Этот алгоритм реализует *сжатие данных с потерями*, когда некоторая часть информации о цвете отбрасывается, и результирующий массив данных становится меньше. Во всех других форматах графики со сжатием применяются алгоритмы *сжатия данных без потерь*.

При использовании алгоритмов сжатия с потерями качество графики ухудшается. Чем сильнее сжатие, тем сильнее искажается изображение. Однако художник может регулировать процент сжатия, выбирая тем самым компромисс между качеством изображения и размером результирующего JPEG-файла.

Формат JPEG, в отличие от GIF и PNG, не поддерживает ни анимацию, ни прозрачность. Однако существует разновидность формата JPEG, называемая "прогрессивный JPEG" (progressive JPEG или p-JPEG), поддерживающая чередование строк.

Область применения формата JPEG достаточно узка — распространение высококачественной полутоновой графики в Интернете. Сканированные полутоновые изображения при использовании умеренного сжатия получаются очень даже неплохими. Особенно популярны "в народе" подборки картин Бориса Вальехо¹ и других так называемых "фэнтезийных" живописцев, сохраненные в JPEG-файлах.

Формат JPEG поддерживается практически всеми современными графическими редакторами. И Активным рабочим столом новых версий Windows.

¹ Известный художник-оформитель, работающий в стиле "фэнтези".

TIFF

Формат TIFF (Tag Image File Format — теговый файловый формат изображений) был разработан фирмой Aldus, разработчиком известного пакета настольного издательства PageMaker, для другого своего продукта — растрового редактора PhotoStyler, не дошедшего до наших времен. Применяется для сохранения высококачественной полноцветной графики без потери качества для издательских целей. Часто используется для обмена высококачественной графикой между пользователями различных программ.

Графика записывается в файлы с расширением tif или tiff. Поддерживаются все цветовые видеорежимы, прозрачность и несколько алгоритмов сжатия: LZW, Deflate и JPEG. Фирменной особенностью этого формата является возможность записи в графический файл так называемых *тегов*: специальных примечаний, вносимых художником или самой программой графического редактора.

Существует две разновидности формата TIFF: совместимый с PC и с Apple Macintosh. Это вызвано различиями в архитектуре перечисленных компьютерных платформ.

Формат TIFF поддерживается всеми графическими программами профессионального уровня. Более того, поддержка какой-либо программой этого формата говорит об ее профессиональной ориентации.

Внимание!

Этот формат Flash непосредственно не поддерживается. Чтобы импортировать графическое изображение в формате TIFF, вам придется либо установить на свой компьютер пакет Apple QuickTime, либо преобразовать изображение в другой формат, например, BMP, используя другую программу.

Другие растровые графические форматы

Осталось перечислить несколько менее распространенных форматов сохранения растровой графики (см. табл. 4.2). В основном, это фирменные форматы, разработанные для какого-либо программного продукта.

Таблица 4.2. Другие растровые графические форматы

Формат	Расширение файлов	Описание
MacPaint	pntg	Формат графических файлов редактора MacPaint
PhotoShop	psd	Формат пакета профессиональной растровой графики Adobe PhotoShop
PICT	pct, pic	Стандартный формат растровой графики в операционной системе Apple Macintosh. Выполняет ту же роль, что формат BMP в Microsoft Windows

Таблица 4.2 (окончание)

Формат	Расширение файлов	Описание
QuickTime	mov, qtif	Формат для сохранения неподвижной графики пакета Apple QuickTime
Silicon Graphics	sai	Стандартный формат растровой графики для рабочих станций Silicon Graphics (SGI)
TARGA	tga	Стандартный формат растровой графики для компьютеров Amiga

Внимание!

Flash непосредственно поддерживает только формат PICT. Чтобы импортировать изображение, сохраненное в другом формате (из перечисленных в табл. 4.2), вам нужно будет установить на свой компьютер пакет Apple QuickTime или преобразовать это изображение к одному из поддерживаемых форматов, используя другую программу.

Векторные форматы

Векторных форматов почему-то значительно меньше, чем растровых. Вероятно, это связано с тем, что реализовать обработку и вывод векторной графики несравнимо труднее, чем растровой. А люди — увы! — всегда идут по пути наименьшего сопротивления.

Shockwave/Flash

Разумеется, мы не могли не начать рассмотрение форматов векторной графики с "родного" формата Macromedia Flash. Как-никак, именно в нем нам предстоит сохранять свои творения.

Формат Shockwave/Flash разработан фирмой Macromedia для сохранения изображений и фильмов, созданных в пакете векторной графики Shockwave. Позднее на основе Shockwave был создан пакет интернет-графики Flash, "унаследовавший" этот формат. Поэтому говорят, что существует единый формат Shockwave/Flash.

Как вы уже знаете, фактически существует два формата представления графики Flash. Во-первых, это формат, в котором сохраняются подготавливаемые в среде Flash изображения и фильмы, — формат документов Flash. Во-вторых, формат, в котором хранится уже завершенная, экспортированная и подготовленная для публикации графика, которую можно загрузить в проигрыватель, — формат распространяемой графики Shockwave/Flash. Мы не будем описывать здесь различия между этими двумя форматами, поскольку они уже были описаны в *главе 1*.

Документы Flash сохраняются в файлах с расширением fla. Экспортированная же графика хранится в файлах с расширением swf.

Помимо самого пакета Flash оба этих формата поддерживаются другими продуктами фирмы Macromedia: Dreamweaver, Fireworks и др. Также эти форматы поддерживаются некоторыми другими графическими программами.

Windows Metafile и Enhanced Windows Metafile

Формат Windows Metafile — простейший формат записи векторных изображений. Разработан фирмой Microsoft для сохранения векторной графики в операционной системе Windows и совместимых с ней программах. Поддержка этого формата встроена непосредственно в ядро системы Windows.

Графика хранится в файлах с расширением wmf. Возможности формата исключительно слабы, более-менее сложную графику сохранить в нем невозможно.

В настоящее время формат Windows Metafile, в отличие от BMP, распространен очень мало, хотя поддерживается практически всеми графическими программами и требует очень мало системных ресурсов для вывода и обработки. Используется для хранения векторных изображений в некоторых программах (например, начальных заставок или элементов пользовательского интерфейса). В частности, Microsoft Word свой комплект картинок (так называемый "клипарт" от английского clipart) хранит в формате Windows Metafile. Вероятно, фирма Microsoft, хочет показать таким образом, что этот формат тоже пригоден для распространения графики.

Формат Enhanced Windows Metafile — дальнейшее развитие Windows Metafile. Он также разработан Microsoft, однако "в народ" почему-то не продвигался и мало-мальски широкого распространения поэтому не получил. Автору за всю его достаточно длинную карьеру компьютерщика не попадалось на одного файла этого формата. Остается добавить, что формат предписывает сохранять графику в файлах с расширением emf и поддерживается многими современными графическими пакетами.

Adobe Illustrator

Этот формат был разработан фирмой Adobe для векторного графического редактора Illustrator. В настоящее время поддерживается практически всеми пакетами векторной графики и используется для обмена векторными изображениями между пользователями различных программ.

Графика сохраняется в файлах с расширением ai. Формат очень устойчив к сбоям, испорченный файл, как правило, с большой вероятностью все же можно прочитать.

CorelDRAW!

Был разработан фирмой Corel для векторного редактора CorelDRAW!. Фактически позволяет хранить гибридную графику. Иногда используется для обмена графикой.

Графика сохраняется в файлах с расширением cdr. Имеет несколько привлекательных возможностей по сравнению с Adobe Illustrator (например, сжатие графики, причем растровая и векторная графики сжимаются отдельно), но несовместимость различных версий формата и невысокая устойчивость к сбоям отнюдь не идут на пользу его популярности.

Внимание!

Этот формат Flash не поддерживается. Чтобы импортировать графическое изображение в формате CorelDRAW!, вам придется преобразовать его в другой формат, например, Adobe Illustrator или Windows Metafile, используя другую графическую программу.

Encapsulated PostScript

Этот формат был разработан фирмой Adobe для обмена векторной графикой между пользователями различных программ. Собственно, это даже не формат, а целый язык, базирующийся на языке описания графики для высококачественных принтеров PostScript. Фактически, файл в этом формате можно скопировать на поддерживающий язык PostScript принтер, набрав в командной строке:

```
copy graphic_file.eps prn
```

и он будет напечатан.

Графика сохраняется в файлах с расширением eps. Такой файл представляет собой простой текстовый документ, содержащий набор команд для принтера напечатать тот или иной примитив. Таким образом, EPS-файл может быть отредактирован в любом текстовом редакторе при наличии знания языка PostScript.

Поддерживается практически всеми графическими пакетами. Поэтому используется для переноса больших графических изображений между различными программами.

VML

Формат VML (Vector Markup Language — язык векторной разметки) разработан фирмой Microsoft для использования в собственной программе Web-обозревателя Internet Explorer версии 5.0 и более поздних. Так же, как и Encapsulated PostScript, это не столько формат, сколько текстовый язык описания векторной графики. По задумке, фрагменты таких описаний помещаются внутрь Web-страниц, среди обычного HTML-кода, и описывают

их графические элементы. Таким образом, можно будет отказаться от традиционной Web-графики, представляющей собой внедренные элементы, хранящиеся в отдельных файлах.

Формат (или язык) VML появился совсем недавно, и пока что поддерживается немногими программами фирмы Microsoft, среди которых Microsoft Internet Explorer 5.0, Microsoft Word 2000. Само собой, говорить о каком-либо распространении этого формата пока еще рано.

Внимание!

Сведения о формате VML включены только для ознакомления. Flash его не поддерживает.

Другие векторные графические форматы

Опять же, перечислим пару менее распространенных векторных форматов, не вошедших в наш небольшой перечень, в табл. 4.3. Это фирменные форматы, разработанные специально под соответствующие программные продукты.

Таблица 4.3. Другие векторные графические форматы

Формат	Расширение файлов	Описание
AutoCAD DXF	dxf	Формат файлов системы автоматизированного проектирования AutoDesk AutoCAD
FreeHand	fh7, fh7, fh8, fh8, fh9, fh9, fh10	Формат пакета профессиональной векторной графики Adobe FreeHand
FutureSplash	spl	Формат распространяемой графики FutureSplash — предшественника Flash

Другие форматы

Здесь мы опишем еще два "не совсем" графических формата, о которых вам нужно знать, если вы собираетесь заняться интернет-графикой. Сразу следует предупредить, что Flash их не поддерживает. Поэтому, если вы захотите поместить в изображение Flash графику, сохраненную в одном из этих форматов, попробуйте воспользоваться буфером обмена Windows.

PDF

Формат PDF (Portable Document Format — формат переносимых документов) был разработан фирмой Adobe для создания переносимых платформно-независимых электронных документов. Такие документы могут содержать,

кроме форматированного текста, различную векторную и растровую графику, разбиваться на страницы, печататься на принтере или просматриваться на экране компьютера. Помимо этого, документы в этом формате имеют очень малый размер (используется сжатие, причем каждый вид графики сжимается по самому подходящему для него алгоритму), таким образом, их можно распространять через Интернет.

Документы этого формата сохраняются в файлах с расширением pdf. Они создаются с помощью пакета Adobe Acrobat, а читаются — с помощью программы чтения Adobe Acrobat Reader, распространяемой бесплатно. По названию этих двух программ формат PDF получил свое второе название — формат документов Acrobat.

Формат PDF получил огромную популярность для распространения электронных документов с богатым форматированием и графикой. Фактически, этот формат теперь стоит на втором месте, после HTML, по распространенности. Кроме Adobe Acrobat, создание документов PDF поддерживают множество других текстовых и графических пакетов. А Adobe Acrobat Reader — одна из популярнейших в мире программ.

VRML

Формат, точнее, язык, VRML (Virtual Reality Modeling Language — язык моделирования виртуальной реальности) был разработан группой независимых разработчиков. Он служит для создания так называемых виртуальных миров и распространения их через Интернет. Описание такого виртуального мира представляет собой текстовый файл с расширением vrm или vrm1, содержащий набор команд на языке VRML. Такой файл может быть "проигран" с помощью особых программ — "проигрывателей" VRML; самым распространенным из них является Cortona VRML Client фирмы Parallel Graphics.

Хоть язык VRML и поддерживающее его программное обеспечение, предназначенное для создания и "проигрывания" виртуальных миров были широко разрекламированы в свое время, распространение он получил очень ограниченное. Связано это с тем, что мощностей современных компьютеров пока что не хватает для создания чего-то более сложного, чем странные геометрические фигуры из двух сфер и конуса. К тому же, виртуальная реальность, похоже, потеряла для простых людей свою притягательность. Поэтому будущее VRML так же туманно, как и настоящее...

Использование графических форматов

Перечислив и кратко описав каждый из наиболее распространенных на текущий момент времени графических форматов, давайте выясним, какой формат когда используется.

Итак, если вы создаете графику с целью продолжить работу над ней в другом графическом пакете (Adobe FreeHand, PhotoShop, CorelDRAW!), то смело сохраняйте ее в "родном" для соответствующего пакета формате. Если нужный формат не поддерживается, то можно воспользоваться одним из универсальных форматов. Так, растровую графику можете сохранить в формате TIFF, а векторную — в Adobe Illustrator или Encapsulated PostScript. Эти форматы должна "взять" любая достаточно мощная графическая программа. Если же вам нужно передать растровое графическое изображение пользователю Apple Macintosh, то вы можете сохранить его в формате PICT.

При необходимости сохранить полутоновое графическое изображение для распространения, то нет ничего лучше, чем формат JPEG. Такой файл можно просмотреть в обычном Web-обозревателе. Отрегулируйте коэффициент сжатия так, чтобы достичь разумного компромисса между качеством картинки и размером файла.

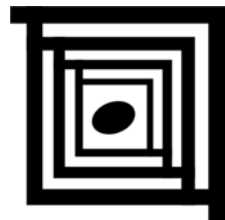
Здесь, правда, есть еще одно решение. Если вы хотите, чтобы вашу графику можно было использовать в качестве "обоев", лучше сохранить ее в формате BMP. Если же вы сохраните ее в формате JPEG, то потребителю вашего "обойного" искусства придется либо самому преобразовывать его в BMP, либо включать Активный рабочий стол, пожирающий уйму системных ресурсов. Файлы BMP, как вы помните, обрабатываются самым ядром Windows, поэтому "нагружают" систему несравнимо меньше.

В случае же векторной графики, на взгляд автора, ничего нет лучше "родного" формата Shockwave/Flash. Экспортируйте ваше творение в файл с расширением swf, и можете раздавать жаждающим. Все современные программы Web-обозревателей имеют в своем составе модуль проигрывателя Shockwave/Flash. Но даже если кто-то из потребителей вашего искусства не имеет такого модуля, вы всегда сможете переписать ему дистрибутив этого модуля — он поставляется в составе пакета Macromedia Flash.

Если вы занимаетесь Web-дизайном и вам нужно создать для своей страницы небольшие графические элементы оформления, то сохраните их в формате GIF. Такие файлы будут иметь минимальный размер. Если же вы озабочены проблемами с авторскими правами, поднятыми вокруг этого формата, в качестве альтернативы выбирайте формат PNG. Есть, правда, еще одна альтернатива — создать все графическое оформление Web-страницы на основе Flash, но это уже на любителя.

Когда же вам нужно выложить в Сеть высококачественные полутоновые изображения, смело выбирайте формат JPEG. К счастью, это открытый формат, и денег за его использование пока что брать никто не собирается. А замены ему — увы! — в обозримом будущем не предвидится.

Глава 5



Рисование

Познакомившись с двумя фундаментальными разновидностями компьютерной графики, выяснив их преимущества и недостатки, рассмотрев современные форматы записи графических данных в файл, можно приступить непосредственно к рисованию. Что мы и сделаем в этой главе.

Кратко напомним, что такое векторная графика. Векторное графическое изображение состоит из графических примитивов: простейших составных частей, каждая из которых может быть описана особой формулой. К графическим примитивам относятся прямые и кривые линии, простейшие геометрические фигуры (эллипсы, прямоугольники и пр.), текстовые надписи и импортированные растровые изображения. (Формально, если векторное изображение содержит в своем составе достаточно большое количество растровых изображений, говорят о гибридной графике, но мы не будем играть в терминологические игры, чтобы не запутаться.) Для каждого примитива в графическом файле сохраняются его тип и параметры описывающей его формулы: местоположение, размеры, цвет, кривизна, радиусы и т. п. Графическая программа, считав с диска такой файл, подставляет в соответствующие формулы нужные параметры, и мы получаем на экране или принтере исходное изображение.

Процесс преобразования векторной графики в растровую называется растеризацией. В частности, программа векторного графического редактора выполняет растеризацию всякий раз, когда выводит векторное изображение на экран или принтер — устройства изначально растровые. Обратный процесс — преобразование растровой графики в векторный вид — называется векторизацией. Векторизация растрового изображения чревата большими потерями в качестве, особенно плохо векторизуется высококачественное полутонное фотографическое изображение.

Вот и все, воспоминания закончились. Подробнее о векторной и растровой графике, графических форматах и их поддержке Flash см. *главу 4*. А нам пора за работу.

В этой главе мы изучим простейшие операции рисования векторных изображений. Эти операции включают в себя создание графических примити-

вов и элементарные операции по их изменению. Мы научимся пользоваться большинством этих инструментов, расположенных в инструментарии Flash, для рисования и правки уже нарисованного.

Но довольно пустой болтовни. Запустите Macromedia Flash. Просмотрите *часть I* этой книги, если забыли, что есть что. И взгляните на инструментарий, расположенный вдоль левого края родительского окна программы. Основная работа в ближайшее время будет вестись именно с ним.

Использование инструментов рисования

Начнем мы, конечно же, с инструментов рисования. А потом выясним, с помощью каких инструментов можно исправить уже нарисованное. Итак...

Но прежде всего поговорим о выборе цвета линии и заливки. Более подробно средства управления цветом будут рассмотрены в *главе 6*, но базовые понятия нужно дать уже сейчас.

Базовые средства управления цветом

Для выбора цвета линии и заливки создаваемых графических примитивов служит небольшая область в середине инструментария, носящая заголовок **Colors**. Она приведена на рис. 5.1. Рассмотрим ее подробнее.



Рис. 5.1. Область **Colors** управления цветом инструментария

Прежде всего, заметим, что в нижней части этой области находятся три небольшие кнопки. Эти кнопки включают особые цветовые режимы, мы рассмотрим их в *главе 6*. В данный же момент нас больше интересуют два селектора цвета, занимающие большую часть области **Colors**. С их помощью и задаются цвета графики. (Правда, кроме них, возможность выбора цвета предоставляют различные панели; реализуется это с помощью все тех же селекторов цвета.)

Как видите, селекторов цветов в области **Colors** два. Верхний из них устанавливает цвет линии — об этом говорит значок карандаша слева от селектора. Нижний задает цвет заливки, что подтверждает значок опрокинутого ведра.

Имейте в виду, что селекторы цветов, расположенные в области **Colors** инструментария, меняют цвета только вновь создаваемой графики. Поэтому, если вы сменили цвет линии с черного на синий, следующие начерченные вами линии будут синими, но начерченные ранее останутся черными. Как изменить цвет уже нарисованных примитивов будет описано далее в этой главе.

"Линия"

Как вы уже знаете, этот инструмент служит для проведения прямых линий. Чтобы выбрать его, щелкните в инструментарии кнопку, показанную на рис. 5.2, или нажмите на клавиатуре клавишу <N>. После этого курсор мыши примет вид небольшого крестика, — это значит, что теперь вы можете проводить мышью прямые линии.



Рис. 5.2. Кнопка включения инструмента "линия"

Чтобы провести на рабочем листе прямую линию с помощью инструмента "линия", сделайте следующее. (Имеется в виду, что вы уже выбрали этот инструмент.) Поместите курсор мыши в том месте, где у вас будет начало линии, и нажмите левую кнопку мыши. После этого, не отпуская эту кнопку, протащите мышь в то место, где у вас будет конец линии. Пока вы буксируете мышь, от начала линии за ее курсором будет тянуться "резиновая" линия. Чтобы поставить вторую точку — конец прямой — и тем самым нарисовать ее, просто отпустите левую кнопку мыши в нужной точке. Все, прямая линия создана.

Вы можете проводить таким образом линии любой длины и под любым углом. Если же вам нужно провести линию под углом, кратным 45° , то во время проведения линии удерживайте нажатой клавишу <Shift>. В этом случае проводимая линия будет автоматически "подгоняться" под подходящий угол.

Если включен модификатор "притягивание" различных частей вашей графики друг к другу, при рисовании линий это учитывается. При буксировке мыши, если какой-либо из концов линии окажется достаточно близко от другой линии (возможно, принадлежащей другому графическому примитиву), правее и ниже курсора мыши появится небольшой кружок, сигнализирующий об этом. Если теперь отпустить кнопку мыши, завершив рисование линии, ее конец будет "приклеен" к уже нарисованной линии, к которой он приблизился.

Вы можете изменять цвет линии с помощью уже знакомого вам селектора цвета линии. Селектор цвета заливки никакого влияния на рисуемые прямые линии не оказывает, так как у линий нет заливки.

Вот и все об инструменте "линия". Теперь рассмотрим два следующих инструмента, позволяющие рисовать другие простейшие графические примитивы: "прямоугольник" и "эллипс".

"Прямоугольник" и "эллипс"

Оба этих инструмента очень похожи. Разница проявляется лишь в результате их применения. Поэтому мы и описываем их здесь вместе.

Чтобы выбрать инструмент "прямоугольник", щелкните в инструментарии кнопку, показанную на рис. 5.3, или нажмите клавишу <R>. Для выбора инструмента "эллипс" щелкните кнопку, показанную на рис. 5.4, или нажмите клавишу <O>. В обоих этих случаях курсор мыши примет вид небольшого крестика — сигнал, что нужный инструмент выбран.



Рис. 5.3. Кнопка включения инструмента "прямоугольник"

Рис. 5.4. Кнопка включения инструмента "эллипс"

Прямоугольник создается следующим образом. Поместите курсор мыши в том месте, где у вас будет его левый верхний угол, и нажмите левую кнопку мыши. После этого, не отпуская эту кнопку, протащите мышь в то место, где у вас будет находиться правый нижний угол прямоугольника. Пока вы буксируете мышь, Flash отображает "резиновый" прямоугольник, так что вы всегда будете видеть, что получается. Переместив курсор в нужную точку, отпустите левую кнопку мыши — и прямоугольник будет нарисован.

Эллипс создается аналогичным способом. Вы выбираете точку, соответствующую левому верхнему углу воображаемого прямоугольника, в который будет вписан эллипс, и буксируете мышь в точку правого нижнего угла этого прямоугольника. В процессе перетаскивания мыши Flash будет показывать вам "резиновый" эллипс.

Вы можете рисовать таким образом прямоугольники и эллипсы любой формы. Если же вам нужно нарисовать правильный квадрат или правильную окружность, то во время рисования удерживайте нажатой клавишу <Shift>.

Каждый нарисованный вами прямоугольник и эллипс будет иметь заливку. Цвет заливки задается нижним селектором цвета в области **Colors** инструментария. А цвет линий, как вы помните, задается с помощью верхнего селектора цвета.

Прямоугольники, создаваемые с помощью одноименного инструмента, могут иметь скругленные углы. Для этого служит модификатор "скругленные углы". Включается он нажатием небольшой кнопки, появляющейся в области **Options** инструментария, если выбран инструмент "прямоугольник". Эта кнопка показана на рис. 5.5.



Рис. 5.5. Кнопка модификатора "скругленные углы"

При нажатии кнопки модификатора "скругленные углы" на экране появляется небольшое диалоговое окно **Rectangle Settings**, показанное на рис. 5.6. В единственном поле ввода **Corner Radius** следует задать радиус кривизны углов в точках. После этого нужно нажать кнопку **ОК** для подтверждения ввода данных или **Cancel** — для отмены. Если был введен радиус кривизны углов, отличный от нуля, и нажата кнопка **ОК**, все следующие нарисованные прямоугольники будут иметь скругленные углы. Чтобы вернуть прямоугольникам обычные, прямоугольные углы, нужно ввести в поле ввода **Corner Radius** ноль и не забыть нажать кнопку **ОК**.

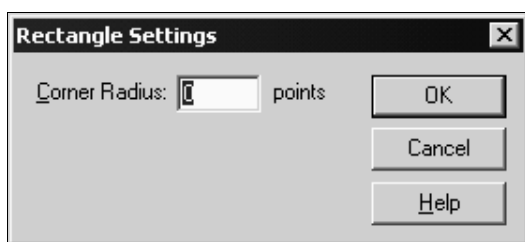


Рис. 5.6. Диалоговое окно **Rectangle Settings**

Есть еще один способ создания скругленных углов у прямоугольников. Для этого во время буксировки мыши при рисовании очередного прямоугольника нужно нажимать клавиши $\langle \uparrow \rangle$ и $\langle \downarrow \rangle$. Клавиша $\langle \uparrow \rangle$ уменьшает радиус кривизны углов, а клавиша $\langle \downarrow \rangle$ — увеличивает. Изменение радиуса отображается прямо на "резиновом" прямоугольнике, так что вы всегда сможете вовремя остановиться.

Итак, с простейшими геометрическими фигурами мы покончили. Перейдем к более сложным примитивам.

"Карандаш"

Этот инструмент позволяет рисовать линии любой (как говорят профессиональные компьютерные художники — *свободной*) формы. Работает он так же, как обычный карандаш, отчего и получил свое название. Вы нажимаете левую кнопку мыши и, не отпуская ее, рисуете линию, какая вам нужна. Пример такой линии показан на рис. 5.7.

Чтобы выбрать инструмент "карандаш", щелкните в инструментарии соответствующую кнопку (рис. 5.8) или нажмите клавишу $\langle Y \rangle$. Курсор мыши при этом примет вид карандаша.

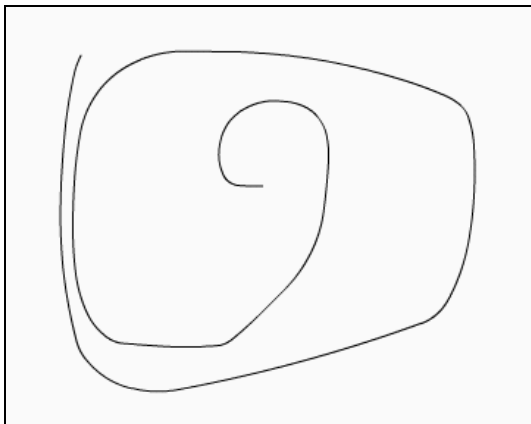


Рис. 5.7. Линия, проведенная "карандашом"

Чтобы нарисовать с его помощью линию, поместите курсор мыши в точку, где она должна начинаться, и нажмите левую кнопку мыши. После этого, не отпуская левой кнопки, двигайте мышь по нужной траектории и, когда курсор мыши окажется в точке, где должен быть конец линии, отпустите кнопку мыши. Flash будет рисовать проводимую вами линию прямо во время движения мыши. Если вам нужно провести строго прямую горизонтальную или вертикальную линию, то во время рисования удерживайте нажатой клавишу <Shift>.

Вы можете задавать цвет линии, пользуясь верхним селектором цвета, расположенным в области **Colors** инструментария. Нижний селектор цвета никакого влияния на проводимые "карандашом" линии не оказывает.

Вы, наверно, заметили, что Flash автоматически сглаживает проведенную вами линию. Эта замечательная особенность, введенная программистами фирмы Macromedia, позволяет вам с помощью инструмента "карандаш" в любом случае проводить прямые линии. Как вы знаете, человеческая рука — не очень точный инструмент, и Flash это учитывает, сглаживая мелкие неровности проводимых вами линий. Однако, иногда такая "услужливость" может навредить, например, если вам нужно нарисовать что-то очень точно, со всеми неровностями и шероховатостями. Для регулирования степени сглаживания служит модификатор "сглаживание линий", представляющий собой кнопку, расположенную в области **Options** инструментария (рис. 5.9).

Вы можете заметить, что в правом нижнем углу этой кнопки имеется небольшая черная стрелка, направленная направо вниз. При нажатии этой кнопки на экране появится меню, предлагающее пользователю на выбор несколько вариантов. Меню, появляющееся при нажатии кнопки модификатора "сглаживание линий", показано на рис. 5.10. Такие кнопки с меню часто применяются во Flash.



Рис. 5.8. Кнопка включения инструмента "карандаш"



Рис. 5.9. Кнопка модификатора "сглаживание линий"

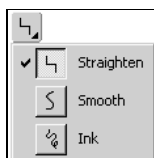


Рис. 5.10. Меню модификатора "сглаживание линий"

Это меню, как видите, имеет три пункта, задающие режим сглаживания:

- ☐ **Straighten** — самое сильное сглаживание, готовые линии получаются прямыми или изломанными, а приближенные подобия геометрических фигур преобразуются в точные фигуры;
- ☐ **Smooth** — менее сильное сглаживание, готовые линии почти всегда получаются кривыми;
- ☐ **Ink** — сглаживание практически отсутствует.

Если вам нужно нарисовать достаточно точную фигуру из прямых линий, выбирайте сглаживание **Straighten**. Если ваша геометрическая фигура будет содержать кривые линии, выбирайте **Smooth**. Для точного рисования сложных кривых следует выбрать режим **Ink**.

"Перо"

Этот мощнейший инструмент поможет вам рисовать как прямые, так и кривые линии, точно указывая начальную и конечную точки и радиус кривизны. Кривые линии, рисуемые с помощью этого инструмента, называют также *кривыми Безье* по имени математика, выведшего формулу для описания этих кривых. (Прямую линию можно считать вырожденным случаем кривой Безье.) Кроме того, "перо" предоставляет возможность создания ломаных линий, состоящих из множества прямых или кривых отрезков, и весьма сложных геометрических фигур.

Чтобы выбрать инструмент "перо", щелкните в инструментарии соответствующую кнопку (рис. 5.11) или нажмите клавишу <P>. Курсор мыши при этом примет вид чертежного рейсфедера.



Рис. 5.11. Кнопка включения инструмента "перо"

Рисование прямых с помощью "пера" выполняется следующим образом. Поместите курсор мыши в точку, где должно быть начало прямой, и щелкните левой кнопкой мыши. (Имейте в виду — нужно именно щелкнуть!) В этом месте на листе появится небольшая полая точка. Далее поместите курсор в конечную точку прямой и снова щелкните мышью. Прямая будет тотчас проведена.

Если вы продолжите щелкать мышью в разных местах на рабочем листе, то сможете построить ломаную линию, аналогичную показанной на рис. 5.12. Чтобы завершить рисование ломаной, при создании последней ее точки сделайте двойной щелчок мышью вместо одинарного. Вы также можете сразу после последнего щелчка выбрать другой инструмент или щелкнуть где-либо на пустом пространстве рабочего листа, удерживая нажатой клавишу <Ctrl>.

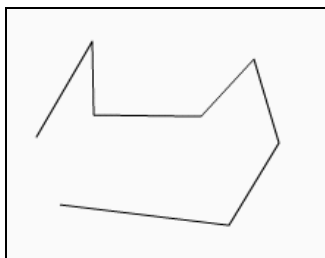


Рис. 5.12. Ломаная линия, построенная с помощью инструмента "перо"

Существует возможность проводить прямые линии под углом, кратным 45° . Для этого вам нужно при втором щелчке удерживать нажатой клавишу <Shift>.

При создании ломаной вы можете замкнуть ее, для чего подведите курсор мыши к одной из уже созданных ее точек так, чтобы ниже и правее его появился небольшой кружок, и щелкните левой кнопкой мыши. В этом случае Flash автоматически создаст замкнутую область и поместит в нее заливку, как показано на рис. 5.13.

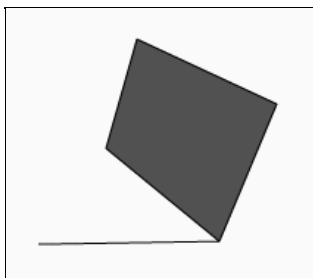


Рис. 5.13. Ломаная, содержащая замкнутый участок с заливкой

Вы уже, наверно, заметили, что при использовании инструмента "перо" линии проводятся не тем цветом, что выбран как цвет линии, а синим. Дело в том, что Flash помечает этим цветом фигуры, рисование которых, по его мнению, не закончено. Чтобы увидеть созданные вами фигуры такими, какими они должны быть, либо выберите другой инструмент в инструментарии, либо выберите пункт **Deselect All** в меню **Edit** или контекстном меню листа.

Как с помощью "пера" создаются прямые линии мы рассмотрели. Рассмотрим теперь, как создавать кривые.

Начальная точка кривой линии создается точно так же, как начальная точка прямой. После этого выберите место на рабочем листе, где будет находиться конечная точка кривой, и щелкните левой кнопкой мыши, но не отпускайте ее. Далее переместите курсор мыши в каком-либо направлении, лучше перпендикулярном направлению получающейся прямой линии. Вы увидите, как при этом меняется форма создаваемой кривой (рис. 5.14).

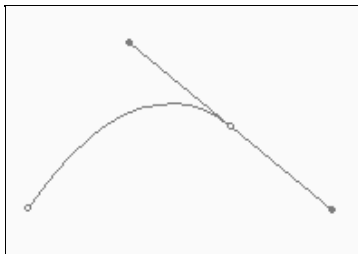


Рис. 5.14. Рисование кривой с помощью инструмента "перо"

Вы можете сделать немного по-другому. Выберите на листе точку, где должно находиться начало кривой, поставьте в нее курсор мыши, нажмите левую кнопку и, не отпуская ее, переместите мышь. После этого отпустите кнопку мыши, переместите курсор на место, где должен быть конец кривой, и щелкните левой кнопкой. Вы получите ту же самую кривую. Ну, конечно, не в точности такую же, но аналогичную. И, наконец, можно совместить оба этих подхода. Поэкспериментируйте и посмотрите на получившиеся результаты — это лучший способ понять, как работает инструмент "перо".

Главное правило таково: перемещайте мышь в том направлении, в каком создаваемая кривая должна быть вогнута. Если вы будете удерживать при этом нажатой клавишу <Shift>, курсор мыши будет перемещаться под ближайшим углом, кратным 45°. Касательная — прямая линия, заканчивающаяся темными точками, рисуемая Flash при создании кривой, — поможет вам выбрать ее форму.

Основное условие при создании кривых Безье — поэкспериментировать с инструментом "перо". В конце концов, практика в некоторых случаях может дать несравнимо больше теории. Перефразируя известную пословицу, можно ска-

зять, что лучше один раз что-то сделать самостоятельно, чем сто раз услышать, как это делается. Поэтому мы вас призываем: экспериментируйте!

Все, что мы говорили о прямых, распространяется и на кривые. Посмотрите на рис. 5.15, там показаны некоторые примеры сложных геометрических фигур, созданных из кривых Безье с помощью "пера". Попробуйте повторить их сами и создать аналогичные.

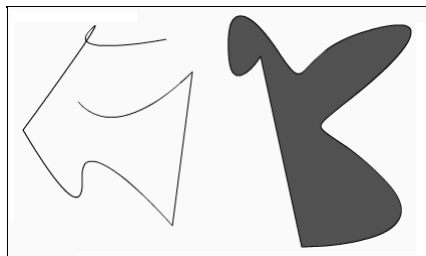


Рис. 5.15. Примеры сложных геометрических фигур, созданных с помощью инструмента "перо"

Остается добавить, что вы можете задавать цвет линии, пользуясь селектором цвета, расположенным в области **Colors** панели инструментов.

"Кисть"

Этот инструмент ведет себя аналогично реальной кисти. С его помощью вы можете наносить мазки разной формы и толщины. Пример таких мазков показан на рис. 5.16.

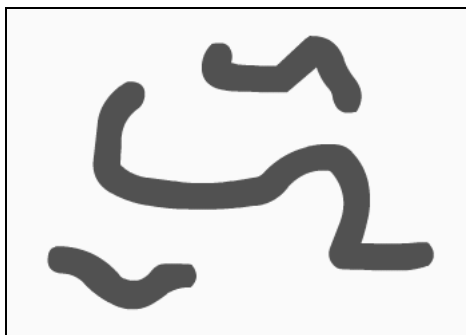


Рис. 5.16. Мазки, проведенные "кистью"

Чтобы выбрать инструмент "кисть", щелкните в инструментарии соответствующую кнопку (рис. 5.17) или нажмите клавишу . Курсор мыши при этом может принимать разные формы, в зависимости от настроек, заданных с помощью модификаторов (см. ниже).



Рис. 5.17. Кнопка включения инструмента "кисть"

Инструмент "кисть" используется аналогично "карандашу". Вы ставите курсор мыши в точку, где должно быть начало мазка, нажимаете левую кнопку, двигаете мышью, рисуя сам мазок, и, в конце концов, заканчиваете рисование, отпустив левую кнопку мыши.

Если вам нужно провести строго прямой горизонтальный или вертикальный мазок, то во время рисования удерживайте нажатой клавишу <Shift>. Это правило справедливо практически для всех инструментов Flash.

Цвет мазка, как вы уже догадались, выбирается с помощью нижнего селектора цвета, расположенного в области **Colors** инструментария. Это вполне логично, так как "кисть" часто используется для создания заливок.

Кроме цвета мазка, Flash предоставляет возможность выбора формы и размера кисти. Для этого служат модификаторы "форма кисти" и "размер кисти", расположенные, как и все другие модификаторы, в области **Options** инструментария. Эти модификаторы представляют собой обычные раскрывающиеся списки, в которых в виде графических изображений представлены все доступные во Flash формы и размеры "кисти". Модификатор "форма кисти" в раскрытом виде показан на рис. 5.18, а модификатор "размер кисти" — на рис. 5.19. Как видите, возможности выбора достаточно велики.

Как только вы выберете с помощью вышеупомянутых модификаторов форму и (или) размер "кисти", форма и размер курсора мыши изменятся. Таким образом, вы всегда будете знать, какой "кистью" вы работаете.



Рис. 5.18. Модификатор "форма кисти"

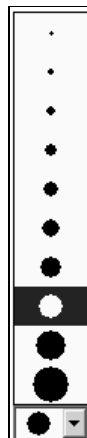


Рис. 5.19. Модификатор "размер кисти"

Еще один модификатор позволяет задать режим работы "кисти", т. е. определяет, будет ли "кисть" закрашивать линии или ограничиваться лишь заливками. Этот модификатор имеет вид кнопки с меню (рис. 5.20).

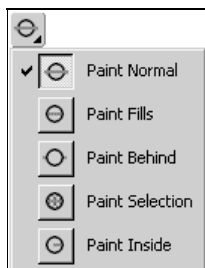


Рис. 5.20. Кнопка модификатора "режим закраски" с раскрытым меню

В меню модификатора "режим закраски" доступно пять пунктов:

- ☐ **Paint Normal** — обычное рисование, когда закрашиваются любые линии и любая заливка (рис. 5.21);
- ☐ **Paint Fills** — закрашивается только заливка и пустые области, линии не закрашиваются (рис. 5.22);
- ☐ **Paint Behind** — закрашиваются только пустые области, линии и заливки не закрашиваются (рис. 5.23);
- ☐ **Paint Selection** — закрашивается только выделенная с помощью "стрелки" заливка, невыделенные заливки и любые линии остаются незакрашенными (рис. 5.24);
- ☐ **Paint Inside** — закрашивается только заливка, находящаяся в пределах или за пределами замкнутого контура. То, что находится по другую сторону линий, составляющих этот контур, не закрашивается. Линии также не закрашиваются (рис. 5.25).

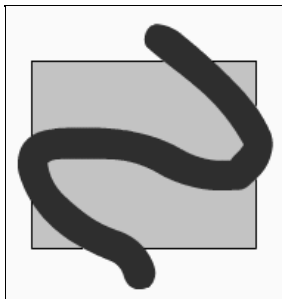


Рис. 5.21. Результат применения режима закраски **Paint Normal**

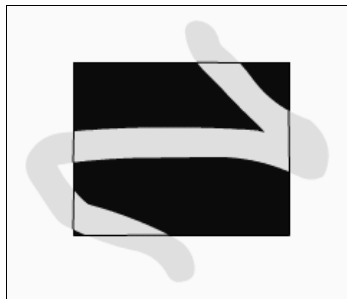


Рис. 5.22. Результат применения режима закраски **Paint Fills** (обратите внимание — линии остались незакрашенными)

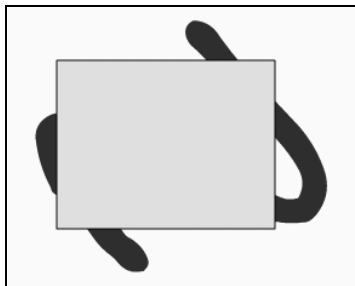


Рис. 5.23. Результат применения режима закрашки **Paint Behind**

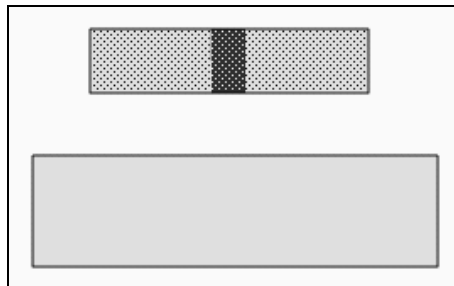


Рис. 5.24. Результат применения режима закрашки **Paint Selection** (заливка верхнего прямоугольника была выделена с помощью инструмента "стрелка", а заливка нижнего — нет)

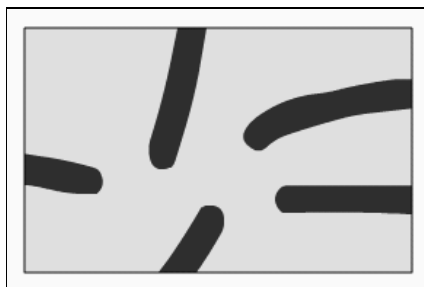


Рис. 5.25. Результат применения режима закрашки **Paint Inside** (начало каждого мазка находилось внутри прямоугольника, а конец — вне его)

"Ведро с краской"

Этот инструмент служит для создания заливок. *Заливкой* называется особый графический примитив, заполняющий замкнутый контур, образованный линиями. Часто Flash создает заливки автоматически, например, когда вы рисуете, используя инструменты "прямоугольник", "эллипс" и "перо". Если же вы применяете другие инструменты для создания графики, о заливках вам придется позаботиться самостоятельно.

Инструмент "ведро с краской" ведет себя аналогично реальному ведру с краской: если "опрокинуть" его над замкнутым контуром, краска "разольется" и заполнит ее. Если же контур имеет "прореху", например, вы забыли его замкнуть, заливка создана не будет. Как видите, Flash не допускает, чтобы "краска" "вытекла" наружу.

Чтобы выбрать инструмент "ведро с краской", щелкните в инструментарии соответствующую кнопку (рис. 5.26) или нажмите клавишу <K>. Курсор мыши при этом примет вид опрокинутого ведра с краской.



Рис. 5.26. Кнопка включения инструмента "ведро с краской"

Использовать этот инструмент очень просто. Выберите нужную замкнутую фигуру, проследите за тем, чтобы она была действительно замкнута (иначе заливка не будет создана), укажите на нее курсором мыши и щелкните левой кнопкой. Flash тотчас создаст заливку, заполняющую эту фигуру. Пример такой заливки показан на рис. 5.27.

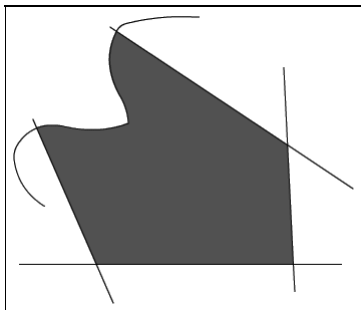


Рис. 5.27. Пример заливки, созданной при помощи инструмента "ведро с краской"

Как вы уже догадались, выбор цвета заливки производится с помощью нижнего селектора цвета в области **Colors** инструментария. Цвет, заданный с помощью верхнего селектора, в этом случае роли не играет. Кроме того, возможно создание градиентных заливок, в которых цвет плавно изменяется, или использование в качестве заливок растровых графических изображений. О градиентных заливках будет рассказано в *главе 6*.

Если контур нарисованной вами фигуры в некоторых местах не замкнут, не стоит огорчаться. Flash автоматически устранил эти огрехи, если, конечно, они действительно невелики. Если же разрыв в контуре больше какой-то определенной величины, то Flash вообще не будет создавать заливку. Размер этого разрыва, которого "не замечает" Flash, можно задать с помощью модификатора "размер разрыва". Этот модификатор расположен в области **Options** инструментария и в раскрытом виде показан на рис. 5.28.

Меню этого модификатора предлагает для выбора четыре пункта:

- ☐ **Don't Close Gaps** — весь контур должен быть замкнут, иначе заливка создана не будет;
- ☐ **Close Small Gaps** — Flash закрывает только маленькие разрывы;
- ☐ **Close Medium Gaps** — Flash закрывает промежутки средних размеров;
- ☐ **Close Large Gaps** — Flash закрывает большие разрывы.

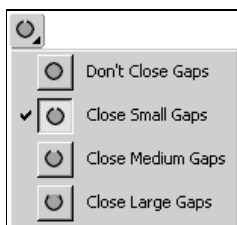


Рис. 5.28. Модификатор "размер разрыва"

И еще. Если вы попытаетесь закрасить сам рабочий лист, выбрав "ведро с краской" и щелкнув по пустому пространству листа, заливка создана не будет. Flash может создать заливку только внутри замкнутого контура — помните это. Чтобы изменить цвет фона листа, воспользуйтесь диалоговым окном **Document Properties**. Работа с ним была подробно описана в *главе 2*.

Правка графики

Итак, создавать графику мы научились. Теперь давайте выясним, как ее редактировать.

Под редактированием графики мы будем понимать изменение ее формы, а также ее полное или частичное удаление. Как добавить новые линии или фигуры вы уже должны знать — воспользуйтесь для этого одним из рассмотренных нами инструментов рисования. Кроме того, редактирование графики также подразумевает изменение ее цвета, но об этом мы подробнее поговорим в *главе 6*.

Для изменения графики Flash предоставляет четыре мощных инструмента: "стрелка выделения", "дополнительная стрелка выделения", "лассо" и "ластик". Первые два инструмента предназначены для выделения и правки элементов изображения, третий — для "вырезания" из фигур "лоскутков" произвольной формы, а четвертый — полного или частичного удаления фигур или их частей. Пользуясь этими четырьмя инструментами, вы сможете проделывать со своей графикой все что угодно.

Но прежде, чем править графику, ее нужно выделить. Давайте выясним, как это делается.

Выделение графики

Для *выделения* (или, как еще говорят, выбора) фрагментов графики для редактирования служит инструмент "стрелка выделения" (или просто "стрелка"). Щелкните кнопку, показанную на рис. 5.29, в инструментарии или нажмите клавишу <V> на клавиатуре. Курсор мыши примет вид не-большой черной стрелки.



Рис. 5.29. Инструмент "стрелка выделения"

Как и в любых других аналогичных Windows-приложениях, чтобы выделить что-то в рабочем окне программы, вам нужно щелкнуть по нему мышью. После этого выбранный фрагмент графики станет выделенным, о чем недвусмысленно скажет заполнение из черных точек (рис. 5.30). Однако Flash вносит в этот несложный процесс свои коррективы.

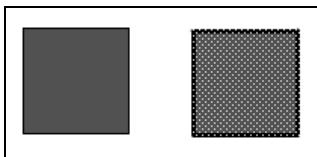


Рис. 5.30. Слева — невыделенный прямоугольник, справа — выделенный

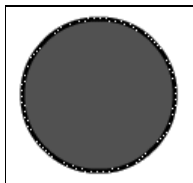
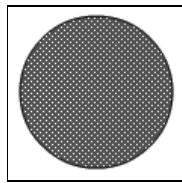
Начнем с того, что геометрическая фигура, нарисованная с помощью инструментов Flash, фактически может состоять из двух частей. Первая часть — это контур, который образуют внешние линии фигуры. Контур может быть замкнутым или незамкнутым. Если контур замкнут, в составе фигуры может присутствовать и вторая часть — заливка. (Выше уже говорилось, что контур и заливка — совершенно разные виды графических примитивов Flash.) Например, из контура и заливки состоят нарисованные вами прямоугольники и эллипсы. Конечно, бывают фигуры, состоящие из одного контура, например, линии, но разговор сейчас не о них.

Вышесказанное означает, что можно выделить отдельно контур какой-либо фигуры и произвести с ним какие-то действия, не затрагивая заливку. И наоборот, выделив заливку этой же или другой фигуры и что-то с ней сделав, мы никак не затронем контур. Более того, мы даже можем отделить их друг от друга, превратив фактически в две абсолютно независимые фигуры.

Чтобы выделить контур фигуры, щелкните по нему мышью. (Предполагается, что вы уже выбрали инструмент "стрелка выделения".) Конечно, это довольно трудно, особенно если он выполнен тонкими линиями, но вы можете изменить масштаб отображения. Круг с выделенным контуром показан на рис. 5.31.

Выделить заливку проще, как правило, она значительно объемистее любых линий, и вы уж точно мимо нее не промахнетесь. Просто щелкните по ней, результат показан на рис. 5.32.

Но что делать, если нужно выделить и контур, и заливку? Есть два способа сделать это. Вы сами можете выбрать, какой из них для вас удобнее.

**Рис. 5.31.** Круг с выделенным контуром**Рис. 5.32.** Круг с выделенной заливкой

Во-первых, практически любое Windows-приложение, работающее с графикой, предоставляет возможность *выделения перетаскиванием* мыши. Поместите курсор мыши в какую-либо точку на листе Flash, нажмите левую кнопку и, не отпуская ее, протаскивайте мышью так, чтобы захватить в прямоугольник выделения все нужные вам фрагменты графики. Таким образом вы можете выделять одновременно сколько угодно графических фрагментов.

Во-вторых, вы можете воспользоваться одной уникальной возможностью Flash. Поместите курсор мыши на контур или заливку (лучше на заливку) и выполните двойной щелчок левой кнопкой мыши. Этот способ применяется для выделения определенного фрагмента графики и всех примыкающих к нему фрагментов, будь то линии или заливки.

Пример фигуры с выделенными контуром и заливкой показан на рис. 5.30 справа.

Вы только что познакомились с первым способом выделения сразу нескольких графических фрагментов. Он подходит в тех случаях, когда нужные фрагменты находятся близко друг к другу. А если они разбросаны по всему листу? В этом случае воспользуйтесь вторым способом.

Второй способ заключается в следующем. (Кстати, его тоже предоставляет большинство Windows-приложений, работающих с графикой.) Выделите первый фрагмент, щелкнув по нему мышью. После этого щелкайте по остальным фрагментам по очереди, удерживая нажатой клавишу <Shift>. Выделив все фрагменты, можете ее отпустить.

Внимание!

Если на вкладке **General** диалогового окна **Preferences** был отключен флажок **Shift Select**, вам не нужно для выделения нескольких графических фрагментов удерживать нажатой клавишу <Shift>. Просто щелкайте по ним мышью, и они будут выделяться. Чтобы снять выделение, щелкните по пустому пространству рабочего листа.

Если вы во время такого выделения (клавиша <Shift> еще нажата) щелкнете уже выделенный фрагмент, он перестанет быть выделенным. Вы можете воспользоваться этим, выделив большую группу фрагментов путем перетаскивания мыши, нажав <Shift> и щелчками мыши сняв выделение с ненужных фрагментов. Опытные компьютерщики поступают так очень часто.

Как бы то ни было, вам лучше всего попрактиковаться с выделением различных фрагментов графики несколькими способами. Нарисуйте что-нибудь и попробуйте все вышеописанные способы выделения графики на практике. Приобретенные навыки очень помогут вам в дальнейшем.

И напоследок остается сказать совсем немного. Если вы хотите выделить все, что находится у вас на рабочем листе, выберите пункт **Select All** меню **Edit** или контекстного меню листа или нажмите комбинацию клавиш <Ctrl>+<A>. Убрать же выделение со всех элементов вам позволит выбор пункта **Deselect All** меню **Edit** или контекстного меню листа или комбинация клавиш <Ctrl>+<Shift>+<A>. Снять выделение можно также, щелкнув по пустому месту, не занятому графикой.

Иногда нужно скрыть выделение графических фрагментов, т. е. сделать так, чтобы штриховка, которой помечаются выделенные фрагменты, не показывалась на экране. Для этого выберите пункт-выключатель **Hide Edges** меню **View** или нажмите комбинацию клавиш <Ctrl>+<H>. Впоследствии вы можете снова выбрать этот пункт или нажать ту же комбинацию клавиш, чтобы вновь вывести на экран штриховку.

Фрагментация и слияние графики

Прежде чем мы приступим к рассмотрению способов правки уже созданной графики, поговорим еще об одной особенности Flash. Это так называемая *фрагментация* графики. Суть ее в том, что Flash-графика в определенных случаях "разваливается" на независимые фрагменты. Иногда это бывает полезно, иногда не очень.

Фрагментации подвержены сложные фигуры, состоящие из множества графических примитивов, нарисованных разным цветом. Это основное правило фрагментации. Вы можете также разбить на фрагменты и монолитные фигуры, но об этом мы поговорим чуть ниже.

Чтобы лучше понять, что такое фрагментация, приведем небольшой пример. Нарисуем на листе Flash несложную фигуру, показанную на рис. 5.33. На этой фигуре мы и рассмотрим фрагментацию Flash-графики как явление.

Нарисовав фигуру, выберем инструмент "стрелка выделения" и щелкнем на пустом месте, чтобы снять любое выделение. После этого выделим какой-либо из концов линии, выступающих за пределы перечеркнутого круга. И получим то, что показано на рис. 5.34. Вместо того чтобы выделить всю линию, Flash выделил только ее фрагмент.

Это произошло потому, что линия, рассекая эллипс на четыре части (две половинки контура и две половинки заливки), и сама оказалась разделенной на три отрезка. На рис. 5.35 это проиллюстрировано наглядно.

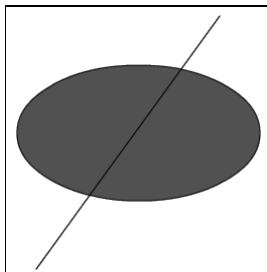


Рис. 5.33. Изначальная фигура, нарисованная для демонстрации эффекта фрагментации графики во Flash

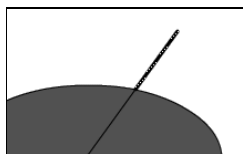


Рис. 5.34. После щелчка мышью выделенным остается только фрагмент линии, а не вся линия

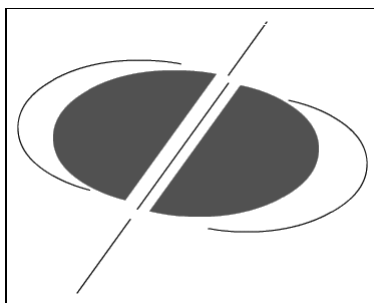


Рис. 5.35. Фрагменты inicialной фигуры, изображенной на рис. 5.33

Как видите, фрагментация налицо. Монолитная, казалось бы, фигура "развалилась" сама собой на семь частей. И каждая ее часть является абсолютно независимым фрагментом.

С фрагментацией тесно связана и другая особенность Flash-графики — *слияние* фигур. Заключается оно в том, что если два или более графических фрагментов, нарисованных одним цветом, соприкасаются, они превращаются в единое целое.

Давайте рассмотрим теперь иной случай. Нарисуем другое изображение, показанное на рис. 5.36. Оно представляет собой два перекрывающихся мазка "кистью", выполненных одним цветом (это обязательно).

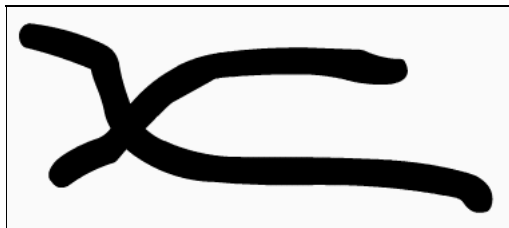


Рис. 5.36. Фигура, нарисованная для демонстрации эффекта слияния графики

Щелкните мышью по одному из этих двух мазков. Вы увидите, что выделен будет не только этот мазок, а вся фигура. Таким образом, два одноцветных мазка слились в одно целое.

Исходя из вышеописанного, можно вывести два правила поведения любой Flash-графики, состоящей из нескольких примитивов. Причем вид примитива в этом случае роли не играет.

- ❑ Примитивы, нарисованные разными цветами, фрагментируются, т. е. становятся независимыми фрагментами графики. Также фрагментируются пересеченные и изломанные линии любого цвета, как прямые, так и кривые, причем пересечены они могут быть как другой линией, так и примитивом другого вида (рис. 5.37).
- ❑ Примитивы, нарисованные одним цветом, сливаются, т. е. становятся одним целым. Исключение составляют только пересеченные и изломанные линии, даже если они нарисованы одним цветом, — такие линии фрагментируются.

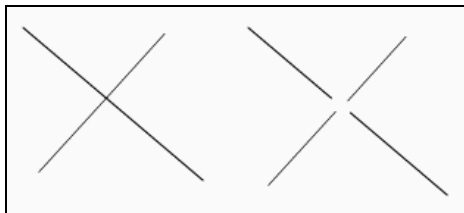


Рис. 5.37. Пример фрагментации пересеченных одноцветных линий

Спору нет, иногда и фрагментация, и слияние бывают полезными. Но не всегда. Сейчас мы рассмотрим способы избежать фрагментации и слияния сложных фигур.

Существует два способа избежать фрагментации:

- ❑ сгруппировать составляющие фигуру фрагменты в единое целое;
- ❑ сохранить созданную фигуру как образец в библиотеке (о библиотеках и образцах см. главу 10).

Чтобы предотвратить слияние фрагментов, Flash предлагает целых три способа:

- ☐ сгруппировать составляющие фигуру фрагменты;
- ☐ сохранить созданную фигуру как образец в библиотеке (о библиотеках и образцах см. главу 10);
- ☐ разнести фрагменты, которые не должны сливаться, по отдельным слоям (о слоях см. главу 15).

Группировка

Выше мы упомянули о группировке графических фрагментов. Выясним теперь, что это такое.

Группировка позволяет временно объединить разрозненные фрагменты в нечто целое — *группу* фрагментов. После группировки с этим целым можно работать как с обычным фрагментом. Чтобы сгруппировать фрагменты, выделите их и выберите пункт **Group** меню **Modify** или нажмите комбинацию клавиш <Ctrl>+<G>. Результат вы можете видеть на рис. 5.38.

Закончив работу с группой фрагментов, вы, вероятно, захотите разбить ее обратно на отдельные составляющие. Для этого выделите группу и выберите пункт **Ungroup** меню **Modify** или нажмите комбинацию клавиш <Ctrl>+<Shift>+<G>. Вы также можете выбрать пункт **Break Apart** в меню **Modify**, одноименный пункт в контекстном меню выбранной группы или нажать комбинацию клавиш <Ctrl>+. Группа будет тотчас разбита на отдельные фрагменты.

Если вам нужно изменить один из фрагментов, составляющих группу, не разгруппировывая ее, сделайте следующее. Выделите эту группу и выберите пункт **Edit Selected** в меню **Edit**, одноименный пункт в контекстном меню выделенной группы или просто дважды щелкните эту группу. На рабочем листе будут показаны только фрагменты, составляющие выделенную группу, все остальное будет закрашено серым. Измените что хотите и либо выберите в меню **Edit** пункт **Edit All**, либо щелкните кнопку возврата, расположенную над рабочим листом слева (рис. 5.39), либо дважды щелкните по пустому пространству на листе, чтобы вернуться к обычному режиму работы.

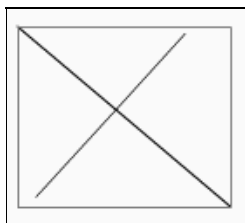


Рис. 5.38. Выбранная группа фрагментов



Рис. 5.39. Кнопка возврата

Правка графики

Flash предоставляет практически те же возможности по редактированию созданной графики, что и другие программы векторных графических редакторов. Если вы знакомы с ними, вам будет совсем просто. Если же нет, не беда — все операции правки графики выполняются исключительно просто и наглядно, так что вы быстро приобретете нужные навыки. Рассмотрим их подробнее.

Проверьте, выбран ли у вас инструмент "стрелка выделения" на панели инструментов (рис. 5.29). Подавляющее большинство операций по правке графики выполняются с помощью именно этого, хорошо уже знакомого вам инструмента. Если для каких-либо операций понадобятся другие инструменты, мы специально предупредим об этом.

Перемещение и удаление графики

Перемещение и удаление графики — самые простые и наглядные операции. Мы рассмотрим их вместе.

Прежде всего, вы можете перемещать выделенные фрагменты. Для этого сначала выделите их. Далее установите курсор мыши на какую-либо часть выделенного фрагмента, нажмите левую кнопку мыши и, не отпуская ее, перетащите выделенный фрагмент на новое место, после чего отпустите кнопку. Если вам нужно переместить графику в направлении под углом, кратным 45°, удерживайте во время перетаскивания клавишу <Shift>.

Для точного перемещения графики можно воспользоваться клавишами-стрелками. Выделите нужный фрагмент и нажимайте клавиши-стрелки, пока он не окажется на нужном месте. При однократном нажатии на клавишу-стрелку происходит смещение на один пиксел. Если же вы при этом будете удерживать клавишу <Shift>, при каждом нажатии выбранный фрагмент будет смещаться сразу на 8 пикселей.

На новое место можно переместить не сам выделенный фрагмент, а его копию. Для этого просто перетащите его, удерживая нажатой клавишу <Alt>. Вы также можете выбрать пункт **Duplicate** в меню **Edit** или нажать комбинацию клавиш <Ctrl>+<D>, чтобы создать новую копию выделенного фрагмента, а потом перетащить ее на новое место обычным способом.

Удалить ненужную графику совсем просто. Для этого выберите ее и либо нажмите клавишу или <Backspace>, либо выберите пункт **Clear** в меню **Edit**.

"Притягивание" графики

Ранее мы уже сказали, что графические фрагменты, перетаскиваемые по рабочему листу, могут "притягиваться" к другим фрагментам, линиям сетки и направляющим. Это сильно облегчает рисование сложных геометрических фигур: не нужно точно подгонять одну линию к другой. Обратите внимание,

что "притягивается" та точка на линии или заливке, за которую вы "ухватились" мышью. Эта точка (назовем ее *"точкой притягивания"*) отображается в виде достаточно большой черной окружности (рис. 5.40).

Однако это может быть и вредно, например, если вы хотите нарисовать очень сложную и извилистую кривую с помощью карандаша или провести две линии очень близко друг к другу. Поэтому Flash предоставляет вам возможность временно отключить "притягивание".

Отключить "притягивание" очень просто. Если у вас выбран инструмент "стрелка выделения", становится доступным модификатор "притягивание". Он представляет собой кнопку, показанную на рис. 5.41. По умолчанию эта кнопка нажата, т. е. модификатор включен. Чтобы его отключить, "отожмите" эту кнопку. Впоследствии вы, конечно, сможете снова его включить.



Рис. 5.40. "Точка притягивания"



Рис. 5.41. Модификатор "притягивание" (отключен)

Если инструмент у вас скрыт, вы можете воспользоваться меню. Пункт-выключатель **Snap to Objects** меню **View** и одноименный пункт контекстного меню листа действуют так же, как кнопка, показанная на рис. 5.41. Но быстрее всего, вероятно, нажать комбинацию клавиш <Ctrl>+<Shift>+</>.

Отключить "притягивание" к линиям координатной сетки и направляющим также очень просто. В первом случае отключите пункт-выключатель **Snap to Grid** подменю **Grid** меню **View** или нажмите комбинацию клавиш <Ctrl>+<Shift>+<'>. Во втором случае, по аналогии, отключите пункт-выключатель **Snap to Guides** подменю **Guides** меню **View** или нажмите комбинацию клавиш <Ctrl>+<Shift>+<;>. (Подменю **Guides** и **Grid** также присутствуют в контекстном меню листа.) Чтобы временно отключить "притягивание" к линиям пиксельной сетки, вы можете нажать клавишу <C>; когда вы ее отпустите, "притягивание" включится снова.

Работа с буфером обмена

Все современные более-менее сложные Windows-приложения позволяют работать со стандартным *буфером обмена* Windows. Пользователь может помещать в буфер обмена фрагменты текстов, графических изображений, схем, программных компонентов и затем вставлять их в другое место этого же или совсем другого документа, открытого совсем в другом приложении.

Разумеется, Flash также поддерживает работу с буфером обмена. И предоставляет стандартный набор операций, которые мы сейчас перечислим.

Вы можете "вырезать" выделенный графический фрагмент с листа и поместить его в буфер обмена. При этом сам выделенный фрагмент с листа пропа-

дает. Чтобы "вырезать" фрагмент, выделите его и выберите пункт **Cut** меню **Edit** или контекстного меню или нажмите комбинацию клавиш <Ctrl>+<X>.

Вы также можете скопировать в буфер обмена выделенный графический фрагмент, оставив его на листе. Для этого выберите пункт **Copy** меню **Edit** или контекстного меню или нажмите комбинацию клавиш <Ctrl>+<C>.

Чтобы вставить содержимое буфера обмена на лист, выберите пункт **Paste** меню **Edit** или контекстного меню или нажмите комбинацию клавиш <Ctrl>+<V>. Содержимое буфера обмена будет помещено в центр листа.

Если вы хотите поместить содержимое буфера обмена в ту же позицию, где находится изначальный фрагмент, выберите пункт **Paste in Place** меню **Edit** или контекстного меню или нажмите комбинацию клавиш <Ctrl>+<Shift>+<V>. Эта операция характерна только для Flash, в других программах она может и не поддерживаться.

Несколько слов о том, как Flash обрабатывает при вставке графику, созданную в других приложениях. При вставке текста на листе создается новый текстовый блок (о текстовых блоках см. главу 7). При вставке векторной графики на листе создается группа графических фрагментов, которая может быть разгруппирована и отредактирована. При вставке растрового изображения оно также помещается на рабочий лист (об импорте растровой графики см. главу 8).

Есть еще возможность вставки графики или текста в другом формате, которая также является стандартной для многих Windows-приложений. Например, векторную графику можно вставить как растровую. Для этого выберите пункт **Paste Special** меню **Edit**. На экране появится стандартное диалоговое окно специальной вставки Windows; это окно мало кому знакомо, поэтому мы приведем его вид на рис. 5.42. В списке, занимающем большую часть этого окна, выберите нужный формат вставки. После этого нажмите кнопку **ОК** для выполнения вставки или **Отмена (Cancel)** для отказа от нее.

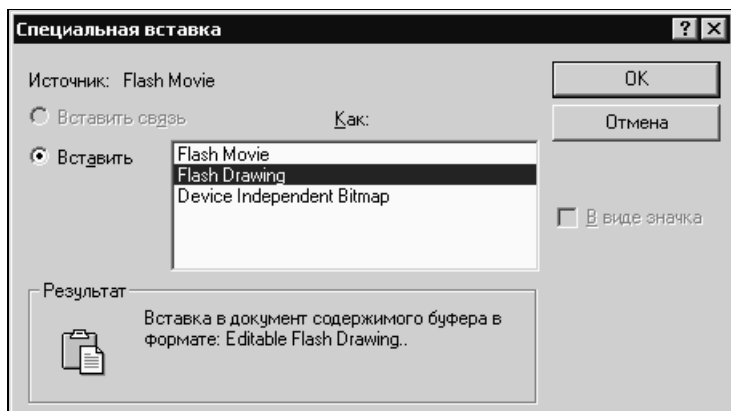


Рис. 5.42. Стандартное диалоговое окно специальной вставки Windows

Изменение цвета графики

Вы можете изменять цвет линий контура и заливки, просто выделив их и задав новый цвет с помощью селекторов цвета, расположенных в области **Colors** инструментария. Кроме того, вы можете воспользоваться аналогичными селекторами цвета, расположенными в редакторе свойств (рис. 5.43). Учтите, что этими селекторами нельзя задать цвета, которыми будет нарисована новая графика.



Рис. 5.43. Селекторы цветов, расположенные в редакторе свойств

Изменение формы фигуры

Контур любой фигуры, нарисованной средствами Flash, можно изменять и деформировать, тем самым меняя форму этой фигуры. Это выполняется все той же "стрелкой выделения" аналогично тому, как вы только что перемещали графику с места на место.

Скажем сразу, что для изменения формы фигуры нужно, прежде всего, снять с нее выделение. Если этого не сделать, будет выполнено перемещение не части фигуры, что повлечет за собой изменение ее формы, а самой фигуры (а именно, ее контура или заливки). А это совсем не то, что нам нужно.

Итак, чтобы изменить форму какой-либо фигуры, будь это простейший прямоугольник или сложная кривая, нужно выбрать на ее контуре нужную точку и переместить ее с помощью мыши. Вы ставите на эту точку курсор мыши, нажимаете левую кнопку, перемещаете мышь и отпускаете кнопку. В результате этого форма фигуры изменится. Если фигура имеет заливку, то форма заливки также изменится, чтобы заполнить получившийся в результате контур. Если контур в результате перемещения перестанет быть замкнутым, заливка сохранит форму, которую имела перед этим.

Линии можно удлинять и укорачивать, перетаскивая их за конечные точки. Вы можете преобразовывать прямые линии в кривые и наоборот, перемещая точки, расположенные на этих линиях, "вытягивая" их в сторону, а также перемещать углы (в терминологии Flash — *угловые точки*, т. е. точки соединения линий) геометрических фигур, тем самым вытягивая или "сжеживая" их. Возможности изменения формы графики Flash на самом деле весьма велики, и, чтобы овладеть ими, нужна практика.

Поясним вышесказанное на нескольких примерах.

Обычный прямоугольник мы можем "вытянуть" по диагонали за угол. Просто "захватите" один из его углов мышью и тяните, пока он не примет нужную вам форму. Результат показан на рис. 5.44. Обратите внимание, что форма заливки прямоугольника также изменяется, чтобы заполнить новый контур.

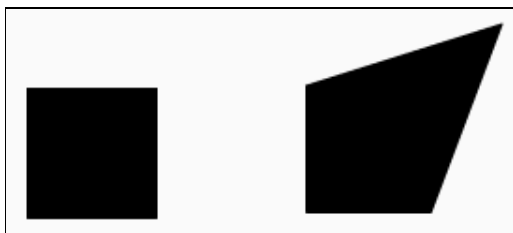


Рис. 5.44. Результат "вытягивания" угла прямоугольника (слева — исходный прямоугольник, справа — после изменения формы)

Также мы можем "вытянуть" любую из сторон этого прямоугольника, превратив ее в кривую. Результат показан на рис. 5.45.



Рис. 5.45. Результат "вытягивания" стороны прямоугольника (слева — исходный прямоугольник, справа — после изменения формы)

Аналогично мы можем изменить форму любой кривой и вообще любой геометрической фигуры. Посмотрите на рис. 5.46 — это результат деформации обычного круга. Как и в предыдущем случае, мы "вытягивали" контур окружности. Вы верите, что это когда-то было кругом?

Flash также предоставляет возможность создания новой угловой точки. Обычно угловые точки задаются при создании фигуры, но иногда впоследствии бывает нужно добавить новую угловую точку. Для этого выполните уже знакомую вам операцию по "вытягиванию" линии (прямой или кривой), но "вытягивайте" ее, удерживая нажатой клавишу <Ctrl>. В той точке, где вы "захватили" ее мышью, появится новая угловая точка. Результат может выглядеть так, как показано на рис. 5.47 — заметьте, что мы получили острый угол, которого раньше не было.

Теперь самое время немного попрактиковаться. Нарисуйте какую-либо достаточно сложную геометрическую фигуру и немного поиздевайтесь над ней. Попробуйте выполнить все описанные нами операции по несколько раз и посмотрите сами, что из этого получится.

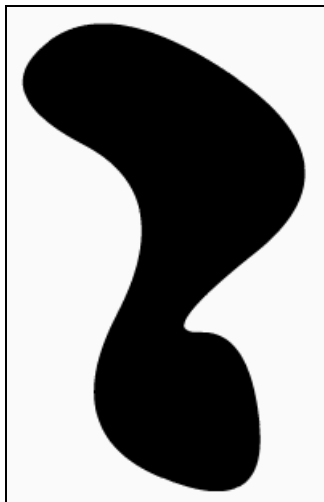


Рис. 5.46. Деформированный круг

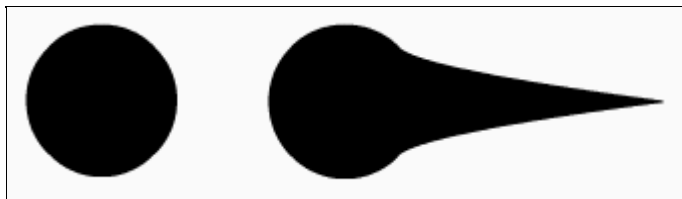


Рис. 5.47. Результат создания угловой точки на окружности (слева — исходная окружность, справа — после изменения формы)

Точная правка кривых

Итак, мы рассмотрели, как можно изменять форму графики с помощью инструмента "стрелка выделения". Как видите, это очень просто.

Однако во многих случаях такой способ может оказаться слишком грубым. В самом деле, далеко не все можно сделать с помощью "стрелки выделения". Некоторые особо точные модификации могут быть недоступны при использовании этого инструмента. Поэтому создатели Flash предусмотрели так называемую "дополнительную стрелку выделения" (или просто "белую стрелку"). Она позволяет вам выполнять самые сложные действия над проведенными линиями как прямыми, так и кривыми, и не только линиями, а над любыми фигурами.

Чтобы выбрать инструмент "дополнительная стрелка выделения", щелкните кнопку, показанную на рис. 5.48, в инструментарии или нажмите клавишу <A> на клавиатуре. Курсор мыши примет вид небольшой белой стрелки.



Рис. 5.48. Инструмент "дополнительная стрелка выделения"

С помощью "белой стрелки" вы можете выделять фрагменты графики, как вы делаете это обычной "стрелкой". Выделенная с ее помощью графика выглядит так, как на рис. 5.49. Имейте в виду, что с помощью "белой стрелки" нельзя выделять заливки — только контуры.

Посмотрите внимательно на рис. 5.49. Вы увидите, что на контуре нарисованной фигуры заметны точки, имеющие вид небольших сплошных квадратиков. Это угловые точки — точки смыкания линий, прямых и кривых. Перетаскивая эти точки, вы можете изменять форму контура (и, разумеется, заливки). Попробуйте самостоятельно это сделать.

Щелкните по любой угловой точке. Вы увидите, что она изменит вид, — вместо сплошного квадратика появится полый. Это означает, что угловая точка активизирована или выделена. Теперь вы можете удалить эту точку, если она не нужна. Для этого нажмите клавишу . Результат удаления одной из угловых точек нашей фигуры показан на рис. 5.50.

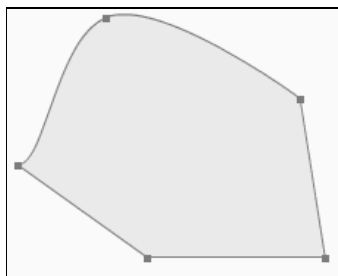


Рис. 5.49. Графика, выделенная с помощью инструмента "дополнительная стрелка выделения"

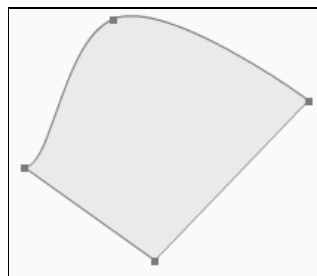


Рис. 5.50. Результат удаления одной из угловых точек фигуры, показанной на рис. 5.49

Если вы хотите добавить новую угловую точку, придется воспользоваться инструментом "перо" (графический фрагмент все еще должен быть выделен). Выберите его на панели инструментов (см. рис. 5.11) и щелкните на нужном месте нужной линии контура. После этого вы можете перетаскивать вновь созданную угловую точку, чтобы изменить контур. Если вы случайно поставили угловую точку не там, где хотели, то, пользуясь инструментом "перо", сможете легко ее удалить, для чего достаточно щелкнуть по ненужной угловой точке.

Теперь снова выберите инструмент "дополнительная стрелка выделения" и щелкните по точке, расположенной в левом верхнем углу нашей фигуры. На рис. 5.51 показан результат этого действия.



Рис. 5.51. Выделенная точка искривления

Как видите, эта точка имеет вид не полого квадратика, а окружности. Это *точка искривления*, она показывает, в каком месте соединяются две кривые или одна кривая и одна прямая. Вы также можете перетаскивать эту точку мышью. Кроме того, можно перетаскивать точки, расположенные на концах синей касательной, чтобы изменить форму получившейся кривой.

Чтобы преобразовать угловую точку в точку искривления, выделите ее и начните перетаскивать при нажатой клавише <Alt>. Вы увидите в этом случае, что, вместо того, чтобы переместить саму точку, Flash начнет изменять кривую. Обратное преобразование — точки искривления в угловую точку — выполняется с помощью инструмента "перо" щелчком по этой точке (второй щелчок удалит и угловую точку).

Внимание!

Если на вкладке **Editing** диалогового окна **Preferences** выключен флажок **Show Solid Points**, невыделенные ключевые точки кривой показываются в виде полых кружков и прямоугольников, а выделенные — в виде сплошных.

Есть также возможность перемещать угловые точки и точки искривления, пользуясь клавишами-стрелками. Конечно, предварительно эти точки нужно выделить.

Можно выделять сразу несколько угловых точек или точек искривления. Для этого при выбранном инструменте "дополнительная стрелка выделения" щелкайте по ним, удерживая нажатой клавишу <Shift>. Вы также можете выбрать все ключевые точки контура перетаскиванием мыши и перетащить на новое место сразу несколько угловых точек и точек искривления.

Сложное выделение. "Лассо"

Мы уже рассмотрели различные способы выделения графики. Добавим к ним еще парочку способов так называемого *сложного* выделения. При сложном выделении выделяются не целые графические примитивы, а их части, что позволяет художнику создавать еще более замысловатые геометрические фигуры и, стало быть, еще более оригинальную графику.

Прежде всего, рассмотрим уже знакомый нам инструмент "стрелка выделения". Все ли мы о нем знаем?

Оказывается, нет.

Давайте нарисуем на чистом листе Flash небольшой эллипс. После этого попытаемся выделить перетаскиванием мыши его небольшую часть. Для

этого сначала поставим курсор мыши выше нарисованного нами эллипса и чуть левее его центра. После этого нажмем левую кнопку мыши и протащим ее, пока ее курсор не окажется ниже эллипса и чуть правее его центра. Отпустим кнопку мыши и посмотрим, что у нас получилось. А получится у нас должно что-то, похожее на рис. 5.52.

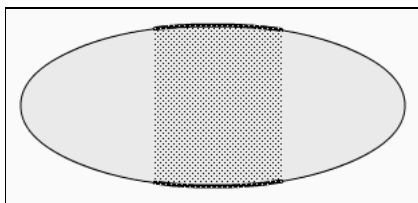


Рис. 5.52. Эллипс, фрагмент которого выделен перетаскиванием мыши

Теперь проверим, действительно ли фрагмент эллипса выделен, и мы сможем проделывать над ним все, что захотим. Для пробы перетащим выделенный фрагмент на другое место листа. Получилось — посмотрите на рис. 5.53.

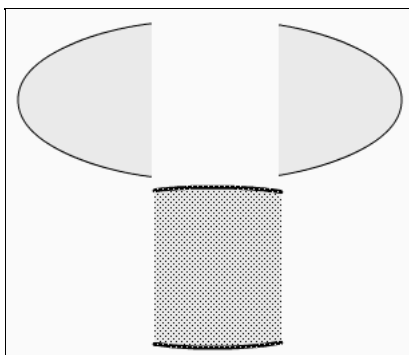


Рис. 5.53. Эллипс, выделенный фрагмент которого был перенесен на другое место

Итак, как видите, с помощью уже знакомой нам "стрелки выделения" можно выделять не только целые фигуры, но и их фрагменты. Это может вам очень пригодиться в дальнейшем, когда вы начнете работать с Flash. В самом деле, каких геометрических фигур можно надеть!

Но как быть, если нужно выделить не прямоугольный фрагмент, а "лоскут" неправильной формы? "Стрелка выделения" здесь не поможет. Нужен другой инструмент, который называется "лассо".

Чтобы выбрать инструмент "лассо", щелкните кнопку, показанную на рис. 5.54, в инструментарии или нажмите клавишу <L> на клавиатуре. Курсор мыши примет вид небольшого лассо.



Рис. 5.54. Кнопка включения инструмента "лассо"

Как пользоваться "лассо"? Поставьте курсор мыши в точку, где собираетесь начать выделение, нажмите левую кнопку мыши и, не отпуская ее, проведите границу выделения. Не забудьте замкнуть ее, впрочем, Flash все равно замкнет ее, когда вы отпустите кнопку мыши. После этого останется только отпустить кнопку мыши — и "лоскут" выделен (рис. 5.55).

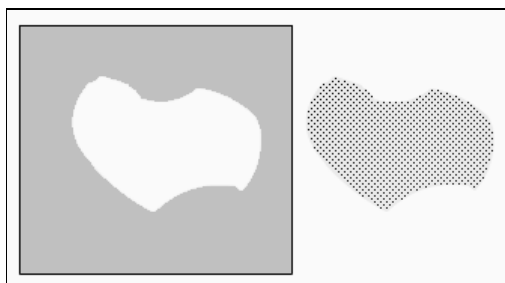


Рис. 5.55. Фрагмент прямоугольника, выделенный с помощью инструмента "лассо"

"Лассо" имеет несколько дополнительных возможностей, которые могут помочь в работе. Эти возможности включаются особыми модификаторами, доступными, как обычно, в области **Options** инструментария, если выбран инструмент "лассо". Давайте их рассмотрим.

Модификатор "полигон" переключает "лассо" в такой режим, когда область выделения представляет собой не замкнутую фигуру "свободной" формы, а многоугольник, состоящий из отрезков прямых линий. Кнопка этого модификатора показана на рис. 5.56. Чтобы включить "полигон", щелкните эту кнопку, после чего она останется нажатой. Чтобы отключить "полигон", еще раз щелкните эту кнопку, и она "отожмется".



Рис. 5.56. Кнопка включения модификатора "полигон"

При включенном "полигоне" работа с "лассо" протекает так. Вы выбираете, где будет начальная точка этого многоугольника, ставите туда курсор мыши, щелкаете левой кнопкой и протаскиваете мышью, "вытягивая" за собой первый отрезок. Подведя курсор к точке, где должна находиться вторая точка многоугольника, вы опять щелкаете кнопкой. Далее точно таким же образом вы щелчками мыши ставите третью, четвертую и все остальные точки этого

многоугольника. Чтобы замкнуть его, на последней точке выполните не одинарный, а двойной щелчок.

Пример такой многоугольной области выделения приведен на рис. 5.57.



Рис. 5.57. Фрагмент прямоугольника, выделенный с помощью инструмента "лассо" при включенном модификаторе "полигон"

Создание многоугольной области выделения похоже на рисование многоугольников с помощью инструмента "перо". Вам будет проще научиться пользоваться "полигоном", если вы будете помнить эту аналогию.

Находясь в обычном режиме работы "лассо", вы можете временно переключиться в режим "полигона". Для этого при рисовании области выделения нужно удерживать нажатой клавишу <Alt>.

Стирание графики. "Ластик"

Компьютерный (да и обычный, "бумажный") художник не только рисует. Увы, иногда приходится и удалять только что нарисованное. В конце концов, художник — человек, а не программа, и он может ошибиться. А иногда стирание просто необходимо, если нужно создать сложную геометрическую фигуру путем удаления части другой фигуры, более простой.

Конечно, вы всегда можете выделить часть графики, которую нужно удалить, и нажать . Выделить ее можно как с помощью уже знакомой вам "стрелки выделения", так и "зааркавив" "лассо". Но Flash предлагает еще один инструмент для удаления нарисованных контуров и заливок — "ластик". Сейчас мы его рассмотрим.

Инструмент "ластик" служит для стирания части графики. В этом смысле он ведет себя как обычный резиновый ластик, которым вы стираете карандашные пометки. Если брать аналогии из Flash, то "ластик" похож на "кисть": он работает точно так же, только не закрашивает лист мазками, а стирает все, что на нем нарисовано.

Чтобы выбрать инструмент "ластик", щелкните кнопку, показанную на рис. 5.58, в инструментарии или нажмите клавишу <E> на клавиатуре. Курсор мыши при этом может принимать разные формы, в зависимости от настроек, заданных с помощью модификаторов (см. ниже).



Рис. 5.58. Кнопка включения инструмента "ластик"

Используется инструмент "ластик" аналогично "кисти". Вы ставите курсор мыши в начальную точку, нажимаете левую кнопку, двигаете мышью, стирая графику, и, в конце концов, отпускаете кнопку мыши, закончив стирание. Пример того, что может получиться у вас, показан на рис. 5.59.

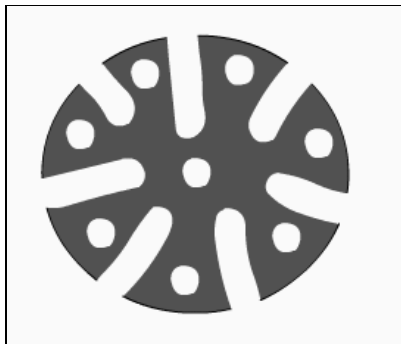


Рис. 5.59. Пример использования "ластика"

Если вам при работе нужно двигать "ластик" строго прямо по горизонтали или вертикали, то во время рисования удерживайте нажатой клавишу <Shift>. Кроме того, вы можете быстро стереть все нарисованное на листе, дважды щелкнув кнопку выбора "ластика" в инструментарии (см. рис. 5.58). Это значительно быстрее, чем последовательно нажимать сначала комбинацию клавиш <Ctrl>+<A> (выбрать всю графику), потом — клавишу (собственно удаление). Только будьте внимательны, не удалите случайно все свои труды.

Так же, как и в случае с "кистью", Flash предоставляет возможность выбора формы и размера ластика. Для этого служит единственный модификатор "форма ластика", расположенный в области **Options** инструментария (рис. 5.60). Этот модификатор представляет собой обычный раскрывающийся список, в котором в виде графических изображений представлены все доступные во Flash формы и размеры "ластика". Надо сказать, что, по сравнению с "кистью", их не очень много.

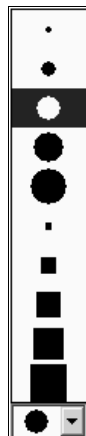


Рис. 5.60. Модификатор "форма ластика"

Как только вы выберете с помощью этого модификатора форму и размер "ластика", форма и размер курсора мыши изменятся. Таким образом, вы всегда будете знать, каким "ластиком" вы работаете.

Еще один модификатор позволяет задать режим работы "ластика", т. е. установить, будет ли "ластик" стирать линии или ограничиваться лишь заливками. Этот модификатор имеет вид кнопки с меню (рис. 5.61).

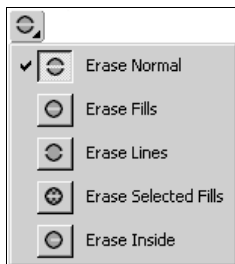


Рис. 5.61. Кнопка модификатора "стирание" с раскрытым меню

В меню модификатора "стирание" доступно пять пунктов:

- ☐ **Erase Normal** — обычный режим, когда стираются любые линии и любые заливки (см. рис. 5.59);
- ☐ **Erase Fills** — стираются только заливки, линии не стираются (рис. 5.62);
- ☐ **Erase Lines** — стираются только линии, заливки не стираются (рис. 5.63);
- ☐ **Erase Selected Fills** — стирается только выделенная с помощью "стрелки" заливка, невыделенные заливки и любые линии остаются нестертыми (рис. 5.64);
- ☐ **Erase Inside** — стирается только заливка, находящаяся в пределах замкнутого контура. То, что находится по другую сторону линий, составляющих этот контур, не стирается. Линии также не стираются (рис. 5.65).

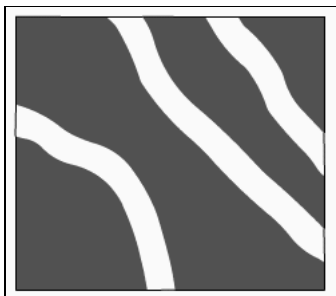


Рис. 5.62. Результат применения режима стирания **Erase Fills** (обратите внимание — линии остались нестертыми)



Рис. 5.63. Результат применения режима стирания **Erase Lines** (обратите внимание — заливка осталась нестертой)

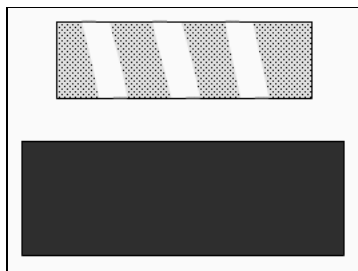


Рис. 5.64. Результат применения режима стирания **Erase Selected Fills** (заливка верхнего прямоугольника была выделена с помощью инструмента "стрелка", а заливка нижнего — нет)

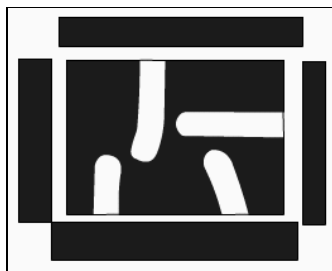


Рис. 5.65. Результат применения режима стирания **Erase Inside** (стиралась заливка внутреннего прямоугольника, причем мазки были направлены наружу, но внешние прямоугольники затронуты не были)

Но что делать, если нужно быстро стереть фрагмент графики, например, заливку или линию? Можно, конечно, переключиться на инструмент "стрелка выделения", выделить нужный фрагмент графики и нажать . А можно воспользоваться так называемым "краном". С помощью такого "крана" вы просто "смоете" ненужный фрагмент с листа. Модификатор "кран" представляет собой кнопку, показанную на рис. 5.66. Чтобы включить "кран", щелкните эту кнопку, после чего она останется нажатой. После этого вы можете просто щелкать по фрагментам графики, чтобы их удалить. Для отключения "крана" еще раз щелкните эту кнопку, и она "отожмется".



Рис. 5.66. Модификатор "кран"

Дополнительные возможности работы с контурами

А теперь перечислим некоторые дополнительные возможности по работе с контурами, предоставляемые Flash.

Вы уже, наверно, заметили, что при рисовании некоторыми инструментами (в частности, "карандашом" и "кистью") Flash автоматически сглаживает получившуюся графику. Однако Flash предлагает средства для сглаживания уже нарисованной графики. Это особые модификаторы, доступные в области **Options** инструментария, если выбран инструмент "стрелка выделения".

Первый из этих модификаторов — "сглаживание". Он, как вы уже поняли, сглаживает кривые линии, устраняет рывки, сдвиги, нарушения кривизны и прочие дефекты, а также по мере возможностей упрощает линии, иначе говоря, *оптимизирует* их. Благодаря такой оптимизации уменьшается количе-

ство описывающих их точек, а значит, уменьшается размер файла фильма и время, необходимое для рисования кривых. Умелая и тщательная оптимизация всегда стоит затраченного на нее времени, так как выигрыш потом окупится многократно.

Модификатор "сглаживание" имеет вид кнопки, показанной на рис. 5.67. Чтобы выполнить сглаживание какой-либо кривой, выделите ее и нажмите эту кнопку. (Если на листе ничего не выбрано, кнопка недоступна.) Вы также можете выбрать пункт **Smooth** меню **Modify** или контекстного меню выделенного фрагмента. Чтобы выполнить более сильное сглаживание, нажмите эту кнопку или выберите этот пункт меню несколько раз.



Рис. 5.67. Модификатор "сглаживание"

Здесь нужно сразу сказать, что применение модификатора "сглаживание" к прямым линиям и геометрическим фигурам, состоящим из прямых, не дает никакого эффекта. Также не дает эффекта применение этого модификатора к простым кривым линиям с одинаковым радиусом кривизны.

На рис. 5.68 показан результат применения модификатора "сглаживание".

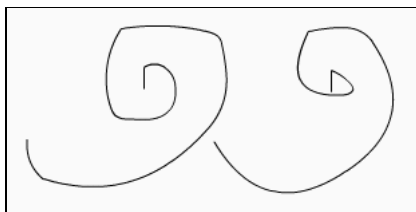


Рис. 5.68. Результат применения модификатора "сглаживание"
(слева — исходная кривая, справа — после применения модификатора)

Модификатор "спрямление" выполняет совершенно противоположную задачу. Он спрямляет кривые линии, заостряет углы и пытается привести созданные вами геометрические фигуры к простейшим: прямая, прямоугольник, треугольник, эллипс. Фактически, он тоже оптимизирует графику, приводя ее к простейшим фигурам.

Модификатор "спрямление" имеет вид кнопки, показанной на рис. 5.69. Чтобы выполнить спрямление линии, выберите ее и нажмите эту кнопку. (Если на листе ничего не выбрано, кнопка также недоступна.) Вы также можете выбрать пункт **Straighten** меню **Modify** или контекстного меню выделенного фрагмента. Опять же, чтобы выполнить более сильное спрямление, нажмите эту кнопку или выберите этот пункт меню несколько раз. Пример применения модификатора "спрямление" показан на рис. 5.70.



Рис. 5.69. Модификатор "спрямление"

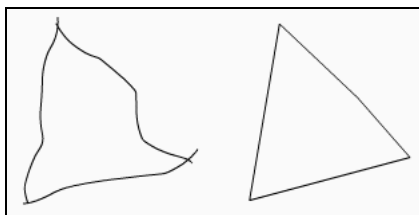


Рис. 5.70. Результат применения модификатора "спрямление" (слева — исходная кривая, справа — после применения модификатора)

Применение этого модификатора к прямым линиям и фигурам, состоящим из прямых линий, не имеет никакого эффекта. Также, по-видимому, он никак не действует на кривые линии с одинаковым радиусом кривизны.

Есть еще один способ сглаживания (и оптимизации) кривых. Выделите на листе графический фрагмент, который нужно оптимизировать, и выберите пункт **Optimize** в меню **Modify** или нажмите комбинацию клавиш <Ctrl>+<Alt>+<Shift>+<C>. На экране появится диалоговое окно **Optimize Curves**, показанное на рис. 5.71.

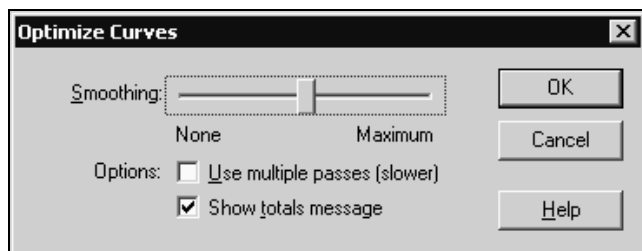


Рис. 5.71. Диалоговое окно **Optimize Curves**

С помощью движкового регулятора **Smoothing** выбирается степень сглаживания кривых. Крайнее левое значение вообще отключает всякое сглаживание, а крайнее правое — задает высшую степень оптимизации. По умолчанию движок стоит на середине шкалы, вероятно, это лучший выбор.

Флажок **Use multiple passes (slower)** заставляет Flash оптимизировать выбранный фрагмент несколько раз подряд, пока не будет ясно, что дальнейшая оптимизация бесполезна. Такая многопроходная оптимизация выполняется значительно медленнее однократной, но позволяет достичь лучших результатов. Однако по умолчанию этот флажок выключен, так как такая многократная оптимизация может занять много времени.

Флажок **Show totals message** выводит в конце оптимизации окно, содержащее сведения о результатах (рис. 5.72). Чтобы закрыть это окно, нажмите кнопку **ОК**. По умолчанию этот флажок включен.

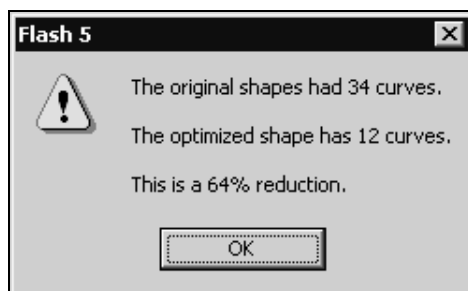


Рис. 5.72. Окно сведений о результатах оптимизации кривых

Установив нужные параметры сглаживания, нажмите кнопку **ОК**. Если вы передумали сглаживать выбранный фрагмент, нажмите кнопку **Cancel**. В конце процесса сглаживания и оптимизации Flash выдаст окно результатов, показанное на рис. 5.72, если, конечно, флажок **Show totals message** был включен.

Осталось рассмотреть совсем немного. А именно, три интересные возможности по работе с заливками и фигурами, содержащими заливки.

Первая возможность — это преобразование линий в заливки. Зачем это может пригодиться? Хотя бы для того, чтобы применить к линии какую-либо сложную, например, градиентную, заливку (о сложных заливках см. главу 6). Flash не позволит применить ее непосредственно к линии, но к заливке — элементарно. Кроме того, разработчики Flash утверждают, что преобразование линий в заливки в некоторых случаях ускоряет отображение графики, хотя размер файла может при этом увеличиться.

Чтобы преобразовать линию в заливку, выделите нужные линии (это может быть как простая прямая линия, так и сложный контур, содержащий множество кривых Безье) и воспользуйтесь пунктом **Convert Lines to Fills**, находящемся в подменю **Shape**, расположенном, в свою очередь, в меню **Modify**. Линия будет тотчас же преобразована в заливку.

Здесь нужно дать небольшие пояснения. После преобразования наша (бывшая) линия будет выглядеть как очень тонкая заливка, заключенная между очень тонкими линиями. Если вы попытаетесь исправить ее форму, воспользовавшись "стрелкой выделения", то сразу это поймете (рис. 5.73). На этом рисунке видно, что на самом деле мы переместили одну из тонких внешних линий, и заливка сразу же "распухла", чтобы заполнить образовавшееся свободное пространство.

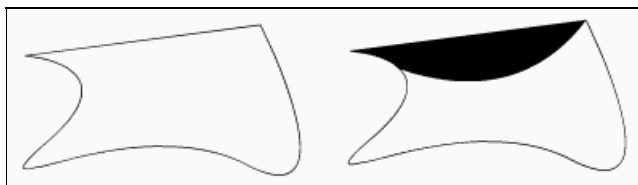


Рис. 5.73. Результат изменения формы контура, преобразованного в заливку (слева — до изменения, справа — после)

Вторая возможность — увеличение или уменьшение размера заливки на определенное число пикселей. Это может пригодиться в некоторых случаях. В частности, очень интересный эффект можно получить, если увеличить таким образом размер заливки, созданной из линии — линия "распухнет". Однако сразу скажем, что при этом может исказиться форма заливки и пропасть контур, если он есть. Поэтому используйте эту функцию осмотрительно.

Выделите нужную заливку (она может содержать или не содержать контур). Далее выберите пункт **Expand Fill** подменю **Shape** меню **Modify**. На экране появится диалоговое окно **Expand Fill** (рис. 5.74).

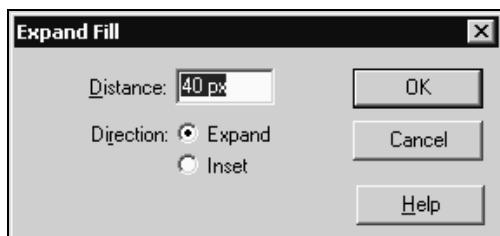


Рис. 5.74. Диалоговое окно **Expand Fill**

В поле ввода **Distance** укажите расстояние, на которое увеличивается или уменьшается заливка. Это расстояние вводится в пикселах. С помощью группы переключателей **Direction** задается, будет ли заливка увеличиваться (переключатель **Expand**) или уменьшаться (**Inset**). После этого остается только нажать кнопку **OK** для выполнения действия или **Cancel** — для отказа от него.

Результат последовательного выполнения сначала уменьшения, а потом — увеличения заливки показан на рис. 5.75. Видно, как после этих операций у заливки пропал контур, а сама заливка исказилась.

Третья возможность — сглаживание контура фигуры. При этом контур не только сглаживается, но еще и делается по возможности незаметным: его толщина уменьшается до минимума, а цвет задается таким же, как и у заливки. Это также может быть полезно во многих случаях, например, для того, чтобы устранить в контуре ненужные линии.

Выделите на листе фигуру, контур которой нужно сгладить. Далее выберите пункт **Soften Fill Edges** подменю **Shape** меню **Modify**. На экране появится диалоговое окно **Soften Edges** (рис. 5.76).

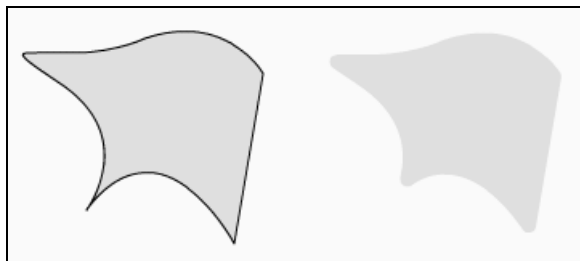


Рис. 5.75. Результат последовательного увеличения и уменьшения заливки (слева — до изменения, справа — после)

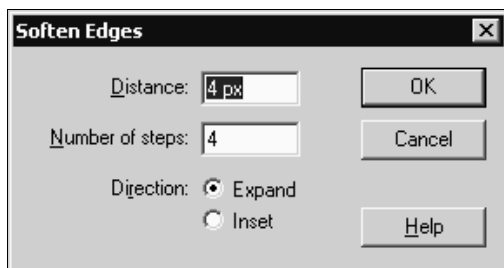


Рис. 5.76. Диалоговое окно **Soften Edges**

В поле ввода **Distance** указывается (в пикселах) толщина контура, который получится в результате сглаживания.

В поле ввода **Number of steps** задается максимальное количество кривых, которое будет использовано для создания нового, сглаженного контура. Чем больше кривых будет при этом использовано, тем более сложным получится контур, но тем большим окажется результирующий файл, и тем дольше будет рисоваться графика. Поэтому нужно выбрать разумный компромисс, возможно, для этого понадобится немного поэкспериментировать.

С помощью группы переключателей **Direction** задается, будет ли фигура увеличена (переключатель **Expand**) или уменьшена (**Inset**) для достижения эффекта сглаживания контура.

Введя нужные данные, нажмите **OK** для выполнения действия или **Cancel** — для отказа от него.

Результат сглаживания контура, примененного к сложной фигуре, нарисованной с помощью "карандаша", показан на рис. 5.77. Видно, что после этой операции контур стал невидимым, а его форма стала несколько менее слож-

ной. Также были устранены некоторые дефекты контура, в частности, ненужные линии.

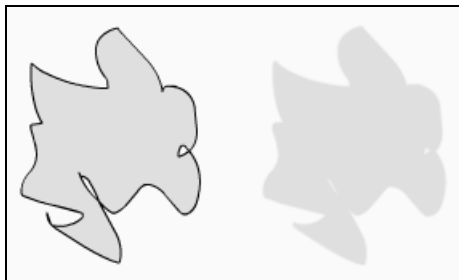


Рис. 5.77. Результат сглаживания контура фигуры (слева — до изменения, справа — после)

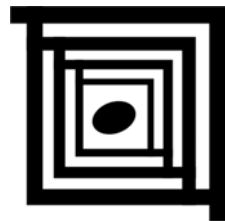
Отмена и повтор выполненных действий

Flash, как и многие другие Windows-приложения, поможет вам исправить случайные ошибки, обязательно возникающие при работе. Вы всегда сможете отменить результат нескольких последних ваших действий, восстановив то, что было до них. Эта возможность, называемая также *"откатом"*, позволит вам экспериментировать смелее, пробуя различные инструменты и преобразования. Если вам что-то не понравится, вы всегда сможете "откатиться" назад.

Чтобы отменить результат последнего вашего действия, выберите пункт **Undo** в меню **Edit** или нажмите комбинацию клавиш <Ctrl>+<Z>. Последовательно выбирая этот пункт (или нажимая клавишную комбинацию), вы можете отменять сразу несколько последних выполненных действий. Помните только, что по умолчанию Flash в состоянии отменить только 100 последних действий; информация о более ранних ваших действиях теряется, чтобы не занимать оперативную память. Однако в настройках программы Flash (см. главу 3) это число можно изменить.

Иногда бывает нужно, наоборот, отменить результат отмены последнего вашего действия ("откатить" сам "откат"). Для этого сразу после выполнения операции "отката" выберите пункт **Redo** в меню **Edit** или нажмите комбинацию клавиш <Ctrl>+<Y>. Такая операция часто называется *"возвратом"*.

Глава 6



Работа с цветом

Итак, с рисованием мы разобрались. Теперь стоит поговорить о цветах и работе с ними.

Зачем мы вынесли разговор о цветах и их поддержке во Flash в отдельную главу? Почему не описали работу с цветами в предыдущей главе, рассказывающей о рисовании? На это есть причины. И первая из них — важность правильного подбора цвета для изображения.

Ведь что такое рисование на самом деле? В основном это работа с цветом. Настроение любой картины создается в значительной степени правильным подбором цветов. Более того, настоящий художник должен иметь превосходное цветовое зрение, чтобы различить и перенести на бумагу (холст, дерево или иной носитель) малейшие оттенки окружающего мира. И любой настоящий художник сам готовит себе краски...

Ну да ладно, мы несколько отвлеклись от темы. Вернемся к нашим компьютерам.

Современные художники, работающие на современных компьютерах, имеют в своем распоряжении богатейший выбор всевозможных цветов. В настоящее время видеокарты могут выводить одновременно на экран миллионы цветов и оттенков — это значительно больше, чем может различить человеческий глаз. Как видите, аппаратные возможности компьютера не накладывают никаких ограничений на творца.

Проблема совсем в другом: сможет ли творец распорядиться этим богатством? Но это уже проблема самого творца. И уж никак не компьютера, и не Flash.

В этой главе мы поведем разговор о средствах, предлагаемых Flash для работы с цветом. Мы подробнее рассмотрим уже знакомый вам селектор цвета, дополнительные инструменты и специальные панели. А в самом конце мы поговорим о том, какие "подводные камни" подстерегают начинающего Flash-художника при публикации его творений.

Инструменты выбора цвета

Но начнем мы с рассмотрения различных способов выбора нужного вам цвета. Вы даже не представляете, сколько их!

Использование селектора цвета

Наличие селектора цвета — отличительная черта всех продуктов фирмы Macromedia. С ним мы уже знакомы. Мы знаем, как выбрать цвет с его помощью. Но всех его возможностей мы еще не знаем.

Давайте взглянем на рис. 6.1, на котором изображен открытый селектор цвета. Мы видим, что большая его часть занята набором небольших квадратов, представляющих все доступные для выбора цвета, — так называемой *палитрой*. Чтобы выбрать требуемый цвет, поместите курсор мыши над нужным квадратиком и щелкните левой кнопкой мыши. Окно селектора цвета при этом закроется.

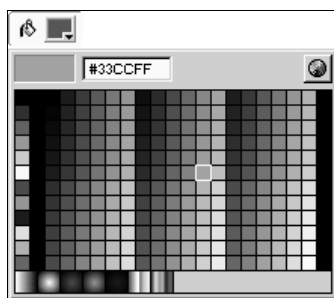


Рис. 6.1. Открытый селектор цвета (был выбран один из селекторов, расположенных в области **Colors** инструментария)

Однако селектор цвета предоставляет еще одну интересную возможность. Вы можете выбрать цвет, просто "взяв" его из любой точки экрана. Если вы присмотритесь к курсору мыши, то заметите, что он имеет вид небольшой пипетки. Щелкните этой пипеткой по нужной точке экрана, и цвет этой точки будет выбран селектором. (Окно селектора при этом должно быть открыто.) Таким образом, вы можете "взять" для работы цвет окна Windows, цвет открытой в графическом редакторе фотографии и т. п.

В верхней части окна селектора находится небольшое поле ввода. В этом поле отображается шестнадцатеричный *код цвета*, который находится под курсором мыши. Все цвета, отображаемые на экране компьютера, однозначно идентифицируются этим кодом. Попробуйте перемещать курсор (имеющий вид пипетки) по экрану и посмотрите, как будет изменяться содержимое этого поля ввода. Вы также можете ввести нужное значение кода цвета вручную, если, конечно, знаете его.

В верхнем правом углу окна селектора цвета находится небольшая кнопка, показанная на рис. 6.2. С помощью этой кнопки вы можете вызвать на экран стандартное диалоговое окно выбора цвета Windows. Это может понадобиться, если вы не смогли найти в палитре селектора нужного вам цвета.



Рис. 6.2. Кнопка вызова стандартного диалогового окна выбора цвета Windows

Само диалоговое окно выбора цвета показано на рис. 6.3. Как им пользоваться описано в интерактивной справке Windows. Вы также можете щелкнуть по кнопке с изображением вопросительного знака, находящейся в правой части заголовка окна, и далее щелкнуть по интересующему вас элементу управления. На экране появится всплывающая подсказка, содержащая текст описания.

Если в инструментарии выбран инструмент "прямоугольник", "эллипс" или "перо", то в верхнем правом углу окна селектора цвета появляется еще одна кнопка, показанная на рис. 6.4. Это кнопка *отключения цвета*, она позволит вам запретить рисование контура или заливки, в зависимости от того, в каком селекторе цвета она была нажата. То есть, если вы открыли селектор цвета фона, включили эту кнопку и нарисовали прямоугольник, то он будет нарисован без заливки.

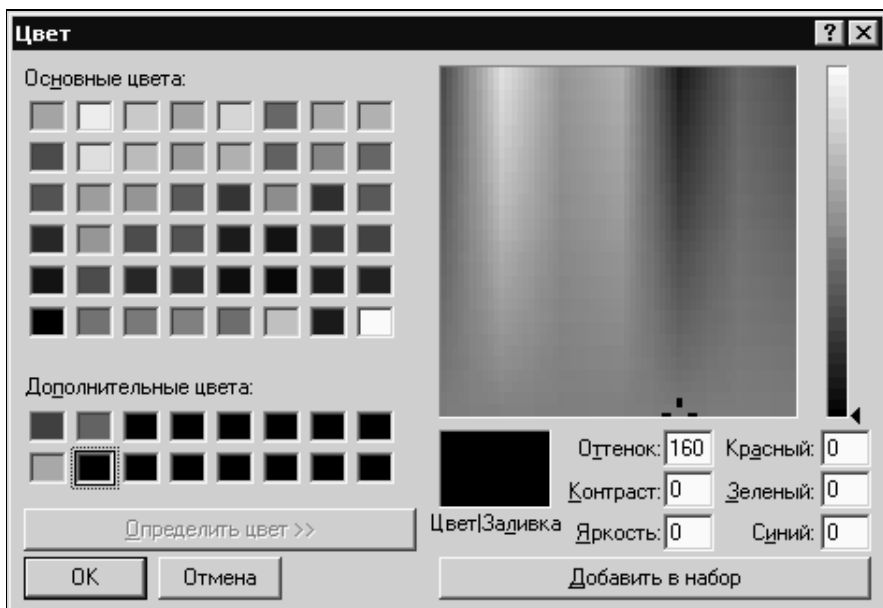


Рис. 6.3. Стандартное диалоговое окно выбора цвета Windows



Рис. 6.4. Кнопка отключения цвета

Если кнопка отключения цвета была нажата, то соответствующий селектор цвета будет перечеркнут (рис. 6.5). Чтобы снова включить цвет, просто выберите его в селекторе цвета.

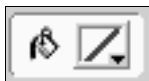


Рис. 6.5. "Отключенный" селектор цвета

Область **Colors** инструментария

Кнопка отключения цвета также дублируется в нижней части области **Colors** инструментария. Она имеет такое же обозначение, только меньше в размерах. На наш взгляд, пользоваться этой кнопкой удобнее, чем ее аналогом, находящемся в окне селектора цвета. Просто щелкните левее нужного селектора цвета (по его обозначению — значку карандаша или ведра с краской), чтобы его выбрать, и нажмите эту кнопку. Чтобы вернуть цвет, выберите селектор цвета и еще раз щелкните эту кнопку — она "отожмется".

Если речь зашла о нижней части области **Colors** инструментария, то нужно сказать еще о двух кнопках, находящихся там. Эти кнопки предоставляют дополнительные возможности выбора цвета.

На рис. 6.6 показана кнопка, задающая для цветов линии и заливки значения по умолчанию. Во Flash это соответственно черный и белый цвета.



Рис. 6.6. Кнопка задания цветов линии и заливки по умолчанию



Рис. 6.7. Кнопка, меняющая местами цвета линии и заливки

Кнопка, показанная на рис. 6.7, позволит вам быстро поменять местами цвета линии и заливки. Иногда это тоже нужно.

Создание новых цветов. Смеситель

Конечно, палитра селектора цветов весьма богата, и в некоторых случаях ее богатства может хватить. Однако для работы часто нужны цвета, которые в ней не присутствуют. Поэтому Flash предоставляет инструмент, позволяющий создавать новые цвета, применять их к выбранным на листе графическим фрагментам и добавлять при необходимости в палитру. Этот инструмент — особая панель, называемая *смесителем цветов*. Сейчас мы его рассмотрим.

Чтобы вызвать смеситель на экран, выберите в меню **Window** пункт **Color Mixer**. Вы также можете выбрать пункт **Mixer** подменю **Panels** контекстного меню листа или нажать комбинацию клавиш <Shift>+<F9>. Появившаяся на экране панель смесителя показана на рис. 6.8.

Вы можете щелкнуть расположенную в нижнем правом углу этой панели кнопку, имеющую вид треугольной стрелки, направленной вниз. После этого смеситель примет вид такой, как на рис. 6.9. В дальнейшем мы будем работать с развернутым смесителем, так как в таком виде он предоставляет больше возможностей, хоть и занимает больше места на экране.

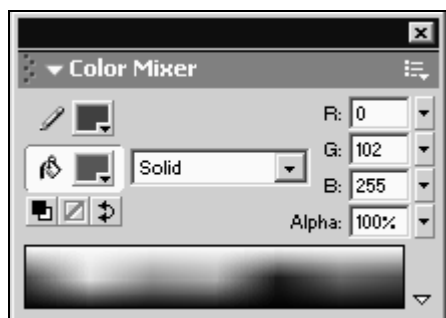


Рис. 6.8. Панель смесителя (сжатая)



Рис. 6.9. Панель смесителя (развернутая)

Посмотрите на левую часть смесителя. Там все знакомо: два селектора цвета — линии и заливки, — которые можно выбирать мышью и раскрывать, три дополнительные кнопки, рассмотренные нами чуть выше. Эти элементы управления дублируют область **Colors** инструментария и могут пригодиться, если инструментарий скрыт.

В нижней части смесителя расположены два поля выбора цвета. Задать с их помощью нужный цвет очень просто. Для этого сначала выберите нужный селектор цвета — линии или заливки. После этого щелкните мышью по нужной точке центрального поля, чтобы выбрать сам цвет, затем щелкните по нужной точке правого поля, чтобы задать его яркость. После этого заданный вами цвет появится в выбранном селекторе и, одновременно, в расположенной слева области предварительного просмотра.

Если вы сжали панель смесителя, поле выбора цвета также будет присутствовать в нем. Но оно имеет очень малые размеры, и зачастую подобрать точно нужный цвет весьма затруднительно. Поэтому не обойтись без тонкой

подстройки цвета, которая выполняется с помощью элементов управления, расположенных в правой части смесителя.

Здесь нужно сказать о *режимах задания цвета*. Как вы уже знаете, каждый цвет в компьютере однозначно описывается неким набором чисел — кодом цвета. Так вот: режим задания цвета как раз и определяет, что собой представляет этот набор чисел и как он интерпретируется компьютером (точнее, программным обеспечением). Flash поддерживает два режима задания цвета, которые мы сейчас рассмотрим.

- ❑ Режим *RGB* (от английского Red, Green, Blue — красный, зеленый, синий). Числовой набор, описывающий цвет, представляет собой три числа, описывающие относительную долю в нем красной, зеленой и синей составляющей. Каждое число может принимать значения от 0 (данная составляющая отсутствует) до 255 (максимальное значение составляющей). Как видите, каждое такое число имеет размер в один байт; итого цвет кодируется тремя байтами. Этот режим выбран по умолчанию.
- ❑ Режим *HSB* (от английского Hue, Saturation, Brightness — цвет, насыщенность, яркость). Числовой набор, описывающий цвет, представляет собой три числа, описывающие оттенок, его насыщенность и яркость. Каждое число также имеет размер в один байт, а значит, цвет кодируется тремя байтами.

Задание нужного цветового режима осуществляется путем выбора соответствующего пункта дополнительного меню смесителя. Эти два пункта работают как один двухпозиционный переключатель. Это значит, что при выборе одного из них слева от его названия появляется галочка, говорящая о том, что данный режим (и пункт) включен. Одновременно пункт, бывший включенным до этого, отключается. Пункт **RGB** дополнительного меню выбирает режим задания цвета RGB, пункт **HSB** — режим HSB.

Собственно задание чисел, описывающих цвет, производится с помощью трех полей ввода, находящихся в правой части панели смесителя. Для обоих режимов RGB это поля ввода **R**, **G** и **B**, где вводятся соответственно доля красной, зеленой и синей составляющей цвета. Для режима HSB это поля ввода **H**, **S** и **B** — оттенок, насыщенность и яркость.

Ниже этих трех полей ввода находится четвертое — **Alpha**, задающее уровень прозрачности цвета. Этот уровень задается в процентах: 100% соответствуют полной непрозрачности, а 0% — соответственно полной невидимости цвета.

Если вы внимательно посмотрите на любое из этих полей ввода, то справа от него увидите небольшую кнопку со стрелкой, направленной вниз, — *регулятор*. При нажатии на кнопку появляется шкала регулятора с движком, перетаскивая мышью этот движок, можно задать нужное значение для поля ввода (рис. 6.10). Чтобы убрать шкалу регулятора, достаточно щелкнуть мышью по любому месту окна или любому другому элементу управления.

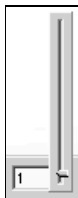


Рис. 6.10. Шкала регулятора с движком

Правее селектора цвета заливки вы видите небольшой раскрывающийся список. С его помощью задается вид заливки. В данное время нас интересуют только два пункта этого списка. Пункт **Solid** задает обычную сплошную заливку, а пункт **None** отключает ее. Остальные пункты этого списка мы рассмотрим далее в этой главе.

Если в процессе работы вы создадите какой-либо очень удачный цвет и захотите сохранить его на будущее, выберите в дополнительном меню смесителя пункт **Add Swatch**. Созданный вами цвет будет добавлен в нижнюю строку палитры селектора цвета (см. рис. 6.1), где вы сможете его в любой момент выбрать.

Работа с линиями

После рассмотрения различных средств выбора нужного цвета поговорим о том, как рисовать линии. Нет, мы не собираемся рассказывать снова о том, как проводить линии, — об этом уже, на наш взгляд, достаточно говорилось в главе 5. Мы поговорим о том, как задать вид вновь рисуемых и уже нарисованных линий. В частности, как сделать линию толще или изменить ее на штрихпунктирную.

Разумеется, прежде чем вы будете менять параметры линии, вам нужно ее выделить.

Задание параметров линий

Для задания внешнего вида уже проведенных вами линий вы можете воспользоваться редактором свойств. Он предоставляет соответствующие элементы управления, если вы выберете на рабочем листе какую-либо линию или контур. Эти элементы управления показаны на рис. 6.11.

С помощью большого раскрывающегося списка вы можете выбирать стиль линии: тонкая, толстая, штриховая, неровная, точечная и др. Левее него расположено поле ввода с уже знакомым вам регулятором, задающее толщину линии в пунктах. Установите нужные параметры, следя за выбранной вами на рабочем листе линией. С помощью верхнего селектора цвета вы можете задать цвет линии.

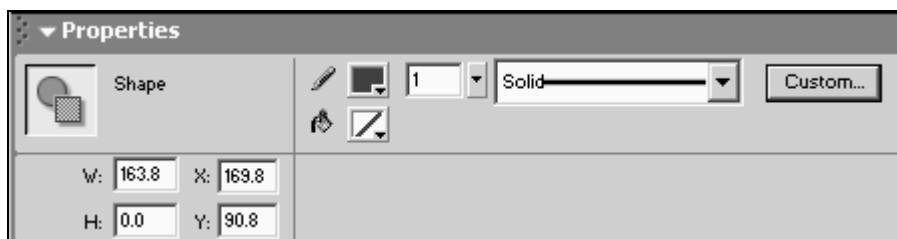


Рис. 6.11. Вид редактора свойств при выбранной линии или контуре

Если вам недостаточно всех этих настроек, можно вызвать диалоговое окно задания параметров линии. Для этого щелкните кнопку **Custom**, расположенную правее раскрывающегося списка стилей линии. На экране появится диалоговое окно **Stroke Style**, показанное на рис. 6.12.

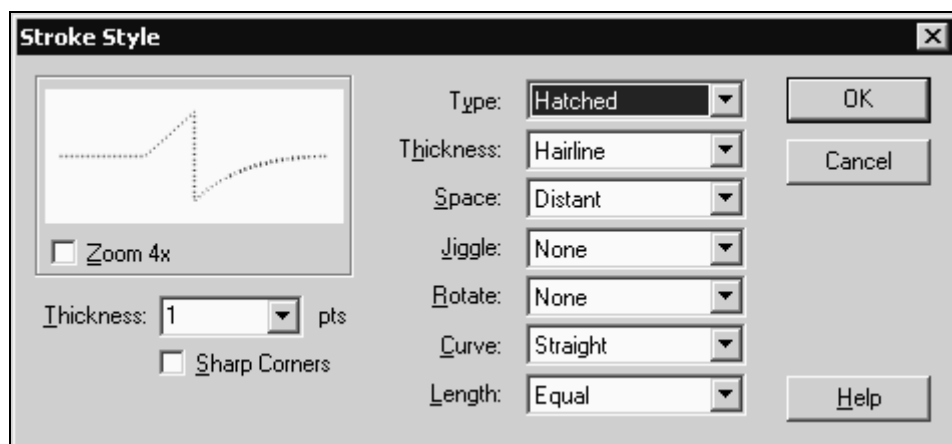


Рис. 6.12. Диалоговое окно **Stroke Style**

В верхнем левом углу этого диалогового окна находится поле предварительного просмотра. В нем всегда виден результат ваших экспериментов над линиями. Флажок **Zoom 4x** позволяет включить или отключить четырехкратное увеличение содержимого поля просмотра, если вы захотите посмотреть, из чего состоит ваша линия.

В поле ввода со списком **Thickness** вводится или выбирается толщина линии в пикселах.

Флажок **Sharp Corners** включает или отключает "заострение" концов линий. Обычно Flash, подобно бывалому дипломату, сглаживает углы.

В раскрывающемся списке **Type** выбирается тип линии:

- ☐ **Solid** — сплошная линия;
- ☐ **Dashed** — пунктирная линия;

- ☐ **Dotted** — точечная линия;
- ☐ **Ragged** — имитирующая линию, проведенную вручную;
- ☐ **Stipple** — линия, состоящая из чернильных пятен;
- ☐ **Hatched** — линия, состоящая из поперечных черточек.

Если выбран пункт **Dashed**, в диалоговом окне, ниже раскрывающего списка **Type**, появятся два дополнительных поля ввода **Dash**. В левом вводится длина штрихов, а в правом — расстояние между штрихами. Обе эти величины задаются в пунктах (не пунктах списка, а единицах измерения размера шрифта).

Если выбран пункт **Dotted**, в диалоговом окне появится одно дополнительное поле ввода **Dot Spacing**. В нем вводится расстояние между отдельными точками, измеряемое в пунктах.

Если выбран пункт **Ragged**, в диалоговом окне появятся три дополнительных раскрывающихся списка. В списке **Pattern** задается тип линии:

- ☐ **Solid** — непрерывная линия;
- ☐ **Simple** — линия, состоящая из отрезков одинаковой длины;
- ☐ **Random** — линия, состоящая из отрезков разной длины;
- ☐ **Dotted** — линия, состоящая из отрезков одинаковой длины, разделенных точками;
- ☐ **Random Dotted** — линия, состоящая из отрезков разной длины, разделенных точками;
- ☐ **Triple Dotted** — линия, состоящая из отрезков одинаковой длины, разделенных двойными точками;
- ☐ **Random Triple Dotted** — линия, состоящая из отрезков разной длины, разделенных двойными точками.

В раскрывающемся списке **Wave Height** задается "волнистость" линии. Он содержит четыре пункта: **Flat**, **Wavy**, **Very Wavy** и **Wild**, задающих соответственно полное отсутствие "волнистости", малую, среднюю и сильную "волнистость". А в раскрывающемся списке **Wave Length** задается длина получившихся "волн"; пункты **Very Short** (очень короткие "волны"), **Short** (короткие), **Medium** (средней длины) и **Long** (длинные) меняют длину от минимальной до максимальной.

Если в раскрывающемся списке **Type** выбран пункт **Stipple**, то в диалоговом окне появятся также три дополнительных раскрывающихся списка. В списке **Dot Size** задается размер пятен — от минимального до максимального — с помощью пунктов **Tiny** (крошечный), **Small** (малый), **Medium** (средний) и **Large** (большой). А с помощью всплывающего списка **Dot Variation** задается разница в размерах точек, составляющих линию. Этот список имеет четыре пункта:

- ☐ **One Size** — пятна имеют одинаковый размер;
- ☐ **Small Variation** — размер пятен немного меняется;

- ❑ **Varied Sizes** — размер пятен меняется сильно;
- ❑ **Random Sizes** — размер каждого пятна выбирается случайным образом. Практически, выбор этого пункта дает тот же самый эффект, что и выбор пункта **Varied Sizes**.

Раскрывающийся список **Density** задает, как близко друг к другу находятся отдельные пятна. Здесь доступны четыре пункта:

- ❑ **Very Dense** — пятна расположены настолько близко друг к другу, что сливаются в одну линию;
- ❑ **Dense** — пятна расположены близко друг к другу, сливаясь в множество непрерывных отрезков;
- ❑ **Sparse** — пятна расположены далеко друг от друга и не сливаются;
- ❑ **Very Sparse** — пятна расположены очень далеко друг от друга.

В случае же выбора в раскрывающемся списке **Type** пункта **Hatched** в диалоговом окне **Line Style** появятся целых шесть дополнительных раскрывающихся списков. Давайте рассмотрим их все.

Раскрывающийся список **Thickness** задает толщину штрихов, составляющих линию. Он имеет четыре пункта: **Hairline** (очень тонкие штрихи), **Thin** (тонкие), **Medium** (средней толщины) и **Thick** (толстые).

Раскрывающийся список **Space** позволяет задать расстояние между отдельными штрихами. Он также имеет четыре пункта: **Very Close** (очень близко друг к другу), **Close** (близко), **Distant** (далеко) и **Very Distant** (очень далеко).

Раскрывающийся список **Jiggle** задает сдвиг штрихов друг относительно друга, иначе говоря, их "разболтанность". Доступны четыре пункта — **None** (без сдвига), **Bounce** (небольшой сдвиг), **Loose** (средний сдвиг) и **Wild** (полнейшая "разболтанность").

Раскрывающийся список **Rotate** задает поворот штрихов друг относительно друга. Доступны четыре пункта — **None** (без поворота), **Slight** (небольшие углы поворота), **Medium** (средний поворот) и **Free** (угол поворота может достигать 90°).

Раскрывающийся список **Curve** позволяет задать искривленность штрихов. Он имеет четыре пункта: **Straight** (никакой искривленности, все штрихи прямые), **Slight Curve** (небольшая искривленность), **Medium Curve** (средняя искривленность) и **Very Curved** (большая искривленность).

Последний раскрывающийся список — **Length** — позволяет задать, насколько меняется размер соседних штрихов. Имеются четыре пункта: **Equal** (все штрихи одинаковы), **Slight Variation** (размеры изменяются в небольших пределах), **Medium Variation** (размеры меняются в средних пределах) и **Random** (размер каждого штриха выбирается случайным образом).

Задав нужный вид линии, нажмите кнопку **ОК**. Если вы передумали, нажмите кнопку **Cancel**. В любом из этих случаев диалоговое окно **Line Style** закроется. А заданные вами параметры линии будут тотчас применены.

"Пузырек с чернилами"

Для того чтобы вы могли изменить цвет и вид контура любой фигуры, щелкнув по любому ее месту, даже по заливке, Flash предлагает инструмент "пузырек с чернилами". Чтобы выбрать его, щелкните кнопку, показанную на рис. 6.13, в инструментарии или нажмите клавишу <S> клавиатуры. Курсор мыши примет вид небольшого опрокинутого пузырька с чернилами.



Рис. 6.13. Кнопка включения инструмента "пузырек с чернилами"

Чтобы изменить цвет и стиль линий контура, задайте нужные параметры с помощью редактора свойств или воспользуйтесь верхним селектором цвета в области **Colors** инструментария. После этого просто щелкните по любому месту фигуры, контур которой вы хотите изменить. Проще всего, разумеется, щелкнуть по заливке, которая в результате этого никак не изменится.

Работа с заливками

А теперь, после рассмотрения различных параметров линий, которые предлагает нам для работы Flash, поговорим о параметрах заливок. Как и в случае линии, выделите нужную заливку и...

Виды заливки

Вы уже знаете, что каждый замкнутый контур во Flash может быть заполнен так называемой заливкой. Также вы знаете, что Flash во многих случаях (в частности, при рисовании прямоугольников и эллипсов) сам создает такие заливки. И, конечно, вам уже известно, как задать цвет заливки. Об этом мы говорили еще в *главе 5*, когда рассматривали операции рисования фигур.

Но разговор о заливках еще не закончен. Сейчас мы рассмотрим еще некоторые параметры этих графических примитивов, которые вы можете задавать. И поможет нам в этом смеситель.

Flash предоставляет художнику несколько видов заливки. Все они задаются с помощью раскрывающегося списка, находящегося правее селектора цвета заливки и расположенного в панели смесителя. Рассмотрим все эти заливки подробнее.

Простейшие виды заливки

Простейшие виды заливки — это однотонная, сплошная заливка и ее отсутствие. В последнем случае опытные компьютерные художники часто говорят о так называемой *прозрачной заливке*, что в случае Flash неверно.

Чтобы отключить создание заливки при рисовании замкнутых фигур, выберите в раскрывающемся списке видов заливки пункт **None**. Это простейший вид заливки — больше вам не нужно задавать для него никаких параметров.

Чтобы включить создание однотонной заливки, выберите пункт **Solid**. В этом случае вы можете задать цвет заливки, воспользовавшись соответствующими селекторами цветов в инструментарии и на редакторе свойств. Если у вас открыт смеситель, вы можете воспользоваться им.

Градиентная заливка

Кроме простейших видов заливки, Flash может создавать и более сложные. В частности, *градиентные заливки*, в которых цвет плавно меняется от одного значения к другому, например, от белого к черному. Вы можете увидеть пример градиентной заливки, если посмотрите на любое окно любой программы, работающей в операционной системе Microsoft Windows 98 или более новой ее версии, — цвет его заголовка плавно меняется от синего к светло-голубому.

Прежде чем мы продолжим разговор о градиентных заливках (иногда их для краткости называют градиентами), давайте введем один термин. Назовем цвета, на основании которых будет создаваться градиент, *ключевыми*. Например, для заголовка окна Windows 98 ключевыми цветами будут синий и светло-голубой. Это поможет нам в дальнейшем.

Flash поддерживает два вида градиентной заливки: *линейную* и *радиальную*. В линейной градиентной заливке цвет меняется линейно при движении от одного конца фигуры к другому (рис. 6.14). В радиальной же градиентной заливке цвет меняется при движении от центра воображаемой окружности, вписанной в фигуру, к ее периферии (рис. 6.15). Как видите, названия видов закраски говорят сами за себя.

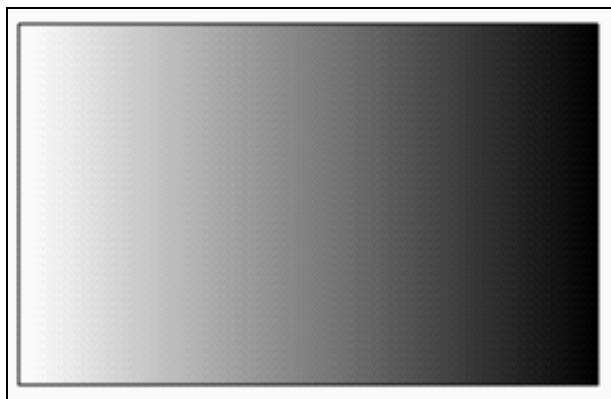


Рис. 6.14. Линейная градиентная заливка

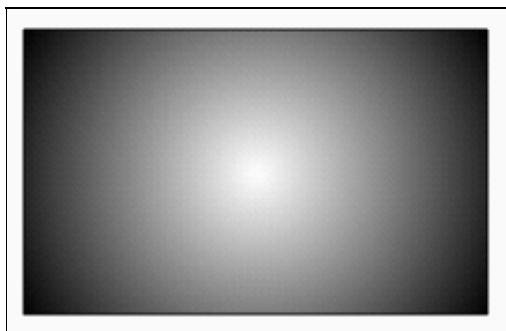


Рис. 6.15. Радиальная градиентная заливка

Чтобы создать линейную или радиальную градиентную заливку, откройте смеситель и выберите в раскрывающемся списке видов заливки соответственно пункт **Linear Gradient** или **Radial Gradient**. После этого смеситель сменит свой вид (рис. 6.16). Рассмотрим, что же предлагается нашему вниманию в этом случае.

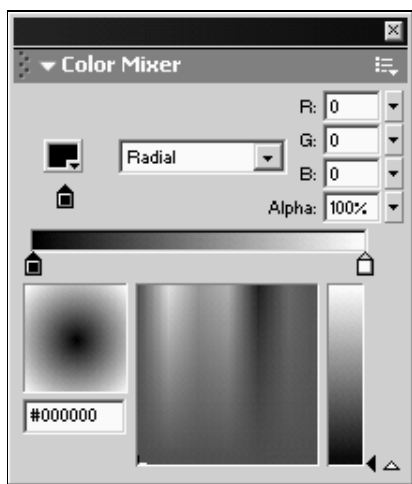


Рис. 6.16. Вид смесителя при выбранной радиальной градиентной заливке

Ниже раскрывающегося списка видов заливки находится линейка задания ключевых цветов заливки. По этой линейке перемещаются два указателя, задающие нужные цвета. Вы можете выбирать тот или иной указатель, просто щелкнув по нему мышью; выбранный указатель имеет не белую, а черную стрелку. Выбрав указатель, вы можете задать соответствующий ключевой цвет, указав его в расположенном правее селекторе цвета или воспользовавшись двумя расположенными ниже панелями. Ну и, конечно, можно перемещать указатели, "захватывая" их мышью, чтобы подобрать нужное расстояние между ключевыми цветами на результирующей заливке.

Результат всех выполняемых вами операций показывается в небольшой панели предварительного просмотра, расположенной левее и ниже линейки. Так что вы сразу сможете оценить результат своих трудов.

Задание ключевых цветов и для линейной, и для радиальной заливок выполняется одинаково. Различен только результат. Поэтому мы рассматриваем эти два вида заливки вместе.

Кроме градиентных заливок с двумя ключевыми цветами, вы можете создавать и многоцветные заливки. Пример такой заливки, содержащей три ключевых цвета, показан на рис. 6.17. Здесь белый цвет плавно изменяется до черного, а потом — снова "белеет".

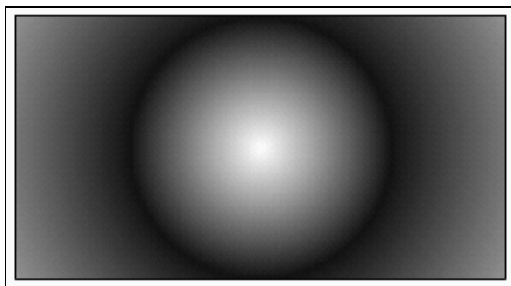


Рис. 6.17. Многоцветная градиентная заливка

Чтобы создать такую заливку, вам нужно добавить на линейку задания цвета (см. рис. 6.16) новый указатель — для промежуточного цвета. Для этого выберите нужное место на линейке и щелкните мышью чуть ниже ее, на уровне остальных указателей. На линейке тотчас появится новый указатель. Вы также можете перемещать его и изменять ключевой цвет в задаваемой им точке. Чтобы удалить созданный вами указатель, просто "сдерните" его вниз мышью. Удалить два изначальных указателя, созданных самой программой, невозможно.

Для заливки, показанной на рис. 6.17, вид линейки задания цвета будет такой, как на рис. 6.18.

Flash поддерживает градиентные заливки, имеющие до восьми цветов включительно. Так что умерьте свои аппетиты.

Может случиться так, что в процессе работы вы создадите какую-либо очень удачную заливку и захотите сохранить ее на будущее. В этом случае выберите в дополнительном меню смесителя пункт **Add Swatch**. После этого созданная вами градиентная заливка будет добавлена в нижнюю часть окна селектора цвета (см. рис. 6.1), где вы сможете ее в любой момент выбрать.



Рис. 6.18. Вид линейки задания цветов для заливки, показанной на рис. 6.17

Использование растрового изображения в качестве заливки

Кроме встроенных заливок, Flash также предлагает художнику использовать в качестве заливки уже существующее растровое графическое изображение (так называемая *графическая заливка*). Единственное требование: изображение должно быть сохранено в файле поддерживаемого Flash формата. И, конечно, нужно помнить, что растровое изображение может сильно увеличить размер конечного SWF-файла.

Чтобы использовать растровое изображение в качестве заливки, его нужно сначала импортировать во Flash. Для этого выберите пункт **Import** в меню **File** или нажмите комбинацию клавиш <Ctrl>+<R>. На экране появится стандартное диалоговое окно открытия файла Windows. Найдите нужный файл и нажмите кнопку открытия файла этого диалогового окна.

Flash, добрая душа, тут же поместит импортированное нами изображение на рабочий лист и сделает его выделенным. Вероятно, он думает, что таким образом помогает нам, но нам в данный момент не нужна такая помощь. Поэтому мы удалим импортированное изображение с листа, для чего просто нажмем клавишу . Будьте осторожны: растровые изображения бывают разные, в том числе и очень маленькие, так что смотрите внимательнее.

На этом подготовительные операции заканчиваются. Можно приступить к созданию графической заливки.

Выберите в раскрывающемся списке видов заливки, расположенном в смесителе, пункт **Bitmap**. Смеситель примет вид, показанный на рис. 6.19.

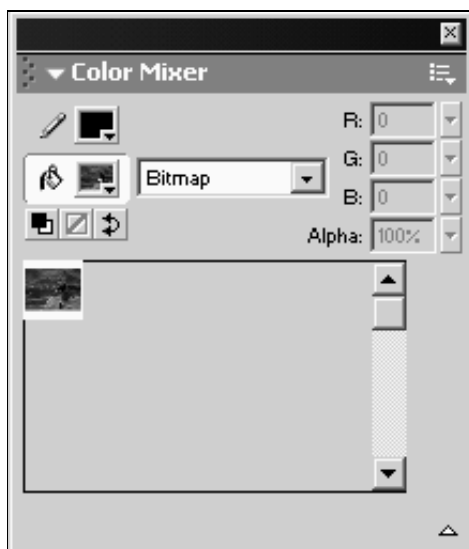


Рис. 6.19. Вид смесителя при выбранной графической заливке

Как видите, в этом случае в нижней половине панели смесителя находится большой список импортированных в документ Flash графических изображений. Вы можете выбрать любое из них, щелкнув по нему мышью. После этого можно создавать замкнутые фигуры с графической заливкой. Пример такой фигуры показан на рис. 6.20.

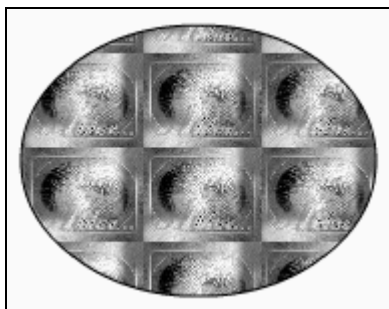


Рис. 6.20. Графическая заливка

Настройка заливки

Если вы используете градиентную или графическую заливку, Flash предоставляет вам дополнительные возможности по настройке заливки. В частности, вы можете изменять направление линейного градиента, радиус воображаемой окружности, на основе которой строится радиальный градиент, размер графического изображения, используемого в качестве заливки.

Для всего этого служит особый инструмент, называемый "трансформатором заливки". Чтобы выбрать этот инструмент, щелкните кнопку, показанную на рис. 6.21; эта кнопка, как вы уже поняли, находится в инструментарии.



Рис. 6.21. Кнопка модификатора "трансформатор заливки"

Все приведенные ниже примеры будут описаны для случая графической заливки, так как в этом случае будет доступно большинство предоставляемых Flash возможностей. Растровое изображение, используемое в качестве заливки, будем называть *изображением-заливкой*. И предположим, что это изображение-заливка заполняет контур целиком.

Чтобы настроить какую-либо заливку, щелкните по ней мышью. (При этом помните, что у вас должен быть выбран инструмент "трансформатор заливки".) После этого изображение-заливка будет выделена особым образом (рис. 6.22). Как видите, она помещается в особый *прямоугольник выделения*, на разных сторонах и в центре которого находятся особые точки, называе-

мые *маркерами*, с помощью которых вы сможете настроить заливку. Такая настройка выполняется перетаскиванием этих маркеров мышью.

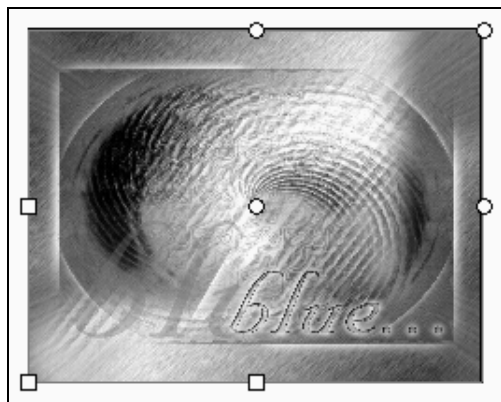


Рис. 6.22. Графическая заливка, выделенная при включенном модификаторе "трансформатор заливки"

Что же вы можете делать в этом случае с изображением-заливкой?

Во-первых, заливку можно смещать внутри заполняемого ею контура. Для этого перетаскиваете ее, "удерживая" мышью за круглый белый маркер, находящийся в центре прямоугольника выделения, — *маркер смещения*. Результат такой операции показан на рис. 6.23, заливка была смещена вправо и вверх.

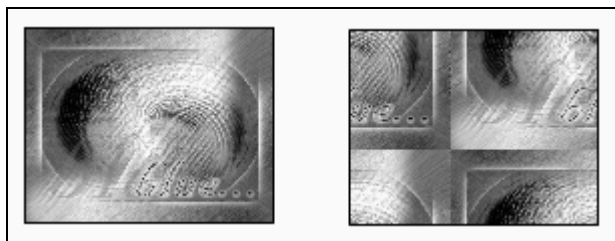


Рис. 6.23. Смещение заливки внутри заполняемого контура направо и вверх (слева — исходное изображение, справа — результат)

Во-вторых, можно изменять геометрические размеры заливки. Для этого перетаскивайте мышью один из трех больших квадратных маркеров, называемых *маркерами изменения размера*. Маркер на левой стороне прямоугольника меняет ширину заливки, а маркер на нижней стороне — его высоту. Если же вы хотите менять одновременно и ширину, и высоту прямоугольника без потери его пропорций, воспользуйтесь маркером, расположенным в его левом нижнем углу. Пример изменения размеров заливки показан на рис. 6.24, заливка была сжата по горизонтали.

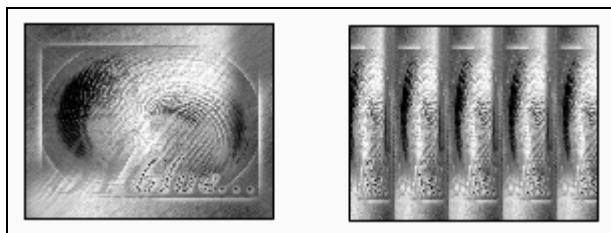


Рис. 6.24. Сжатие заливки по горизонтали (слева — исходное изображение, справа — результат)

Как видите, если размер заливки оказывается меньше размера заполняемого ей контура, то для заполнения его будет использовано несколько изображений данного размера. Опытные художники говорят в этом случае, что программа закрашивает контур мозаикой (по-английски *tile*).

В-третьих, можно вращать изображение-заливку вокруг центра, обозначенного маркером смещения. Для этого перетаскивайте мышью круглый маркер, расположенный в правом верхнем углу прямоугольника выделения, — это *маркер поворота*. Результат, достигаемый при этом, показан на рис. 6.25. Как видите, здесь тоже была использована мозаика, так как по высоте изображение оказалось уже, чем нужно, чтобы заполнить контур целиком.

В-четвертых, изображение-заливку можно сдвигать. Для этого перетащите мышью один из двух круглых *маркеров сдвига* (не путать с маркером смещения), расположенных на правой и верхней стороне прямоугольника выделения. Верхний маркер "отвечает" за сдвиг по горизонтали, правый — соответственно за сдвиг по вертикали. Возможный результат сдвига заливки показан на рис. 6.26.

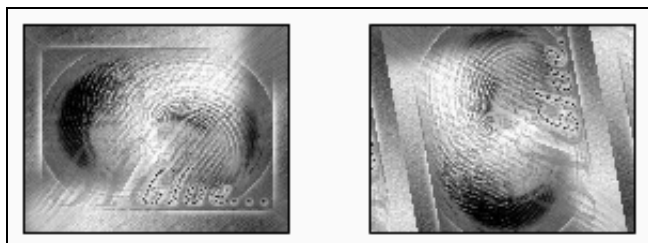


Рис. 6.25. Поворот заливки (слева — исходное изображение, справа — результат)

Многие из этих действий вы можете выполнять и над линейными градиентными заливками. А именно, смещение, изменение одного из размеров и поворот; если вы взглянете на рис. 6.27, то увидите все доступные маркеры.

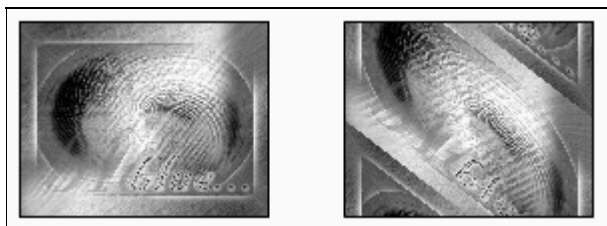


Рис. 6.26. Сдвиг заливки по вертикали (слева — исходное изображение, справа — результат)

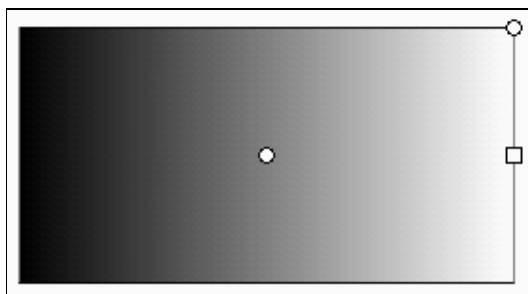


Рис. 6.27. Линейная градиентная заливка, выделенная с помощью инструмента "трансформатор заливки"

В случае же радиальной градиентной заливки все несколько сложнее. Взгляните на рис. 6.28 — и вы все поймете. Прежде всего, здесь нужно говорить не о прямоугольнике, а об *окружности выделения*. Далее, сразу не понятно, какой маркер за что "отвечает". Хорошо заметны только маркер смещения (он всегда находится в центре) и маркер изменения размера, а именно, ширины (он находится справа и имеет вид квадрата — единственный квадратный маркер из изображенных на рисунке). Остальные же маркеры мы сейчас рассмотрим.

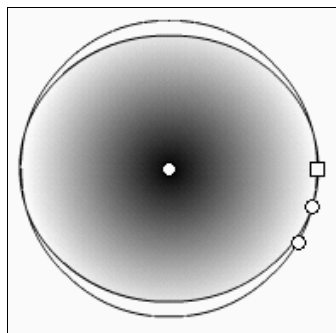


Рис. 6.28. Радиальная градиентная заливка, выделенная с помощью инструмента "трансформатор заливки"

Если от маркера изменения ширины заливки следовать по окружности по часовой стрелке, первым встретившимся нам маркером будет *маркер изменения радиуса* окружности. Не путайте его с маркером изменения ширины. Попробуйте переместить оба этих маркера и посмотрите, что получится (рис. 6.29).

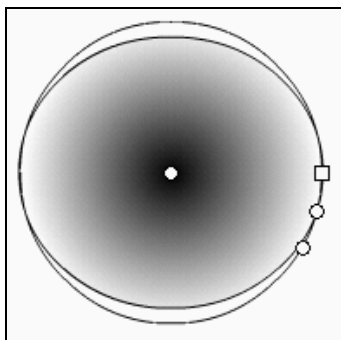


Рис. 6.29. Сжатие по ширине (слева) и уменьшение радиуса окружности (справа) заливки

Если продолжать наше воображаемое путешествие по окружности, то следующим маркером будет маркер поворота. Как им пользоваться, вы уже знаете.

Фиксация заливки

Обычно, если вы создаете градиентные или графические заливки для различных контуров, Flash делает их независимыми друг от друга. При этом считается, что каждая замкнутая фигура на листе существует сама по себе, вне зависимости от того, какие фигуры ее окружают. Это хорошо заметно на рис. 6.30.

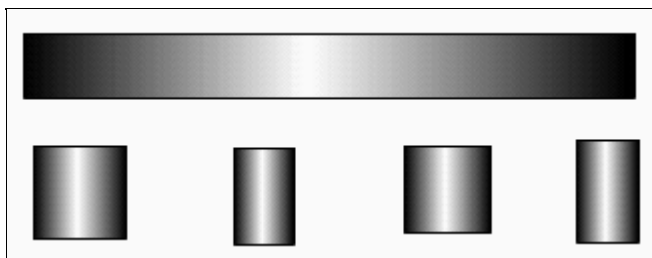


Рис. 6.30. Набор геометрических фигур с независимыми заливками

Однако иногда бывает нужно сделать набор геометрических фигур с зависимыми заливками. Например, такой, какой показан на рис. 6.31. Хорошо видно, что заливка как бы "растягивается" на несколько фигур, в то время как ее начало зафиксировано в одной из них. Такая заливка называется *фиксированной*.

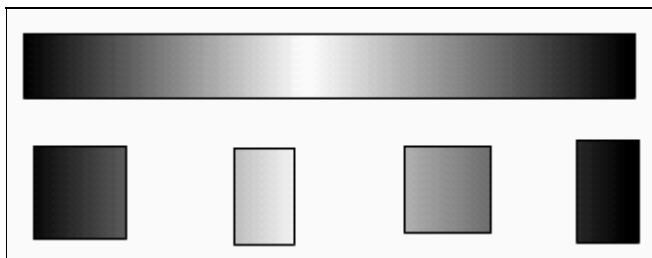


Рис. 6.31. Набор геометрических фигур с зависимыми заливками

Чтобы создать фиксированную заливку, воспользуйтесь модификатором "фиксация заливки", доступным при выборе "кисти" или "ведра с краской". Этот модификатор включается кнопкой-выключателем, показанным на рис. 6.32; кнопка эта, как и все другие кнопки модификаторов, находится в области **Options** инструментария.



Рис. 6.32. Кнопка модификатора "фиксация заливки"

Чтобы создать фиксированную заливку, выполните следующую последовательность действий.

1. Выберите инструмент "кисть" или "ведро с краской".
2. Включите или отключите другие модификаторы, если хотите.
3. Щелкните в том контуре, где хотите создать начало фиксированной заливки.
4. Включите модификатор "фиксация заливки".
5. Продолжайте щелкать в других контурах, на которые вы хотите распространить зафиксированную на предыдущем шаге заливку.
6. Закончив, обязательно отключите модификатор "фиксация заливки", если не хотите, чтобы созданная вами фиксированная заливка распространялась и на другие контуры вашего изображения.

Вот и все. Как видите, создать фиксированную заливку очень просто. А чтобы добавить ей эффектности, вы можете воспользоваться модификатором "трансформатор заливки" и изменить ее.

"Пипетка"

Иногда бывает необходимо скопировать параметры какой-либо линии или заливки и применить их к другой линии или заливке. Конечно, можно просто выставить параметры, заданные с помощью редактора свойств, вручную,

но, согласитесь, это долго и неудобно. Проще воспользоваться инструментом "пипетка", предусмотренным как раз для такого случая.

Чтобы выбрать инструмент "пипетка", щелкните кнопку, показанную на рис. 6.33, в инструментарии или нажмите клавишу <I> клавиатуры. Курсор мыши примет вид пипетки.



Рис. 6.33. Кнопка включения инструмента "пипетка"

Выбрав "пипетку", вы можете щелкнуть нужную линию или заливку, параметры которых вы хотите скопировать. Если вы щелкните линию, Flash автоматически выберет инструмент "пузырек с чернилами", если же заливку — "ведро с краской". В последнем случае также автоматически включится модификатор "фиксация заливки"; если он вам не нужен, вы можете его отключить.

Теперь, чтобы применить выбранные параметры к другой линии или заливке, вам остается только ее (линию или заливку) щелкнуть. Просто, не правда ли?

Работа с палитрами

Когда мы знакомились с селектором цветов, то называли весь доступный в его окне набор цветов палитрой. Как вы уже знаете, в эту палитру можно добавлять новые цвета и градиентные заливки, созданные с помощью смесителя. После этого вы сможете выбрать и применить в выделенной графике созданные вами цвета и заливки, воспользовавшись селектором цвета.

Настала пора подробнее поговорить о палитре и управлении ее содержимым.

Прежде всего, нужно сказать, что каждый документ Flash содержит свою собственную уникальную палитру, т. е. используемый в нем набор цветов. Эта палитра сохраняется и в файле формата Shockwave/Flash, предназначенного для распространения. Отсюда следует, что, умело подобрав палитру и, возможно, ограничив количество используемых в изображении цветов, вы можете значительно уменьшить размер этого файла.

Вся работа над содержимым палитры осуществляется в панели **Color Swatches**. Чтобы вызвать ее на экран, выберите пункт-выключатель **Color Swatches** в меню **Window** или нажмите комбинацию клавиш <Ctrl>+<F9>. Можно также выбрать пункт **Swatches** подменю **Panels** контекстного меню листа. Сама панель **Color Swatches** показана на рис. 6.34.

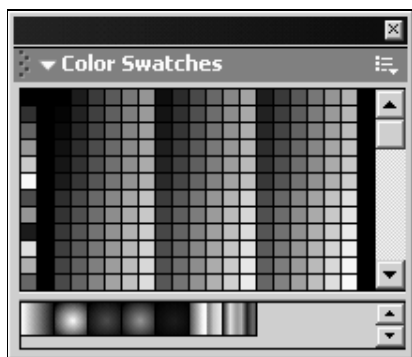


Рис. 6.34. Панель **Color Swatches**

Эта панель состоит из двух частей. В верхней ее части находится уже знакомая вам палитра цветов, в нижней — различные виды градиентных и графических заливок, как заданные по умолчанию самой программой, так и созданные пользователем. Вы можете перемещать мышью серую полосу, разделяющую эти части. Для этого установите на нее курсор мыши, нажмите левую кнопку, не отпуская ее, перетащите полосу вверх или вниз и отпустите кнопку мыши.

Выбирать любые цвета или виды заливок можно, просто щелкая по ним мышью, как вы это делали в окне селектора цветов. Выбранный цвет или вид заливки выделяется белым квадратом. Если на рабочем листе у вас была выбрана какая-либо заливка, то выбранный в панели **Color Swatches** цвет или вид заливки будет тотчас к ней применен. (Для линий это почему-то не срабатывает.)

Вы уже знаете, как добавлять в палитру новые цвета. Рассмотрим теперь, что еще можно с ними делать.

Если Flash позволяет вам добавлять в палитру новые цвета и виды заливок, то уж конечно он должен позволить вам и удалять их. В самом деле, что делать, если палитра разрастется до нечеловеческих размеров и перестанет помещаться в панели **Color Swatches**? Только удалить ненужное. А удаляются ненужные цвета и виды заливки из палитры очень просто. Для этого выделите цвет или вид заливки, который вы хотите удалить, и выберите пункт **Delete Swatch** дополнительного меню.

Кроме того, есть возможность продублировать выбранный цвет или вид заливки, чтобы в дальнейшем внести в него изменения. Для этого выделите нужный цвет или вид заливки, который вы хотите удалить, и выберите пункт **Duplicate Swatch** дополнительного меню.

Если вы создали, с вашей точки зрения, исключительно удачную палитру и хотите сохранить ее для дальнейшего использования, то можете сохранить ее на диске. Палитра сохраняется в файле с расширением `clr` (так называемые файлы цветовых наборов Flash) или `act` (цветовые таблицы). Цветовые наборы Flash могут быть использованы только в самом Flash, а цветовые таблицы

поддерживаются также другими графическими программами, в частности, Macromedia Fireworks и Adobe Photoshop. Кроме того, вы можете импортировать таблицы цветов из файлов GIF. Учтите только, что градиенты импортировать из и экспортировать в файлы цветowych таблиц и GIF нельзя.

Чтобы сохранить цветовую палитру в файле, выберите пункт **Save Colors** дополнительного меню. На экране появится стандартное диалоговое окно сохранения файла Windows. Задайте имя файла, выберите его формат и нажмите кнопку сохранения.

Также вы можете сохранить созданную вами палитру в качестве *палитры по умолчанию*, которая будет использоваться во всех вновь создаваемых документах Flash. Для этого выберите пункт **Save as Default** дополнительного меню.

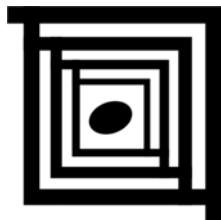
В дальнейшем сохраненные вами палитры можно снова загрузить. Чтобы загрузить палитру из файла, затерев существующую, выберите в дополнительном меню пункт **Replace Colors**. Если же вы хотите добавить цвета из загружаемой палитры в существующую палитру, выберите пункт **Add Colors**. В обоих этих случаях на экране появится стандартное диалоговое окно открытия файла Windows. Выберите формат файла, сам файл и нажмите кнопку открытия. Таким образом вы можете импортировать палитры из файлов цветowych наборов Flash, цветowych таблиц и файлов GIF.

Чтобы загрузить палитру по умолчанию, выберите в дополнительном меню пункт **Load Default Colors**. А чтобы удалить из палитры все цвета, оставив только черный и белый, и все виды заливки, кроме простейшего черно-белого линейного градиента, выберите пункт **Clear Colors**.

Пункт **Web 216** дополнительного меню позволит вам загрузить так называемую *безопасную палитру цветов WWW*. Эта палитра включает в себя 216 цветов, которые гарантированно будут отображаться на любом компьютере. Если вы создаете изображения для публикации в Интернете, то лучше всего использовать именно эту палитру.

Последний пункт дополнительного меню **Sort by Color** позволит вам отсортировать цвета в палитре по оттенку. К сожалению, вернуть обычный порядок сортировки цветов после этого невозможно.

Глава 7



Работа с текстом

В этой главе мы поговорим о возможностях, предоставляемых Flash для работы с текстовыми данными.

В самом деле, не все же нам рисовать. Посмотрите на любую Web-страницу: сколько места занимают на ней тексты и сколько — графика? Подавляющее превосходство текстов налицо, более того, несмотря на то, что графика пришла в Интернет уже достаточно давно, она все еще находится в меньшинстве. И это правильно: с помощью текстов можно описать все что угодно, а графика — увы! — не так гибка и исчерпывающа. Попробуйте, например, нарисовать курс валюты — и вы это поймете.

Вы скажете: зачем вообще было вносить во Flash средства поддержки текста? Разве недостаточно старого доброго языка HTML, с помощью которого создаются обычные Web-страницы, чтобы закодировать и поместить в Сеть любой текст? В большинстве случаев, вполне достаточно, если текст достаточно прост, а дизайн Web-страницы не запредельно сложен. Текст, выведенный разными шрифтами, с разным выравниванием, отступами и выступами, таблицы, рисунки, а, при определенном навыке и нехитрых ухищрениях, многоколоночный, с точным форматированием — все это можно сделать средствами HTML. И делают, кстати, без всякого Flash.

Так зачем Flash поддерживает ввод текста? И не только Flash, но и многие другие графические программы: Adobe PhotoShop, Corel DRAW!, да и стандартно поставляемый в составе Windows растровый редактор Paint. Зачем вообще в графическом изображении нужен текст?

Это нужно, чтобы сделать текст частью изображения. Например, вписать его в какую-нибудь геометрическую фигуру. Так, очень часто вписывают текст в окружность или сложную кривую. Также это может понадобиться для создания каких-либо специальных эффектов, например, тени под буквами или придания тексту полупрозрачности. Без этого не обойтись, если вы захотите создать какой-нибудь умопомрачительный дизайн для вашего текстового документа, — HTML в этом случае бессилен. Да даже если вы хотите, чтобы ваш текст выводился редким и хитрым шрифтом, который есть только у вас, вам также

придется оформлять ваш документ как графическое изображение, профессиональные Web-художники так и делают. Ну и, конечно, это нужно, если вы в дальнейшем будете делать видеоролик, "главным героем" которого будет текст.

Вот поэтому все графические программы, даже самые простые (как, например, Microsoft Paint) поддерживают ввод текста. И Flash не исключение.

И об этом пойдет разговор в данной главе.

Текстовые блоки

Текст, помещаемый на рабочий лист, находится в так называемом *текстовом блоке*. Это особый вид графических примитивов, служащий именно для размещения текста. (Опытные компьютерщики такие "вместилища" называют *контейнерами*; в частности, текстовый блок — контейнер для текста; по аналогии, рабочий лист — контейнер для графики.) Блок текста имеет прямоугольную форму, собственно текст помещается внутри его границ.

Работа с текстовыми блоками

Создание текстовых блоков выполняется с помощью инструмента "текст". Чтобы выбрать его, щелкните в инструментарии кнопку, показанную на рис. 7.1, или нажмите на клавиатуре клавишу <T>. После этого курсор мыши примет вид небольшого крестика с буквой "А".



Рис. 7.1. Кнопка включения инструмента "текст"

Текстовый блок создается примерно так же, как прямоугольник. Поместите курсор мыши в том месте, где у вас будет его левый верхний угол, и нажмите левую кнопку мыши. Затем, не отпуская эту кнопку, протащите мышью в то место, где у вас будет находиться правый нижний угол текстового блока. Пока вы буксируете мышью, Flash будет отображать синий "резиновый" прямоугольник, так что вы всегда будете видеть, что же у вас получается. Переместив курсор мыши в нужную точку, отпустите левую кнопку мыши — текстовый блок будет нарисован, и внутри него появится текстовый курсор (рис. 7.2). Наберите нужный текст и либо создайте новый текстовый блок, либо смените инструмент на "стрелку выделения".

Для переноса текста на новую строку при наборе текста вы можете использовать клавишу <Enter>. Также вы можете редактировать текст, пользуясь клавишами-стрелками, <Backspace>, и др.

Пользуясь мышью или обычными, используемыми в других программах, клавиатурными комбинациями, можно выделять фрагменты текста и рабо-

тать с буфером обмена Windows. В частности, вы можете выделять слова двойным щелчком мыши и выделять все содержимое текстового блока нажатием клавишной комбинации <Ctrl>+<A>.

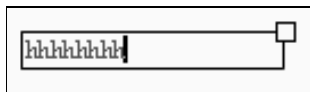


Рис. 7.2. Текстовый блок с фиксированной шириной

Это означает также, что вы можете перенести текст из другого Windows-приложения, например, Microsoft Word. В случае достаточно большого текста это может быть более удобным, чем набор его прямо во Flash, т. к. Word содержит, кроме всего прочего, систему проверки орфографии, которая, несмотря на все свои недостатки, все же может принести пользу.

Вы, вероятно, уже заметили, что при наборе текста созданный нами текстовый блок растягивается только по вертикали. Если текст подходит к правой границе текстового блока, то переносится на следующую строку, для чего блок растягивается вниз. Горизонтальный же размер (ширина) остается постоянным, таким, каким мы его задали при создании текстового блока. Подобный текстовый блок называется *блоком с фиксированной шириной* (см. рис. 7.2).

Вы также можете, выбрав инструмент "текст", просто щелкнуть по листу мышью. В этом случае будет создан *текстовый блок с фиксированной высотой* (рис. 7.3). При наборе текста этот блок будет сам растягиваться или сужаться по горизонтали, чтобы вместить весь набранный в нем текст и не "захватить" ничего лишнего.



Рис. 7.3. Текстовый блок с фиксированной высотой

Текстовые блоки можно выделять щелчком мыши и перемещать по листу, как и любые другие графические фрагменты. Будьте внимательны: когда текстовый блок не выделен, отображается только помещенный в него текст, но не его границы. Чтобы увидеть границы текстового блока, выделите его (рис. 7.4).



Рис. 7.4. Текстовый блок, выделенный с помощью инструмента "стрелка выделения"

Вы можете редактировать текст, помещенный в текстовый блок, пользуясь приемами и командами, хорошо знакомыми вам по обычным текстовым редакторам. Чтобы получить возможность редактировать текст в текстовом блоке, сделайте следующее:

- если в данный момент выбран инструмент "стрелка выделения", дважды щелкните по нужному текстовому блоку;
- если выбран инструмент "текст", щелкните по текстовому блоку один раз или выберите пункт **Edit Selected** меню **Edit**.

После этого текстовый блок примет вид, показанный на рис. 7.2 или 7.3, в зависимости от того, какого он типа. И вы сможете редактировать помещенный в него текст.

В верхнем правом углу расположен *маркер изменения размера*, имеющий вид либо окружности (для блока с фиксированной высотой), либо полого квадрата (для блока с фиксированной шириной). Вы можете изменять ширину блока с фиксированной шириной, перетаскивая его мышью. Существует возможность менять вид текстового блока, т. е. превращать блок с фиксированной шириной в блок с фиксированной высотой, и наоборот. Для этого служит двойной щелчок по маркеру изменения размера.

Форматирование

Текстовые возможности Flash отнюдь не исчерпываются набором простого текста, разбитого на абзацы с помощью клавиши <Enter>. Ваш текст может содержать фрагменты, набранные разными шрифтами, с разным выравниванием и отступами. Фактически, текстовые возможности Flash равны аналогичным возможностям языка HTML, а то и превосходят их.

Форматирование текста

Форматирование текста во Flash, а именно, задание шрифта и цвета, выполняется с помощью все того же редактора свойств. Его вид при выделенном текстовом блоке показан на рис. 7.5. Давайте рассмотрим "ответственные" за форматирование текста элементы управления сверху вниз.

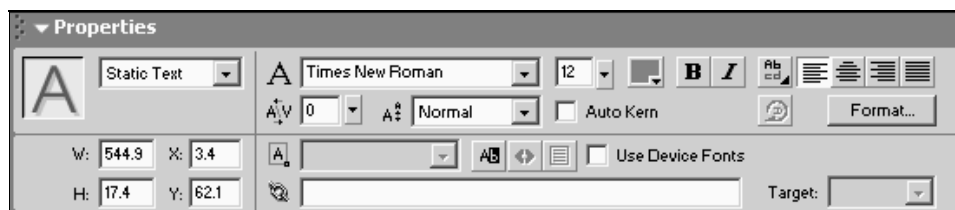


Рис. 7.5. Вид редактора свойств при выделенном текстовом блоке

Сразу заметен раскрывающийся список шрифтов, который позволяет вам выбрать собственно шрифт. В этом списке перечислены все шрифты, установленные в системе. Когда вы выбираете в раскрытом списке шрифт, левее в небольшом окошке для вашего удобства показывается пример текста, набранного этим шрифтом.

Правее этого списка находится поле ввода с регулятором, с помощью которого задается размер (или, как говорят полиграфисты, *кегель*) шрифта в пунктах. Правее же поля ввода размера находятся селектор цвета шрифта, а также кнопки **B** (включение или отключение "жирности" шрифта) и **I** (включение или отключение курсивного начертания). Все это знакомо вам по текстовым редакторам.

Чтобы изменить какие-либо параметры всего текста в текстовом блоке или его фрагмента, вам нужно будет выделить этот текст или фрагмент. Это выполняется так же, как и в других программах редактирования текста. Если же вы хотите изменить форматирование всего текста в текстовом блоке, просто выделите текстовый блок одиночным щелчком мыши при выбранном инструменте "стрелка выделения".

Внимание!

Flash поддерживает векторные шрифты стандартов TrueType и Type 1.

Для выбора шрифта вы можете воспользоваться подменю **Font** меню **Text**. Там перечислены все установленные в системе шрифты. Аналогично выбрать размер шрифта можно, воспользовавшись подменю **Size** меню **Text**, — там перечислены наиболее употребительные размеры шрифтов. Это может быть полезно, если редактора свойств нет на экране.

В том же меню (**Text**) есть пункт **Style**, позволяющий задать и другие параметры шрифта. В частности:

- ☐ пункт **Bold** (или комбинация клавиш <Ctrl>+<Shift>+) включает или отключает "жирность" шрифта;
- ☐ пункт **Italic** (или комбинация клавиш <Ctrl>+<Shift>+<I>) включает или отключает курсивное начертание.

Оба этих пункта работают как выключатели. А пункт **Plain** (и комбинация клавиш <Ctrl>+<Shift>+<P>) позволяют вернуть шрифту обычный вид.

Но вернемся к редактору свойств.

Ниже и левее списка шрифтов находится другое поле ввода с регулятором. С его помощью задается так называемый *трекинг*, т. е. дополнительное пространство между символами. Это иногда бывает очень полезно, например, для выделения заголовков — просто задайте для шрифта заголовка большой трекинг, чтобы буквы отстояли далеко друг от друга.

Значение трекинга вводится в пунктах, причем можно вводить как положительные, так и отрицательные значения. В первом случае символы текста раздвигаются, а во втором — сближаются. При этом главное — не переборщить с трекингом, иначе символы или "разбредутся" слишком далеко друг от друга, перестав быть единым текстом, или сольются в точку.

Задавать значения трекинга вы также можете, используя подменю **Tracking** меню **Text**. Пункт **Increase** (и комбинация клавиш <Ctrl>+<Shift>+<→>) увеличивает трекинг, а пункт **Decrease** (и комбинация клавиш <Ctrl>+<Shift>+<←>) — уменьшает. Пункт **Reset** (и комбинация клавиш <Ctrl>+<Shift>+<↑>) позволяет вернуть значение трекинга по умолчанию.

Также в редакторе свойств находится флажок **Kern**. Этот флажок включает или отключает *кернинг* — специальное управление пространством между символами. Дело в том, что некоторые пары символов требуют задания различных значений промежутка между ними, например, промежуток между "А" и "W" должен быть меньше, чем между "А" и "Н". Именно этим и занимается кернинг, благодаря ему текст выглядит более ровным.

Правее поля задания трекинга расположен еще один раскрывающийся список. Он задает положение символов относительно *базовой линии* текста — воображаемой линейке, по которой "пишется" текст. В нем доступны три пункта:

- ☐ **Normal** — обычное положение текста, когда он "лежит" на базовой линии;
- ☐ **Superscript** — верхний индекс;
- ☐ **Subscript** — нижний индекс.

Вы также можете воспользоваться пунктами-выключателями **Superscript** и **Subscript** подменю **Style** меню **Text**. Первый превращает текст в верхний индекс, второй — в нижний.

Вдоль нижнего края редактора свойств находится поле ввода **URL**. Это поле ввода позволит превратить выделенный фрагмент текста в гиперссылку. Для этого просто введите в него нужный интернет-адрес. На рис. 7.6 показана созданная таким образом гиперссылка.



Рис. 7.6. Гиперссылка, созданная средствами Flash

Правее поля ввода **URL** находится раскрывающийся список **Target**. С его помощью задается так называемая *цель гиперссылки*. То есть, будет ли Web-страница, на которую указывает гиперссылка, загружаться в отдельное окно Web-обозревателя или в определенный фрейм. Этот список содержит четыре пункта:

- ☐ **_blank** — Web-страница загрузится в отдельное окно Web-обозревателя;

- ❑ **_parent** — Web-страница загрузится в родительский набор фреймов;
- ❑ **_self** — Web-страница загрузится в текущий фрейм (значение по умолчанию);
- ❑ **_top** — Web-страница загрузится в текущее окно Web-обозревателя, заменив собой весь набор фреймов, если он есть.

Кроме того, вы сами можете ввести в этот список другое значение. Таким значением может быть имя фрейма, в который должна загрузиться Web-страница.

Внимание!

Вы можете превратить в гиперссылку только текст, находящийся в горизонтальном текстовом блоке.

Поддержка шрифтов во Flash

А теперь самое время поговорить о поддержке различных шрифтов во Flash.

Сразу скажем, что для создания текстовых блоков Flash позволяет использовать только векторные шрифты, т. е. шрифты формата TrueType и Adobe Type 1. Любые растровые шрифты, даже если они и установлены в системе (а они наверняка установлены, ведь большинство системных шрифтов Windows — растровые), игнорируются и не показываются в списке шрифтов редактора свойств (см. рис. 7.5).

Когда вы экспортируете готовое изображение в формат Shockwave/Flash, то Flash фактически сохраняет в результирующем SWF-файле все использованные в изображении шрифты. Этот процесс называется *внедрением* шрифтов. Благодаря этому проигрыватель Flash сможет вывести текст, набранный этими шрифтами, даже если они не установлены на компьютере пользователя. Таким образом, вам не нужно ограничивать себя небольшим набором стандартных шрифтов или распространять недостающие шрифты отдельно, чтобы пользователь увидел ваше изображение в первозданном виде.

Внимание!

Сказанное выше относится только к файлам Shockwave/Flash. Если же вы передадите кому-то другому исходный документ Flash (FLA-файл), и у этого "другого" не окажется нужного шрифта, изображение исказится, т. к. Flash будет вынужден использовать один из тех шрифтов, что есть на его компьютере. Поэтому позаботьтесь о том, чтобы у вашего коллеги оказались все нужные для работы шрифты. Есть, правда, еще один способ решить проблему недостающих шрифтов — подстановка — но о ней мы поговорим в конце этой главы.

Конечно, независимость от конфигурации системы пользователя — это хорошо. Но, к сожалению, эта независимость достается дорогой ценой. Если

вы использовали в своем изображении слишком много разных шрифтов или один очень сложный шрифт, результирующий файл Shockwave/Flash может сильно вырасти в размерах. Таким образом, принцип "все свое несу с собой" может выйти боком.

Из этой ситуации есть два выхода.

Выход первый, спартанский. Вы должны умерить свои аппетиты. Иначе говоря, использовать в создаваемых изображениях небольшой набор относительно простых шрифтов. В результате файл Shockwave/Flash хоть и "раздуется", но ненамного, и пользователи останутся довольны. Кстати, вы можете выбирать, какие именно символы шрифта будут экспортированы в результирующий файл, об этом читайте в конце данной главы.

Выход второй, компромиссный. Вы должны будете использовать в своих творениях только так называемые *шрифты-псевдонимы* Flash. (В терминологии Flash они названы *device fonts*, т. е. шрифты устройства.) Это шрифты `_sans`, `_serif` и `_typewriter`. Чтобы задать их для текста, выберите соответствующий пункт в раскрывающемся списке шрифтов редактора свойств (они расположены в начале этого списка).

На самом деле, таких шрифтов нет, т. е. они не существуют на жестком диске компьютера в виде файлов. Это действительно своего рода псевдонимы для реальных шрифтов, которые Flash автоматически подберет из имеющихся на клиентском компьютере. В частности:

- ☐ `_sans` обозначает все шрифты без засечек, например, Arial или Helvetica;
- ☐ `_serif` обозначает шрифты с засечками, например, Times New Roman;
- ☐ `_typewriter` обозначает моноширинные шрифты, наподобие Courier или Lucida Console.

Разработчики Flash настоятельно рекомендуют для набора обычного (информативного, а не декоративного) текста использовать именно шрифты-псевдонимы. Более того, утверждается, что на малых кеглях (10 пунктов и менее) текст, набранный шрифтами-псевдонимами, выглядит лучше, чем набранный обычными шрифтами. К тому же, шрифты-псевдонимы не помещаются в результирующий файл Shockwave/Flash, что само по себе является большим достоинством. Недостатком может служить только ограниченность в изобразительных средствах, но для информативного текста это не бог весть какая проблема.

И еще по поводу обычных шрифтов. Мы уже сказали, что информация о них помещается в результирующий файл. Но не для всех шрифтов это справедливо. Включите пункт **Antialias Text** в меню **View** и присмотритесь к текстовым блокам. Если шрифт текста какого-то из них имеет зазубренный вид, он не будет правильно экспортирован в SWF-файл, т. к. не будет распознан Flash. Вы должны поменять такой шрифт на другой, например, на шрифт-псевдоним.

Форматирование абзаца

Форматирование целого текстового абзаца, а именно, задание выравнивания и отступов, выполняется с помощью все того же редактора свойств. Для этого используются другие элементы управления, не рассмотренные нами ранее. Но сейчас-то мы их рассмотрим.

Чтобы отформатировать какой-либо абзац, сначала поставьте в него текстовый курсор. (Для этого сначала выберите инструмент "текстовый блок".) Вы также можете выделить какой-либо фрагмент текста в этом абзаце — это не сыграет никакой роли.

Выравнивание текста в абзаце задается с помощью находящегося в верхнем правом углу редактора свойств набора из четырех кнопок-переключателей. Также вы можете воспользоваться пунктами подменю **Align** меню **Text**.

Перечислим все кнопки этого набора слева направо:

- ☐ первая кнопка задает выравнивание по левому краю (пункт **Align Left** подменю **Align** меню **Text** или комбинация клавиш <Ctrl>+<Shift>+<L>);
- ☐ вторая — выравнивание по центру (пункт **Align Center** или комбинация клавиш <Ctrl>+<Shift>+<C>);
- ☐ третья — выравнивание по правому краю (пункт **Align Right** или комбинация клавиш <Ctrl>+<Shift>+<R>);
- ☐ четвертая — полное выравнивание по обоим краям (пункт **Justify** или комбинация клавиш <Ctrl>+<Shift>+<J>).

Ниже этого набора кнопок находится кнопка **Format**. При нажатии этой кнопки на экране появится небольшое диалоговое окно **Format Options**, с помощью которого вы можете задать другие параметры абзаца. Это диалоговое окно показано на рис. 7.7.

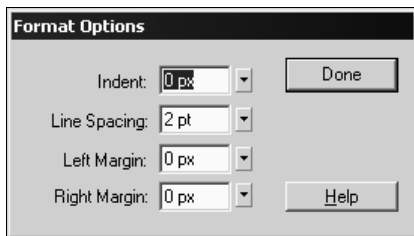


Рис. 7.7. Диалоговое окно **Format Options**

Поле ввода **Indent** задает значение отступа красной строки, т. е. расстояние между левой границей текстового блока и первым символом первой строки текста абзаца. Это значение задается в пикселах.

Поле ввода **Line Spacing** задает расстояние между строками по вертикали. Оно вводится в пунктах.

Поля ввода **Left Margin** и **Right Margin** задают отступ слева (левое) и справа (правое) текста от границ текстового блока. Эти значения задаются в пикселах.

Задав параметры абзаца в диалоговом окне **Format Options**, нажмите кнопку **Done**, чтобы закрыть его. После закрытия этого окна заданные вами параметры будут сразу же применены.

Параметры текстового блока

Задавать параметры можно не только для фрагмента текста или отдельного абзаца, но и для всего текстового блока. В частности, вы можете сделать текстовый блок вертикальным или дать возможность пользователю выделять находящийся в нем текст.

Чтобы сделать текстовый блок вертикальным, нажмите кнопку, находящуюся левее набора кнопок, задающих выравнивание. При этом на экране появится небольшое меню (рис. 7.8), в котором вы сможете выбрать нужный пункт. Всего таких пунктов три:

- ☐ **Horizontal** — горизонтальный текстовый блок (значение по умолчанию);
- ☐ **Vertical, Left to Right** — вертикальный текстовый блок с направлением текста слева направо;
- ☐ **Vertical, Right to Left** — вертикальный текстовый блок с направлением текста справа налево.

Для европейских языков с направлением текста слева направо второй и третий пункты меню равнозначны. Вы можете выбирать любой. Пример вертикального текстового блока показан на рис. 7.9.

Ниже кнопки задания направления находится кнопка поворота текста. Изначально в вертикальном блоке текста буквы расположены горизонтально, т. е. одна над другой. Если вы нажмете кнопку поворота (она работает как кнопка-выключатель), то буквы текста будут повернуты на 90° (рис. 7.10).

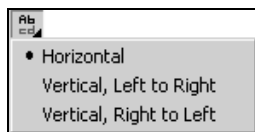


Рис. 7.8. Кнопка и раскрытое меню задания направления текста в текстовом блоке

Включение флажка **Use Device Fonts** заставляет Flash использовать для отображения текста, введенного в текстовом блоке, шрифты-псевдонимы. Включение флажка **Use Device Fonts** аналогично выбору любого шрифта-псевдонима в раскрывающемся списке шрифтов редактора свойств (см. рис. 7.5), но затрагивает не выделенный фрагмент текста, а весь текст в текстовом блоке.

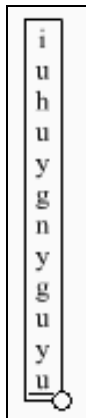


Рис. 7.9. Вертикальный текстовый блок (выделен для правки текста)

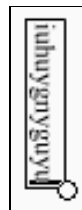


Рис. 7.10. Буквы текста в вертикальном текстовом блоке повернуты на 90°

Если включена кнопка-выключатель **Selectable** (рис. 7.11), пользователь может выделять и копировать в буфер обмена текст, помещенный в текстовый блок. По умолчанию кнопка **Selectable** отключена, т. е. пользователь не может выделять текст. (Это, кстати, может быть полезно, если вы хотите защитить какие-либо тексты от несанкционированного копирования.)



Рис. 7.11. Кнопка-выключатель **Selectable**

Специальные текстовые блоки

А теперь давайте поговорим о так называемых специальных текстовых блоках, т. е. о текстовых блоках, выполняющих особые функции.

Поля ввода

Что такое *поле ввода*, вы должны знать. Это своего рода небольшой текстовый редактор, помещенный в окно Windows-программы и предназначенный для ввода одно- или многострочного текста. При этом Windows сама управляет текстовым вводом: вы просто набираете символы, пользуетесь клавишами-стрелками, клавишами <Backspace>, , выделяете текст, пользуетесь буфером обмена, как и в "большом" текстовом редакторе. Программисты фирмы Microsoft много поработали над тем, чтобы сделать поля ввода по-настоящему удобными.

Также вы знаете, что поля ввода поддерживаются языком HTML. Вы можете поместить на своей Web-странице форму с полем ввода, в котором посетитель вашего сайта должен будет ввести, например, свое имя или идентификационный код. Такие поля ввода также предоставляют пользователю базовые возможности редактирования текста (правда, это зависит от операционной системы, под которой работает программа Web-обозревателя).

Flash также поддерживает создание полей ввода. Давайте рассмотрим, как это делается. Однако предупредим сразу, что эта информация дается "на вырост". Сейчас мы рассмотрим только создание полей ввода, как обрабатывать введенные в них данные, вы узнаете в *части 3* этой книги.

Итак, прежде всего нам нужно создать обычный текстовый блок. Переключитесь на инструмент "стрелка выделения" и выделите его. После этого обратитесь к редактору свойств, найдите в его верхнем левом углу раскрывающийся список и выберите в нем пункт **Input Text**. Редактор свойств примет вид, показанный на рис. 7.12.

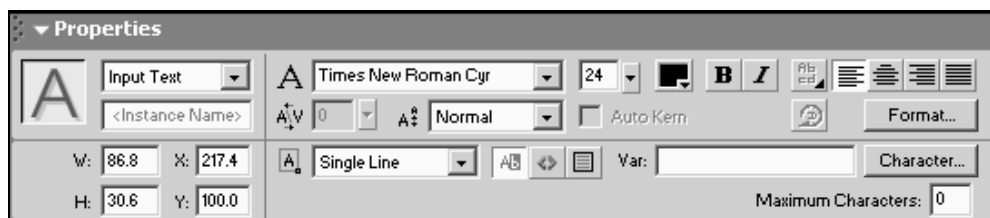


Рис. 7.12. Вид редактора свойств при выделенном поле ввода

Нижее этого списка находится поле ввода, в котором вам нужно будет указать уникальное *имя*, идентифицирующее созданное вами поле ввода Flash. Каждое поле ввода, созданное вами на рабочем листе, должно иметь имя.

Нижее и правее поля ввода имени находится другой раскрывающийся список. С его помощью задается вид поля ввода. В этом списке доступны четыре пункта:

- ☐ **Single Line** — обычное однострочное поле ввода;
- ☐ **Multiline** — многострочное поле ввода, иначе говоря, область редактирования;
- ☐ **Multiline no wrap** — многострочное поле ввода, в котором не будет выполняться автоматический перевод строки (хотя пользователь все же может перенести строку вручную, нажав клавишу <Enter>);
- ☐ **Password** — поле ввода пароля. Аналогично обычному однострочному полю ввода, но скрывает введенный в него текст, отображая вместо любого символа звездочки.

Правее этого списка находится набор кнопок-выключателей. Самая левая из этих кнопок вам уже знакома — это кнопка **Selectable** (см. рис. 7.11); как видите на рис. 7.12, она нажата и недоступна, т. е. не может быть "отжата". Остальные две показаны на рис. 7.13.



Рис. 7.13. Кнопки-выключатели **Render Text as HTML** и **Show Border Around Text**

Кнопка **Render Text as HTML**, будучи включенной, предписывает Flash обрабатывать все встретившиеся в тексте теги HTML. Если вы заключили фрагмент текста в тег ``, Flash выделит его полужирным шрифтом. Ниже перечислены все теги HTML, поддерживаемые Flash:

- ☐ `<A>` (гиперссылка);
- ☐ `` (полужирный шрифт);
- ☐ `` (задание цвета шрифта);
- ☐ `` (задание шрифта);
- ☐ `` (задание размера шрифта);
- ☐ `<I></I>` (курсив);
- ☐ `<P></P>` (отдельный абзац текста в многострочном поле ввода);
- ☐ `<U></U>` (подчеркивание).

Вообще-то, эта возможность полезна только, если вы задаете содержимое поля ввода из сценария ActionScript, так что описана она также "на вырост". Если же кнопка **Render Text as HTML** выключена, Flash игнорирует любые теги HTML, встретившиеся в тексте.

Кнопка **Show Border Around Text** включает или отключает показ границы и фона поля ввода. Если она отключена, поле ввода имеет "плоский" вид, если же включена, то напоминает стандартное поле ввода Windows. Если поле ввода является частью оригинального интерфейса, лучше оставьте эту кнопку отключенной. Если же вы хотите сделать поле ввода более привычным для пользователя или как-то выделить и сделать его заметнее, можете включить эту кнопку.

Правее этих кнопок расположено поле ввода переменной, в которую будет помещен введенный в созданное нами поле ввода текст. Переменные рассматриваются в *главе 21*. А пока что скажем, что это своего рода "ячейка камеры хранения" данных, созданная в памяти компьютера и имеющая уникальное имя, по которому ее можно однозначно определить. Вы можете обратиться к этой переменной по имени, извлечь из нее данные или поместить новые.

В поле ввода **Maximum Characters** задается максимальное количество символов, которое в него сможет вводить пользователь. Если вы не хотите ограничивать это количество, введите 0.

Вы уже знаете, что при экспорте изображения Flash помещает в файл Shockwave/Flash описания всех использованных в изображении шрифтов. Благодаря этому проигрыватель Flash может отобразить это изображение на любом компьютере, даже если нужные шрифты на нем не установлены. Однако это может сильно увеличить размер SWF-файла. Есть, конечно, два компромиссных способа решения проблемы, которые были описаны ранее в этой главе, но сейчас речь не об этом.

Дело в том, что Flash позволяет вам выбрать символы использованного в поле ввода шрифта, описания которых будут внедрены в изображение Flash при его экспорте. Благодаря этому вы можете внести в результирующий файл только те символы шрифта, которые действительно нужны для вывода текста, а значит, уменьшить размер этого файла до необходимого минимума. Выбор символов выполняется в диалоговом окне **Character Options** (рис. 7.14), которое появляется на экране при нажатии кнопки **Character** редактора свойств.

С помощью набора переключателей **Embed Font Outlines For** вы можете задать, описания каких символов шрифта будут помещаться в файл Shockwave/Flash. В этом наборе доступны три переключателя:

- ☐ **No Characters** — в файл Shockwave/Flash вообще не будут помещаться описания символов шрифта. Тогда проигрывателю Flash придется искать эти шрифты на компьютере клиента; если же их не окажется, то проигрыватель подставит один из имеющихся шрифтов, в результате чего изображение может исказиться;
- ☐ **All Characters** — в файл Shockwave/Flash помещаются описания всех символов шрифта;
- ☐ **Only** — в файл Shockwave/Flash помещаются описания только избранных символов шрифта.

Если был выбран переключатель **Only**, становится доступным набор флажков, расположенный ниже. Всего в этом наборе имеется четыре флажка:

- ☐ **Uppercase Letters (A-Z)** — помещаются описания больших латинских букв;
- ☐ **Lowercase Letters (a-z)** — помещаются описания малых латинских букв;
- ☐ **Numerals (0-9)** — помещаются описания цифр;
- ☐ **Punctuation (!@#\$%...)** — помещаются описания знаков пунктуации и специальных символов.

Ниже этих четырех флажков находится поле ввода. В нем вы можете вручную ввести символы, описания которых должны быть помещены в файл Shockwave/Flash. Похоже, это единственный способ поместить в него опи-

сания символов кириллицы, кроме включения флажка **All Characters** набора **Embed Font Outlines For**.

Задав параметры абзаца в диалоговом окне **Character Options**, нажмите кнопку **Done**, чтобы закрыть его. После закрытия этого окна заданные вами параметры будут сразу же применены.

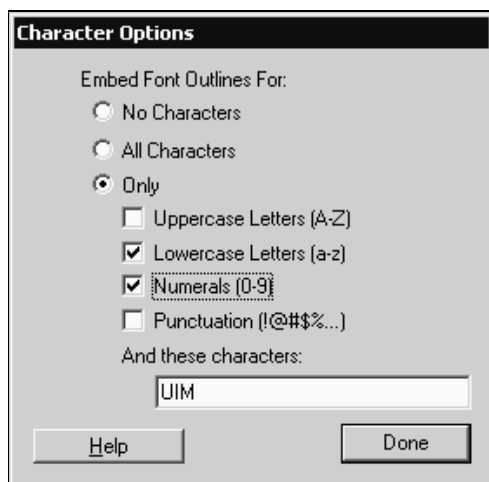


Рис. 7.14. Диалоговое окно **Character Options**

Динамические текстовые блоки

Часто возникает необходимость не показывать какой-то статический текст, а выводить результаты работы какой-либо программы. Например, встроенная в изображение Flash программа может извлекать новости с сайта "Компьюленты", с "Апорта" — прогноз погоды, а с "Рэмблера" — курсы валют, добавлять к ним текущее время и все это в виде текста выводить на экран. Для этого используется так называемый *динамический текстовый блок*. (Обычные же текстовые блоки называют иногда *статическими*.) Пожалуй, нужда в таком динамическом текстовом блоке возникает чаще, чем в обычном поле ввода.

Для вывода текста в динамический блок используются уже знакомые вам переменные. Программа помещает результат своей работы в переменную, а динамический текстовый блок потом забирает их оттуда и выводит на экран. Собственно вывод текста при этом выполняется автоматически самим Flash.

Как и в случае поля ввода, создайте обычный текстовый блок и выделите его с помощью "стрелки". После этого обратитесь к редактору свойств, найдите в его верхнем левом углу раскрывающийся список и выберите в нем пункт **Dynamic Text**. Редактор свойств примет вид, показанный на рис. 7.15.

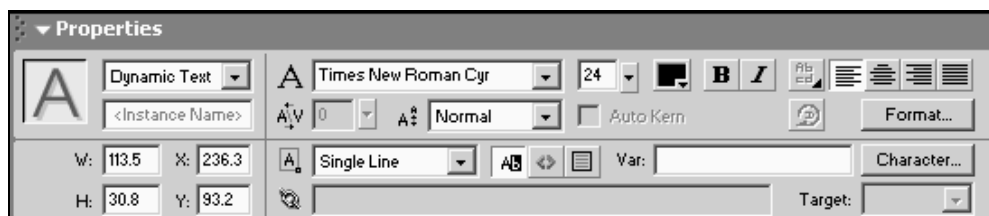


Рис. 7.15. Вид редактора свойств при выделенном динамическом текстовом блоке

Как видите, здесь доступны практически те же элементы управления, что и в случае поля ввода. Поэтому мы не будем описывать их полностью. Ограничимся только перечислением немногих отличий.

А отличие фактически единственное и, вдобавок, уже вам знакомое. Кнопка **Selectable** включает или отключает возможность выделения и копирования текста, помещенного в динамический текстовый блок. По умолчанию флажок **Selectable** включен, т. е. пользователь может выделять этот текст. (Как вы помните, для полей ввода эта кнопка была включена и недоступна для отключения.)

Вы можете сделать динамический текстовый блок *прокручиваемым*. Обычный динамический блок должен иметь такие размеры, чтобы в нем помещалось все его содержимое. Для прокручиваемого же блока это не обязательно: подобно областям редактирования Windows он может прокручивать содержащийся в нем текст. Правда, полосы прокрутки он содержать не будет, но это не слишком большая проблема, вы можете использовать компонент полосы прокрутки. (О компонентах см. главу 23.)

Чтобы сделать динамический текстовый блок прокручиваемым, сначала выделите его. После этого выберите пункт-выключатель **Scrollable** меню **Text** или одноименный пункт контекстного меню текстового блока. Если вы в данное время редактируете содержимое этого текстового блока, то можете дважды щелкнуть по его маркеру изменения размера, удерживая нажатой клавишу <Shift>. Готовый прокручиваемый текстовый блок показан на рис. 7.16, обратите внимание на форму его маркера изменения размеров.

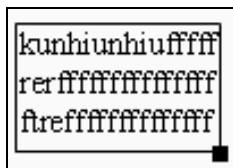


Рис. 7.16. Прокручиваемый динамический текстовый блок

Работа с символами текста как с графикой

Для желающих поглумиться над текстом Flash предоставляет возможность работы с символами текста как с графическими фрагментами. Это означает, что вы можете менять форму отдельных символов текста, как если бы это были обычные графические фрагменты, нарисованные с помощью обычных инструментов рисования.

Прежде чем править символы как графику, вам нужно превратить текст в набор графических фрагментов. Для этого выберите текстовый блок, пользуясь инструментом "стрелка выделения", и выберите пункт **Break Apart** в меню **Modify**, в контекстном меню текстового блока или нажмите комбинацию клавиш <Ctrl>+. Каждый символ созданного вами текста преобразуется в набор графических фрагментов, который впоследствии будет выделен.

Вы можете применять для бывшего текста любые знакомые вам приемы для изменения формы и цвета графики. В частности, можно расположить символы такого текста по окружности или кривой — похоже, это единственный доступный во Flash прием сделать фигурный заголовок. Также вы можете создавать новые, причудливые шрифты или просто изменять форму отдельных символов. Посмотрите на рис. 7.17, там изображен результат преобразования формы обычной латинской буквы "U" с помощью инструмента "стрелка выделения".



Рис. 7.17. Изменение формы символа "U" (слева — изначальный символ, справа — после изменения)

Есть еще одна область применения этой возможности. Помните, мы говорили, что при экспорте изображения в формат Shockwave/Flash все использованные в нем шрифты помещаются в результирующий файл? Мы еще упомянули о двух компромиссных путях решения этой проблемы. Так вот, есть возможность решить ее третьим, более удачным путем. А именно, преобразовать текст, набранный каким-либо сверхсложным и сверхэкзотическим шрифтом, в графику. Ведь чтобы отобразить набор кривых, в который

превратится наш текст, проигрывателю Flash не нужно вообще никаких шрифтов. А значит, SWF-файл вашего изображения станет меньше.

Единственный недостаток: вы не сможете преобразовать набор графических фрагментов обратно в текст. А значит, что подобный текст не подлежит редактированию. Поэтому перед преобразованием текста в графику убедитесь, что он не содержит ошибок.

Внимание!

Вы можете преобразовывать в графику только текст, набранный шрифтами формата TrueType или Adobe Type 1.

Подстановка шрифтов

Мы много говорили о том, как Flash сохраняет информацию об использованных в изображении шрифтах в файле Shockwave/Flash. Мы узнали, что при отсутствии того или иного шрифта в большинстве случаев изображение будет отображено правильно, т. к. описания шрифтов включаются в сам результирующий файл. Если, конечно, разработчик специально не отключил включение описаний этих шрифтов.

Однако иногда бывает и другая ситуация. Вы отдаете документ Flash (FLA-файл) своему коллеге, а тот пытается открыть его на своем компьютере. И тут выясняется, что в его системе нет шрифта, которым был набран некий текст. Что случится в этом случае?

Как только Flash понадобится для отображения текста этот самый отсутствующий шрифт, он выдаст предупреждение об этом. Это предупреждение показано на рис. 7.18. Давайте рассмотрим его во всех подробностях.

Итак, текст этого предупреждения гласит, что Flash для вывода текста нужен шрифт, который в данный момент не установлен в системе. Чтобы вывести этот текст, Flash предлагает вам выполнить так называемую *подстановку шрифта*, т. е. заменить отсутствующий шрифт другим, установленным в системе. Конечно, при этом текст исказится, но зато вы сможете просмотреть, отредактировать документ и даже экспортировать его в формат Shockwave/Flash.

Интересная особенность Flash: после подстановки шрифтов он не сохраняет об этом сведений. Давайте поясним это на такой ситуации. Предположим, что вы открыли документ Flash, присланный вашим коллегой, и у вас в системе не оказалось необходимого шрифта. Вы заменили этот шрифт (назовем его *изначальным*) на другой, установленный в вашей системе, и открыли-таки документ. После этого вы изменили что-то в документе, возможно, отредактировали текст, но не меняли *изначальный* шрифт на другой. Затем вы сохранили этот документ. Позднее вы все-таки нашли нужный вам шрифт, установили его и снова открыли этот документ. И после этого он будет набран именно тем, *изначальным* шрифтом. Удобно, не правда ли?

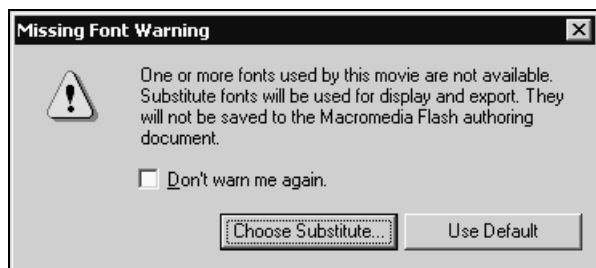


Рис. 7.18. Предупреждение об отсутствии нужного шрифта

Да, все это хорошо. Но как же все-таки подставить шрифт вместо отсутствующего?

Очень просто. Для этого Flash предлагает две альтернативы.

Альтернатива первая и самая простая: отдать все на откуп Flash. Как вы помните из главы 3, в настройках Flash можно задать шрифт, который будет автоматически подставляться вместо отсутствующих. Конкретно, он задается с помощью всплывающего списка **Font Mapping Default**, расположенного на вкладке **General** диалогового окна **Preferences**. Чтобы выбрать эту альтернативу, нажмите кнопку **Use Default** окна предупреждения, показанного на рис. 7.18.

Альтернатива вторая и более "продвинутая": взять бразды правления шрифтами в свои руки. Для этого нажмите кнопку **Choose Substitute**. На экране появится диалоговое окно **Font Mapping**, показанное на рис. 7.19.

Большую часть этого окна занимает список отсутствующих шрифтов. Он оформлен в виде таблицы из двух колонок: **Missing Fonts** (отсутствующие шрифты) и **Mapped To** (замененные шрифты). Изначально в колонке **Mapped To** для всех отсутствующих шрифтов присутствует только шрифт, заданный в настройках Flash; он обозначается строкой **System Default Font**. Вы можете выбирать строки этого списка и задавать для соответствующих отсутствующих шрифтов замененные, выбирая нужные шрифты в раскрывающемся списке **Substitute Font**. Если вы для какого-то отсутствующего шрифта хотите вернуть шрифт, заданный в настройках Flash, нажмите кнопку **System Default**. Вы также можете выбирать в списке сразу несколько строк, удерживая клавишу <Shift>.

Закончив подстановку шрифтов, нажмите кнопку **OK**, чтобы сохранить заданные вами параметры. Если вы хотите отказаться от них, нажмите кнопку **Cancel**, тогда вместо всех отсутствующих шрифтов будет подставлен шрифт, заданный в настройках Flash в качестве шрифта по умолчанию. Впоследствии, если вы выделите текстовый блок, содержащий текст, набранный отсутствующим шрифтом, то увидите, что в раскрывающемся списке шрифтов этот (отсутствующий) шрифт стоит в скобках (рис. 7.20).

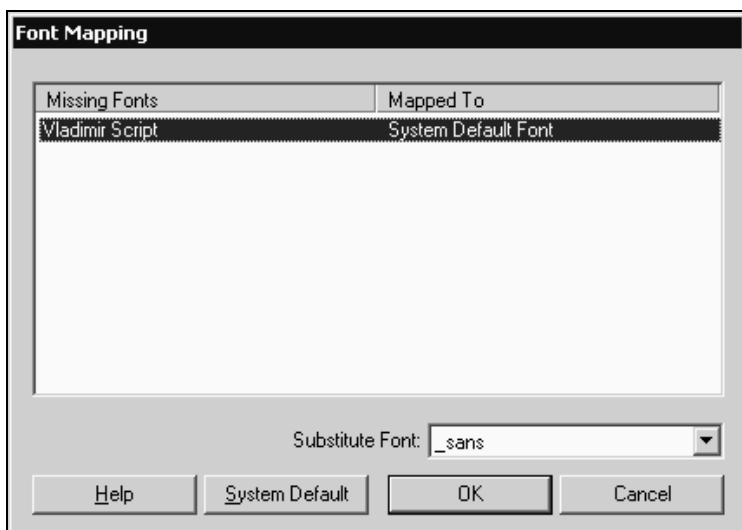


Рис. 7.19. Диалоговое окно **Font Mapping**

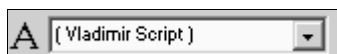


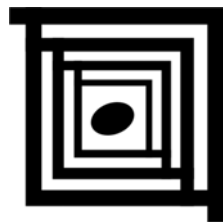
Рис. 7.20. Отсутствующий и замененный шрифт показывается в скобках

Если вы взглянете на рис. 7.18, то увидите, что окно предупреждения содержит также флажок **Don't warn me again**. Если вы его включите перед тем, как сделать свой выбор, Flash будет всегда применять его при отсутствии каких-либо шрифтов, не спрашивая у вас.

Вы можете вызвать диалоговое окно **Font Mapping**, выбрав пункт **Font Mapping** в меню **Edit**. Это позволит вам управлять подстановкой шрифтов уже после того, как документ будет открыт.

Flash сохраняет для дальнейшего использования все ваши установки, заданные в окне **Font Mapping**. Впоследствии вы можете изменить их или удалить, закрыв все окна документов и выбрав пункт **Font Mapping** в меню **Edit**. В этом случае в диалоговом окне **Font Mapping** появится кнопка **Delete**, нажав которую, вы сможете удалить выбранную позицию списка подставленных шрифтов.

Глава 8



Импорт графики

Всем хорош Flash. И рисовать на нем можно, и создавать анимацию, и даже писать довольно сложные программы — ну просто мастер на все руки. Неудивительно, что так популярен он на просторах Сети.

Однако Flash может делать не все. Некоторые задачи лучше всего выполнить в других программах, а потом импортировать во Flash результаты их работы. И профессиональные Flash-художники так и делают. В самом деле, каждую задачу лучше всего решать с помощью наиболее подходящего инструмента. Ведь никто не забивает гвозди микроскопом, хотя теоретически это вполне возможно.

Для забивания гвоздей служит молоток (если кто этого не знает). Для редактирования растровой графики служат программы растровых редакторов, например, Adobe PhotoShop и Macromedia Freehand. Для очень сложной векторной графики есть более мощные, чем Flash, векторные редакторы: Corel DRAW! и Macromedia Freehand. Есть также специальные графические программы, например, AutoDesk AutoCAD, служащая для создания чертежей и схем, и Microsoft Visio, где так удобно и приятно рисовать бланки.

Каждый из перечисленных выше программных пакетов — в своей области узкий специалист. Полнота его односторонняя, как у недоброй памяти флюса, но свою задачу он знает великолепно.

А что же Flash? Задача Flash — принести в Интернет компактную векторную графику и интерактивную анимацию, которая не слишком бы "нагружала" маломощные клиентские компьютеры. Flash не работает со сверхсложной графикой, не создает сверхмощные программы и не редактирует растровые изображения. Но свою прямую задачу он выполняет на "отлично", о чем говорят цифры, показывающие его распространение в Сети. У создателей Flash не было намерения объять необъятное. И слава Богу...

Поддержка графических форматов

Сначала поговорим о том, какие графические форматы поддерживает Flash. Это важно узнать в самую первую очередь, чтобы потом не пытаться импортировать файл заведомо не поддерживаемого формата и не потерять случайно половину содержащихся в файле графических данных.

Если вы хотите импортировать во Flash файл не поддерживаемого им графического формата, вам придется найти программу, поддерживающую этот формат. С ее помощью вы сможете преобразовать этот файл к формату, поддерживаемому Flash. Однако будьте готовы к тому, что при таком преобразовании возможны потери некоторой информации.

Список поддерживаемых форматов

Приведем список графических форматов, которые можно импортировать во Flash. Этот список оформим в виде табл. 8.1.

Таблица 8.1. Список графических форматов, которые можно импортировать во Flash

Название формата	Расширение файлов
Adobe Illustrator	ai, eps
AutoDesk AutoCAD	dxf
FutureSplash	spl
GIF (обычный и анимированный)	gif
JPEG	jpg, jpe, jpeg
Macromedia Freehand	fh7, ft7, fh8, ft8, fh9, ft9, fh10
Macromedia Shockwave/Flash	swf
PNG	png
Метафайлы Windows	wmf
Растровые файлы Windows	bmp
Расширенные метафайлы Windows	emf

Если на компьютере пользователя Flash установлен пакет Apple QuickTime, становятся доступными еще несколько графических форматов. Все они перечислены в табл. 8.2.

Таблица 8.2. Список дополнительных графических форматов, доступных Flash

Название формата	Расширение файлов
Apple MacPaint	pntg
Adobe PhotoShop	psd
Растровые файлы Macintosh	pct, pic
Apple QuickTime (статичные изображения и фильмы)	qtif, mov
Растровые файлы Silicon Graphics	sai
TGA	tga
TIFF	tif, tiff

При импорте файла растрового изображения (формата BMP, GIF или любого другого), чье имя кончается числом, Flash ищет аналогичные ему файлы. Если он находит последовательность таких пронумерованных файлов, например, Рис1.bmp, Рис2.bmp, Рис3.bmp и т. д., то предлагает импортировать всю эту последовательность как видеоклип. На экране появляется соответствующее предупреждение, нажмите кнопку **Yes (Да)** или **No (Нет)**. Если вы нажмете **Yes**, Flash импортирует последовательность файлов и превратит их в клип; если же вы нажмете **No**, будет импортирован только выбранный вами файл.

Импорт графики

Теперь рассмотрим, как же выполняется сам импорт графики.

Существует два способа импортировать во Flash графику, созданную в другом графическом пакете. Мы рассмотрим их оба.

Первый способ вам уже знаком. Мы пользовались им, когда создавали графическую заливку в *главе 6*. Давайте вспомним, что мы тогда делали.

Для того чтобы импортировать во Flash чужую графику, выберите пункт **Import** в меню **File** или нажмите комбинацию клавиш <Ctrl>+<R>. На экране появится стандартное диалоговое окно открытия файла **Windows**. Найдите нужный файл и нажмите кнопку открытия файла этого диалогового окна. Как видите, это очень просто.

Второй способ еще проще. Запустите программный пакет, откройте в нем нужный файл или создайте его заново, скопируйте нужный графический фрагмент или все изображение в буфер обмена **Windows** и вставьте его во Flash.

Импортированное вами графическое изображение будет тотчас помещено на рабочий лист и выделено. Вы можете перемещать его, пользуясь "стрелкой выделения", помещать его в буфер обмена и выполнять другие операции, доступные для выделенных графических фрагментов.

Кроме того, импортированное изображение помещается в качестве образца в библиотеку. Так что вы можете в дальнейшем создавать сколько угодно экземпляров этого изображения на листе, а размер файла Flash при этом не увеличится. (О библиотеках, образцах и экземплярах см. главу 10.)

Кстати, есть возможность импортировать графическое изображение прямо в библиотеку, не помещая его на рабочий лист. Для этого выберите пункт **Import to Library** в меню **File**. На экране появится стандартное диалоговое окно открытия файла Windows. Найдите нужный файл и нажмите кнопку открытия файла этого диалогового окна.

Особенности поддержки графических форматов

Поддержка некоторых графических форматов имеет особенности, которые мы здесь опишем.

Macromedia Fireworks

Пакет создания Web-графики Macromedia Fireworks — в чем-то "коллега" Flash. Он тоже создает интернет-графику, но иного рода: во-первых, это графика растровая, а во-вторых, используемая, в основном, как элементы оформления Web-страниц. Fireworks экспортирует графику в форматах GIF, JPEG и PNG; последний формат также служит для хранения рабочих документов (наподобие FLA-файлов Flash).

Создаваемые в Fireworks графические изображения (имеются в виду рабочие документы) могут содержать как растровую, так и векторную графику. Это становится возможным из-за того, что формат PNG может хранить дополнительные данные (теги) наряду с растровой графикой. При экспорте готового изображения Fireworks автоматически растрирует всю векторную графику.

Вы можете импортировать созданные в Fireworks PNG-файлы во Flash без всяких проблем. При этом вы можете импортировать ее как полностью растровое изображение, так и в изначальном виде. В первом случае Flash растрирует всю содержащуюся в файле векторную графику, и вы потеряете возможность ее редактировать. Во втором случае растровая графика останется растровой, а векторная — векторной, и вы сможете редактировать векторную графику во Flash.

Импорт графики, созданной в Fireworks, имеет некоторые особенности. После выбора нужного файла PNG в стандартном диалоговом окне открытия файла на экране появится диалоговое окно выбора параметров импорта **Fireworks PNG Import Settings**. Оно показано на рис. 8.1.

Если вы хотите импортировать все изображение Fireworks как одно огромное растровое изображение, включите флажок **Import as a single flattened bitmap**. Если же вы хотите сохранить векторную графику, отключите его. Если вы его отключите, станут доступны три группы переключателей, с помощью которых вы сможете задать дополнительные параметры импорта.

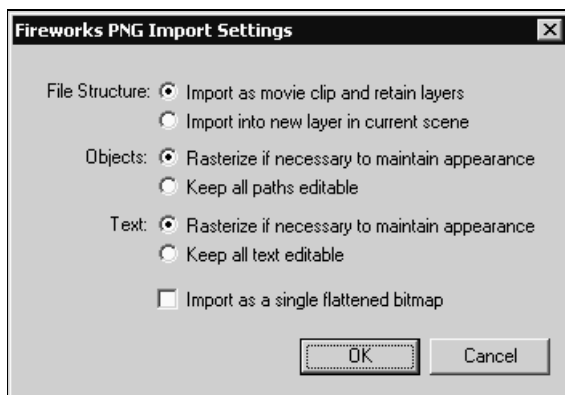


Рис. 8.1. Диалоговое окно **Fireworks PNG Import Settings**

С помощью переключателей группы **File Structure** вы можете задать, в каком виде будет импортирована графика Fireworks. Переключатель **Import as movie clip and retain layers**, включенный по умолчанию, позволяет импортировать изображение Fireworks как отдельный клип с сохранением всех его слоев. Переключатель **Import into new layer in current scene** задает импорт изображения Fireworks в один вновь созданный слой. (О слоях см. главу 15.)

Переключатели группы **Objects** задают, в каком виде будут импортированы графические примитивы Fireworks, не поддерживаемые Flash. Если включен переключатель **Rasterize if necessary to maintain appearance** (а он включен по умолчанию), то такие примитивы будут превращены в растровые изображения. Если же включен переключатель **Keep all paths editable**, никакой растеризации выполняться не будет, в результате этого некоторые графические фрагменты изображения Fireworks могут быть потеряны.

Переключатели группы **Text** задают те же самые параметры, что и переключатели группы **Objects**, но только для текста. Так, если включен переключатель **Rasterize if necessary to maintain appearance** (а он включен по умолчанию), текст, содержащий преобразования, не поддерживаемые Flash, будет растриван. Если же включен переключатель **Keep all text editable**, никакой растеризации выполняться не будет, в результате этого некоторые текстовые блоки исходного изображения могут быть потеряны.

Задав нужные параметры, нажмите кнопку **ОК**. Если вы передумали импортировать файл Fireworks, нажмите кнопку **Cancel**.

Macromedia Freehand

Векторный графический редактор Freehand версий 7—10 — лучшее средство для создания сложной векторной графики. Созданные в нем файлы можно импортировать во Flash без особого труда и практически без потерь качества. Flash автоматически "подгонит" импортированную графику "под себя", чтобы преодолеть свои ограничения.

Если графическое изображение, созданное во Freehand, имеет градиентную заливку с более чем восемью цветами, Flash создаст несколько перекрывающихся заливок. Учтите, что это может увеличить размер результирующего файла Shockwave/Flash, поэтому используйте градиенты с восемью или меньшим количеством цветов.

Как вы уже знаете, Flash сливает или фрагментирует накладывающиеся друг на друга графические примитивы. Freehand же этого не делает. Поэтому импортированная во Flash графика Freehand может быть из-за этого искажена. Чтобы предотвратить слияние и фрагментацию примитивов, при рисовании во Freehand разместите их в отдельных слоях.

Если изображение Freehand было создано в цветовой схеме *CMYK* (Cyan, Magenta, Yellow, black — голубой, пурпурный, желтый, черный), оно автоматически преобразуется Flash в формат RGB. Также Flash скругляет углы у линий в графике Freehand и преобразует черно-белые изображения в цветной формат, т. к. не поддерживает черно-белую графику. А если графика Freehand содержит внедренные изображения в формате EPS, то перед внедрением вам нужно будет включить флажок **Convert editable EPS when imported** на вкладке **Import** диалогового окна настройки Freehand **Preferences**. Иначе Flash не сможет распознать такие изображения.

Импорт графики, созданной в Freehand, также имеет особенности. После выбора нужного файла в стандартном диалоговом окне открытия файла на экране появится диалоговое окно выбора параметров импорта **Freehand Import**. Оно показано на рис. 8.2.

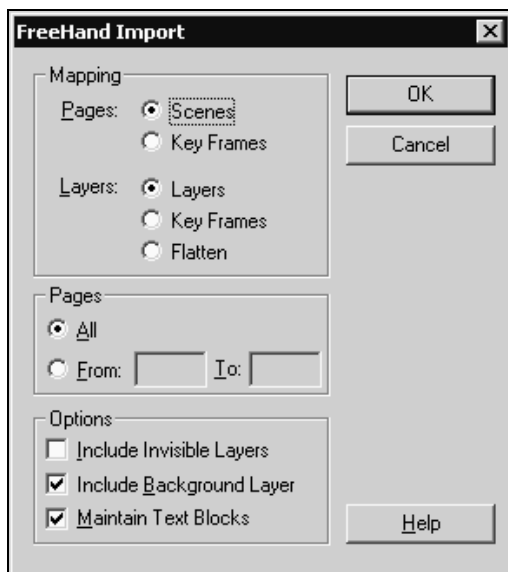


Рис. 8.2. Диалоговое окно **Freehand Import**

Это диалоговое окно содержит несколько групп элементов управления. Рассмотрим их по порядку.

Группа элементов управления **Mapping** позволит вам задать действия, предпринимаемые Flash в отношении страниц и слоев изображения FreeHand. Она в свою очередь содержит две группы переключателей, рассматриваемые ниже.

Группа переключателей **Pages** управляет импортом страниц. Переключатель **Scenes** заставляет Flash преобразовать каждую страницу в сцену (Flash не поддерживает страницы), а переключатель **Key Frames** — в ключевой кадр. О сценах и ключевых кадрах см. главу 13.

Группа переключателей **Layers** управляет импортом слоев. Переключатель **Layers** (включен изначально) заставляет Flash преобразовать каждый слой FreeHand в свой слой, переключатель **Key Frames** — в ключевой кадр, а **Flatten** "сваливает" содержимое всех слоев FreeHand в один слой Flash. Flash сам поддерживает слои, но все же в этом вопросе требует дополнительной ясности. О слоях см. главу 15.

Группа элементов управления **Pages** задает, какие страницы документа FreeHand будут импортированы. Если выбрать переключатель **All** (а он включен по умолчанию), будут импортированы все страницы. Если выбрать переключатель **From**, будут импортированы только страницы с номерами, указанными в полях ввода **From** (начальный номер) и **To** (конечный).

Флажки, расположенные в группе элементов управления **Options**, задают дополнительные параметры импорта графики FreeHand. Флажок **Include Visible Layers** включает импорт только видимых слоев изображения FreeHand; если он отключен, импортируются все слои. Флажок **Include Background Layer** включает или отключает импорт фонового слоя FreeHand. А флажок **Maintain Text Blocks** включает или отключает помещение импортированного текста в текстовые блоки.

Задав нужные параметры, нажмите кнопку **ОК**. Если вы передумали импортировать файл FreeHand, нажмите кнопку **Cancel**.

Adobe Illustrator

Flash позволяет импортировать файлы, созданные в версиях 8.0 или более ранних.

Перед импортом графики Illustrator во Flash разгруппируйте все сгруппированные графические фрагменты. Если этого не сделать, Flash не сможет правильно их обработать.

После выбора файла Illustrator в стандартном диалоговом окне открытия файла на экране появится диалоговое окно выбора параметров импорта **Illustrator Import**. Оно показано на рис. 8.3.

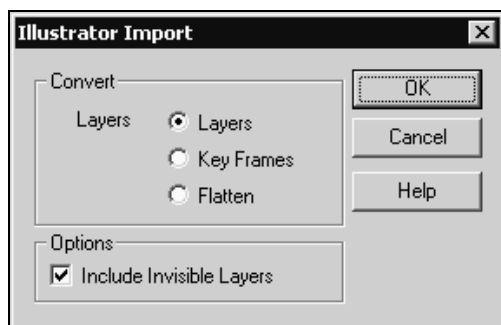


Рис. 8.3. Диалоговое окно **Illustrator Import**

Группа переключателей **Convert** задает параметры преобразования слоев Illustrator в слои Flash. Переключатель **Layers** (включен изначально) указывает Flash преобразовать слои Illustrator в свои слои, переключатель **Key Frames** — в ключевые кадры, а переключатель **Flatten** помещает содержимое всех слоев Illustrator в один слой Flash.

Флажок **Include Invisible Layers** включает или отключает импорт невидимых слоев Illustrator. Изначально он включен.

Задав нужные параметры, нажмите кнопку **OK**. Если вы передумали импортировать файл Illustrator, нажмите кнопку **Cancel**.

AutoDesk AutoCAD

Flash позволяет импортировать чертежи и схемы AutoCAD версии 10 или более поздней. Учтите, что Flash поддерживает только файлы, сохраненные в текстовом формате. Flash не поддерживает трехмерные чертежи и схемы — только двумерные.

Поскольку AutoCAD не поддерживает стандартные системные шрифты, при импорте возможно искажение текста. Также AutoCAD не поддерживает заливки, поэтому вся импортированная графика будет состоять из одних контуров.

Работа с импортированной графикой

Вы уже знаете, что над импортированной графикой можно выполнять все манипуляции, доступные и для других графических фрагментов. Однако Flash предлагает вам еще несколько возможностей, которые мы сейчас рассмотрим. Они вам могут пригодиться в дальнейшем.

Векторизация растровой графики

Что такое векторизация, вы уже знаете. Если же не знаете, обратитесь к *главе 4*. А здесь мы только кратко напомним, чем она может нам помочь и чем навредить.

Как вы знаете, импортированное растровое изображение имеет очень большой размер. А значит, размер результирующего файла Shockwave/Flash может сильно возрасти, если в документ было импортировано растровое изображение. Уменьшить его может или отказ от этого изображения, или векторизация.

Если вы выбрали векторизацию, то должны решить, какое качество хотите получить в результате. Если в настройках векторизации вы зададите слишком большое качество результирующего векторного изображения, то размер документа Flash может даже вырасти. Если задано слишком низкое качество угадайте, что получится?

Векторизация, кроме этого, может быть полезна, если вы хотите отредактировать импортированное растровое изображение в среде Flash, не прибегая к другим программам. При этом растровое изображение будет разбито на группу графических примитивов, с которыми вы можете сделать все, что угодно.

Для векторизации растрового изображения, прежде всего, выберите его на рабочем листе. После этого выберите пункт **Trace Bitmap** в меню **Modify**. На экране появится диалоговое окно **Trace Bitmap**, показанное на рис. 8.4.

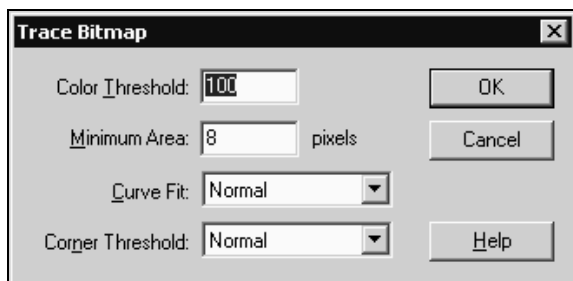


Рис. 8.4. Диалоговое окно **Trace Bitmap**

В поле ввода **Color Threshold** задается числовое значение разницы между цветами. Если две соседние точки имеют цвета, различающиеся на значение, меньшее, чем введенная разница, они считаются одноцветными. Таким образом, чем больше значение разницы цветов, тем меньше будет цветов в результирующем векторном изображении. Допускается вводить значения от 1 до 500, значение по умолчанию — 100.

В поле ввода **Minimum Area** указывается количество окружающих пикселей, которые будут взяты для определения цвета пиксела. Иными словами, это разрешающая способность системы векторизации Flash, и чем это значение больше, тем меньше нюансов будет иметь результирующее изображение. Допускается вводить значения от 1 до 1000 пикселей, значение по умолчанию — 8 пикселей.

Раскрывающийся список **Curve Fit** служит для задания точности передачи контуров изображения. Доступны шесть пунктов, задающих точность от

максимальной до минимальной: **Pixel** (точность до пиксела), **Very Tight**, **Tight**, **Normal** (значение по умолчанию), **Smooth** и **Very Smooth** (очень приближенные, "сглаженные" контуры).

Задав нужные параметры, нажмите кнопку **ОК**. Если вы передумали векторизовать растровое изображение, нажмите кнопку **Cancel**. Результат векторизации показан на рис. 8.5. Если вы внимательно посмотрите на него, то заметите потерю качества.



Рис. 8.5. Результат векторизации растрового изображения (слева — исходное изображение, справа — векторизованное)

Разбиение растровой графики. "Волшебная палочка"

Flash позволяет также разбить импортированное растровое изображение на части, исходя из цвета его пикселей. (Одновременно импортированное изображение преобразуется из экземпляра в "обычный" графический фрагмент.) Это позволит вам перекрасить любые части этого изображения в любой цвет и, вообще, изменить его средствами Flash.

Для того чтобы разбить растровое изображение на части, прежде всего, выделите его на рабочем листе. После этого выберите пункт **Break Apart** в меню **Modify** или контекстном меню растрового изображения или просто нажмите комбинацию клавиш <Ctrl>+.

На первый взгляд, растровое изображение не изменится. Но это только на первый взгляд. Давайте попробуем закрасить какую-либо его часть другим цветом.

Чтобы выбрать часть разбитого на части растрового изображения, выберите уже знакомый вам инструмент "лассо". И нажмите кнопку-выключатель, показанную на рис. 8.6. Эта кнопка включает модификатор "волшебная палочка", с помощью которого и осуществляется выбор части разбитого растрового изображения. После включения этой кнопки курсор мыши примет вид небольшой волшебной палочки.



Рис. 8.6. Кнопка модификатора "волшебная палочка"

Для выделения части разбитого растрового изображения просто щелкните по ней. Она будет выделена (рис. 8.7). Чтобы добавить к выделению другие части изображения, продолжайте щелкать по ним. Если вы щелкнете по уже выделенной части, все выделение будет снято.

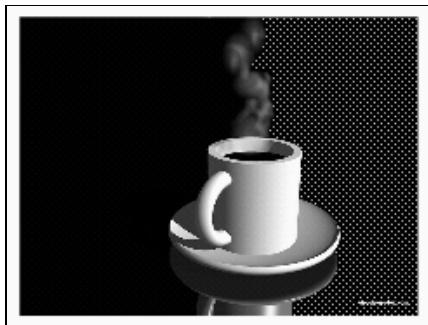


Рис. 8.7. Выделенная часть разбитого растрового изображения

Теперь вы можете воспользоваться инструментом "ведро с краской", чтобы закрасить выделенные части изображения. Вы также можете выбрать инструмент "пипетка", чтобы использовать разбитое на части изображение как графическую заливку.

Нажав кнопку, показанную на рис. 8.8, вы сможете настроить параметры модификатора "волшебная палочка". Эти настройки выполняются в диалоговом окне **Magic Wand Settings**, показанном на рис. 8.9.



Рис. 8.8. Кнопка настройки параметров модификатора "волшебная палочка"

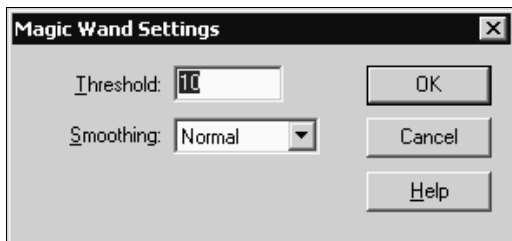


Рис. 8.9. Диалоговое окно **Magic Wand Settings**

В поле ввода **Threshold** задается числовое значение разницы между цветами. Если две соседние точки имеют цвета, различающиеся на значение, меньшее, чем введенная разница, они считаются одноцветными. Таким образом, чем больше значение разницы цветов, тем больший диапазон цветов будет включать в себя выделяемая с помощью "волшебной палочки" часть изображения. Допускается вводить значения от 0 до 200, значение по умолчанию — 10.

Раскрывающийся список **Smoothing** служит для задания степени сглаживания контуров, охватывающих части разбитого растрового изображения. Доступны четыре пункта, задающие точность от максимальной до минимальной: **Pixel** (точность до пиксела), **Rough**, **Normal** (значение по умолчанию) и **Smooth** ("мягкие" контуры).

Задав нужные параметры модификатора "волшебная палочка", нажмите кнопку **ОК**, чтобы их сохранить. Нажатие кнопки **Cancel** вызовет отмену введенных параметров и возврат к старым значениям.

Замена одного изображения на другое

Flash предоставляет возможность быстро заменить одно импортированное изображение на другое. Для этого сначала импортируйте изображение в документ и, если нужно, удалите его с рабочего листа. В этом случае очень поможет пункт **Import to Library** меню **File**, выполняющий импорт изображения прямо в библиотеку. И проверьте, не разбили ли вы это изображение на части, заменить можно только одно неразбитое изображение на другое неразбитое.

Выделите на рабочем листе нужное изображение. Выберите в меню **Modify** пункт **Swap Bitmap**. Вы также можете выбрать одноименный пункт в контекстном меню импортированного изображения или просто нажать кнопку **Swap** в редакторе свойств (рис. 8.10). На экране появится диалоговое окно **Swap Bitmap** (рис. 8.11).

В списке, занимающем большую часть этого окна, выберите нужное изображение. Выбранное изображение показывается в области предварительного просмотра, расположенной левее списка. После этого нажмите кнопку **ОК**. Если же вы передумали заменять изображение, нажмите кнопку **Cancel**.

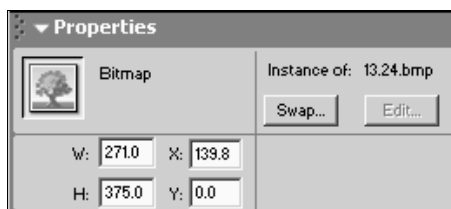


Рис. 8.10. Вид редактора свойств при выбранном импортированном изображении

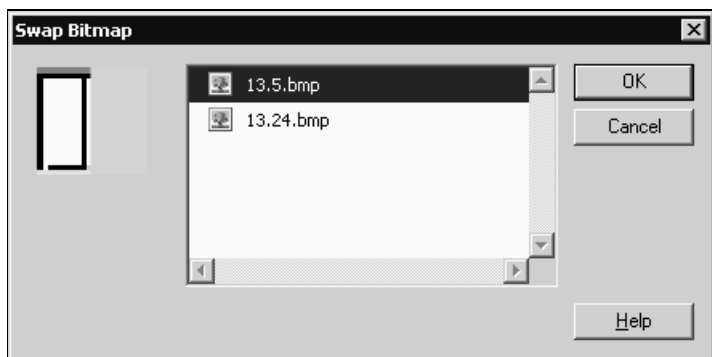


Рис. 8.11. Диалоговое окно **Swap Bitmap**

Задание параметров растрового изображения

При экспорте готового изображения, содержащего импортированную растровую графику, Flash выполняет ее сжатие. Вы можете управлять параметрами сжатия графики, а именно, степенью и используемым алгоритмом. Рассмотрим, как это делается.

Прежде всего, откройте окно библиотеки (о библиотеках и образцах см. главу 10). Для этого выберите пункт **Library** в меню **Window** или нажмите клавишу <F11>. Появившееся на экране окно будет содержать список образцов, найдите в нем образец нужного растрового изображения, выделите его и выберите в контекстном или дополнительном меню пункт **Properties**. На экране появится диалоговое окно **Bitmap Properties** (рис. 8.12).

В поле ввода, находящемся в верхней части этого окна, задается имя образца. Ниже этого поля ввода расположены следующие данные (в порядке сверху вниз):

- ☐ путь и имя изначального файла;
- ☐ дата и время создания образца;
- ☐ параметры изображения, а именно, размеры в пикселах и цветовой режим.

В левой части окна находится небольшая панель предварительного просмотра, в которой отображается само это растровое изображение.

Флажок **Allow smoothing** включает или отключает сглаживание границ изображения. Рекомендуется всегда оставлять его включенным.

В раскрывающемся списке **Compression** задается алгоритм сжатия растрового изображения. В нем доступны два пункта, которые мы перечислим ниже.

Пункт **Photo (JPEG)** задает для изображения алгоритм сжатия с потерями, такой же, какой применяется для сжатия графики формата JPEG. Используйте этот алгоритм для сложных полутоновых изображений с большим количеством цветов или оттенков серого, например, фотографий или рисунков с градиентными заливками.

Рис. 8.12. Диалоговое окно **Bitmap Properties**

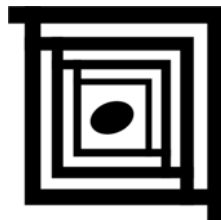
Если выбран пункт **Photo (JPEG)** списка **Compression**, ниже его появится флажок **Use document default quality**. Включите его, чтобы задать для этого изображения качество по умолчанию. Если вы его отключите, то еще ниже появится поле ввода **Quality**, где вы сможете ввести числовое значение от 1 до 100, задающее качество изображения. Максимальное качество при этом составляет 100 единиц, а качество по умолчанию — 50.

Пункт **Lossless (PNG/GIF)** задает для изображения алгоритм сжатия без потерь, который применяется для сжатия графики форматов GIF и PNG. Используйте этот алгоритм для простых штриховых изображений с небольшим количеством цветов, например, схем или карандашных рисунков.

Задав нужные параметры сжатия, щелкните кнопку **Test**. Flash тотчас обновит изображение в панели предварительного просмотра в левой части окна; оцените качество получившегося изображения. А в самом низу диалогового окна появятся данные о том, насколько новое изображение получилось меньше изначального. Возможно, вам придется несколько раз изменить параметры сжатия и нажать кнопку **Test**, чтобы получить нужное качество при приемлемом размере изображения.

Воспользовавшись кнопкой **Import**, вы можете импортировать другое изображение из другого файла. При этом новое изображение заменит то, которое вы в данный момент выбрали в списке образцов библиотеки. Когда вы нажмете эту кнопку, на экране появится обычное диалоговое окно открытия файла Windows, выберите нужный файл и нажмите кнопку открытия. А кнопка **Update** позволит вам обновить данное изображение, например, после правки его в посторонней программе.

Задание нужных параметров сжатия завершается, как и обычно, нажатием на кнопку **OK**, отказ — **Cancel**.



Глава 9

Работа с графическими фрагментами

Итак, рисовать графику мы уже научились. Теперь выясним, что мы можем делать с уже нарисованной графикой.

Вы, конечно, помните, что векторной графикой, в отличие от растровой, можно очень легко манипулировать без потери качества. Вы можете изменять ее размеры, искажать, сдвигать, вращать, менять ее цвет и т. п., а изображение останется все таким же гладким. Если же вы попытаетесь увеличить растровое изображение, оно распадется на отдельные громадные пиксели и, соответственно, ухудшится его качество. Правда, Flash, как и многие современные графические программы, используют для масштабирования и искажения растровой графики особые алгоритмы, минимизирующие потери качества. Но все равно векторной графикой манипулировать проще.

Да, но что можно делать с векторной графикой во Flash? В этой главе мы и узнаем, что и как. (На вопрос — зачем — мы думаем, вы ответите сами.)

Все описанные здесь манипуляции выполняются над выделенным графическим фрагментом. Как выделять графические фрагменты на листе, вы уже знаете, об этом рассказывалось в *главе 5*. Там же рассказывалось о простейших манипуляциях, выполняемых над выделенными фрагментами: перемещении, удалении, копировании и т. п. А в *главе 6* рассказывалось об изменении цвета и вида линий контура и заливок. Так что кое-что вы уже знаете.

Пришла пора узнать остальное.

Простейшие манипуляции

Здесь мы рассмотрим простейшие манипуляции над графическими фрагментами.

Изменение порядка наложения

Вы, вероятно, уже заметили, что графические фрагменты, расположенные на рабочем листе, могут перекрывать друг друга. При этом один фрагмент

может оказаться наверху, другой — внизу, а третий — между первыми двумя. Как видите, все графические фрагменты на листе "сложены" согласно особому порядку, называемому *порядком перекрытия*. (Опытные компьютерные художники часто говорят в этом случае о *z-координате*, по аналогии с координатами *x* и *y*.)

Отсчет в порядке перекрытия ведется, начиная с самого нижнего фрагмента, имеющего номер 0, до самого верхнего. Исходя из этого, фрагменты с большим номером в порядке перекрытия находятся выше фрагментов с меньшим номером. Как видите, здесь все просто.

Когда вы рисуете что-то на уже существующем изображении, Flash руководствуется только одним правилом: то, что нарисовано позже, лежит выше. Поэтому, если вы зачеркнете нарисованный прямоугольник, линии зачеркивания будут находиться поверх него. Есть только одно исключение из этого правила: группы и экземпляры библиотечных образцов всегда находятся выше обычной графики. Чтобы переместить обычную графику выше, вам придется или сгруппировать ее, или преобразовать в образцы и поместить на рабочий лист их экземпляры. (Об образцах и экземплярах см. главу 10.)

Часто бывает необходимо переместить какой-либо графический фрагмент выше или ниже в "стопке" фрагментов, "сложенных" на листе. Для этого Flash предлагает различные пункты подменю **Arrange**, расположенное в меню **Modify**.

Если вам нужно переместить какой-либо графический фрагмент выше или ниже в порядке перекрытия, выберите соответственно пункт **Bring to Front** или **Sent to Back** этого подменю. Вы также можете нажать комбинацию клавиш <Ctrl>+<Shift>+<↑> или <Ctrl>+<Shift>+<↓> соответственно.

Если вам нужно переместить какой-либо графический фрагмент на самый верх или в самый низ, вам следует выбрать соответственно пункт **Bring Forward** или **Sent Backward** этого подменю. Аналогичные комбинации клавиш: <Ctrl>+<↑> и <Ctrl>+<↓>.

Вы также можете выполнять эти операции сразу над несколькими выделенными графическими фрагментами. Помните только, что порядок перекрытия внутри самой выделенной группы не меняется.

Выравнивание

Что такое выравнивание текста, вы знаете. Но о *выравнивании* графических фрагментов, наверно, слышите впервые.

Однако и графику часто бывает необходимо выровнять, скажем, по верхней стороне листа или по правой стороне самого правого фрагмента в выделенной группе. Часто выполняется также центрирование выделенных фрагментов по горизонтальной или вертикальной оси. Возможно также *распреде-*

ние графических фрагментов, т. е. размещение их таким образом, чтобы их центры или границы находились на одинаковом расстоянии друг от друга по горизонтали или вертикали. И, наконец, все графические фрагменты можно сделать одинаковых размеров по горизонтали или вертикали.

Все вышеперечисленные действия вполне укладываются в понятие выравнивания графических фрагментов.

Выравнивание графики выполняется с помощью панели **Align**. Чтобы вызвать ее на экран, выберите пункт **Align** меню **Window** или нажмите комбинацию клавиш <Ctrl>+<K>. Также вы можете выбрать пункт **Align** подменю **Panels** контекстного меню выделенного графического фрагмента. Сама панель показана на рис. 9.1.

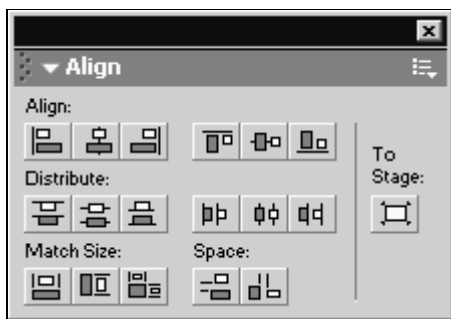


Рис. 9.1. Панель **Align**

Как видите, эта панель содержит четыре группы кнопок, выполняющих различные операции по выравниванию графики. Им соответствуют различные пункты подменю **Align**, находящегося в меню **Modify**. Рассмотрим их по очереди.

Группа кнопок **Align** выполняет собственно выравнивание. Перечислим все находящиеся в ней кнопки слева направо:

- ☐ выравнивание по левой границе самого левого выделенного фрагмента (пункт **Left**, комбинация клавиш <Ctrl>+<Alt>+<1>);
- ☐ выравнивание по центральной горизонтальной оси всей выделенной группы фрагментов (пункт **Center Vertical**, комбинация клавиш <Ctrl>+<Alt>+<2>);
- ☐ выравнивание по правой границе самого правого выделенного фрагмента (пункт **Right**, комбинация клавиш <Ctrl>+<Alt>+<3>);
- ☐ выравнивание по верхней границе самого верхнего выделенного фрагмента (пункт **Top**, комбинация клавиш <Ctrl>+<Alt>+<4>);
- ☐ выравнивание по центральной вертикальной оси всей выделенной группы фрагментов (пункт **Center Horizontal**, комбинация клавиш <Ctrl>+<Alt>+<5>);
- ☐ выравнивание по нижней границе самого нижнего выделенного фрагмента (пункт **Bottom**, комбинация клавиш <Ctrl>+<Alt>+<6>).

Группа кнопок **Distribute** выполняет распределение графических фрагментов, т. е. перемещение фрагментов так, чтобы их границы или центры находились на равном расстоянии друг от друга. Аналогично перечислим все ее кнопки слева направо:

- ☐ распределение верхних границ — перемещение фрагментов по горизонтали таким образом, чтобы их верхние границы находились на одинаковом расстоянии друг от друга;
- ☐ распределение центров по вертикали, как показано на рис. 9.2 (пункт **Distribute Heights**, комбинация клавиш <Ctrl>+<Alt>+<9>);
- ☐ распределение нижних границ (соответствующего пункта меню **Align** нет);
- ☐ распределение левых границ — перемещение фрагментов по вертикали таким образом, чтобы их левые границы находились на одинаковом расстоянии <Ctrl>+<Alt>+<7>);
- ☐ распределение друг от друга;
- ☐ распределение центров по горизонтали (пункт **Distribute Widths**, комбинация клавиш правых границ (соответствующего пункта меню **Align** нет).

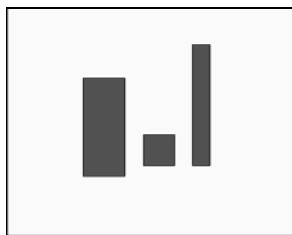


Рис. 9.2. Распределение центров по вертикали. Видно, что центры всех прямоугольников находятся на равном расстоянии друг от друга по вертикали

Группа кнопок **Match Size** задает общие размеры для выделенных фрагментов. Вот находящиеся в ней кнопки в порядке слева направо:

- ☐ все фрагменты должны быть одинаковой ширины (пункт **Make Same Widths**, комбинация клавиш <Ctrl>+<Alt>+<Shift>+<7>);
- ☐ все фрагменты должны быть одинаковой высоты (пункт **Make Same Heights**, комбинация клавиш <Ctrl>+<Alt>+<Shift>+<9>);
- ☐ все фрагменты должны быть одинаковой ширины и высоты (соответствующего пункта меню **Align** нет).

Группа кнопок **Space** задает одинаковое расстояние между выделенными фрагментами. Вот находящиеся в ней кнопки в порядке слева направо (соответствующих им пунктов меню **Align** не существует):

- ☐ вертикальное расстояние между фрагментами должно быть одинаково;
- ☐ горизонтальное расстояние между фрагментами должно быть одинаково.

Последняя кнопка-выключатель — **To Stage** — позволяет вам применить выравнивание не к группе выделенных фрагментов, а ко всему рабочему листу. Например, если вы примените выравнивание по верхней границе к выделенным фрагментам при включенной кнопке **To Stage**, они будут выровнены не по верхней границе самого верхнего из них, а по верхней границе листа (рис. 9.3 и 9.4). Эту кнопку удобно использовать в том случае, если вам нужно, например, расположить какой-либо графический фрагмент точно в центре листа. Кнопке **To Stage** соответствует пункт-выключатель **To Stage** подменю **Align** меню **Modify** и комбинация клавиш <Ctrl>+<Alt>+<8>.



Рис. 9.3. Выравнивание по верхней грани при выключенной кнопке **To Stage**



Рис. 9.4. Выравнивание по верхней грани при включенной кнопке **To Stage**

Перемещение и изменение размеров

Мы уже знаем, как перемещать выделенные фрагменты по рабочему листу с помощью мыши и клавиш-стрелок. Однако наши знания далеко не полны. Здесь мы рассмотрим еще один способ перемещения выделенной графики — заданием числовых значений в поля ввода, расположенные в редакторе свойств и на панели **Info**. Также можно менять размеры выделенного фрагмента графики. И редактор свойств, и эта панель содержат примерно один и тот же набор элементов управления.

Выделите на листе какой-либо графический фрагмент и посмотрите на редактор свойств. В его нижнем левом углу находится набор из четырех полей ввода (рис. 9.5). С их помощью задается местонахождение выделенного фрагмента и его размеры.

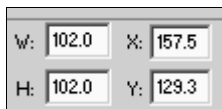


Рис. 9.5. Набор полей ввода **W**, **H**, **X** и **Y** в редакторе свойств

Назначение полей ввода, находящихся в этой группе, достаточно понятно. В полях ввода **W** и **H** задаются соответственно ширина и высота выделенного фрагмента. В полях ввода **X** и **Y** указываются координаты фрагмента,

горизонтальная и вертикальная. Сразу же после ввода каких-либо значений в эти поля нужно нажать клавишу <Enter>, чтобы Flash перенес эти значения на рабочий лист.

Чтобы вызвать на экран панель **Info**, выберите пункт **Info** в меню **Window** или нажмите комбинацию клавиш <Ctrl>+<I>. Также вы можете выбрать пункт **Info** подменю **Panels** контекстного меню выделенного фрагмента. Сама панель **Info** показана на рис. 9.6.

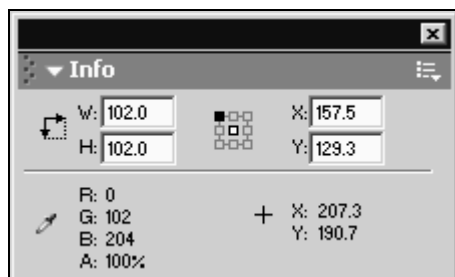


Рис. 9.6. Панель **Info**

Эта панель содержит те же четыре поля ввода. Казалось бы, зачем она нужна? Посмотрите на значок, находящийся левее полей ввода **X** и **Y**. Он показан на рис. 9.7. Это *переключатель точки отсчета*, позволяющий выбрать точку выделенного фрагмента, относительно которой будут отсчитываться координаты.



Рис. 9.7. Переключатель точки отсчета

Как видите, переключатель имеет девять точек, которые могут быть выбраны в качестве текущей точки отсчета. В реальности же некоторые из них всегда запрещены, об этом говорит серая закрашка. Доступные точки закрашены белым цветом, на рис. 9.7 есть только одна такая точка — центральная. Та же точка, которая выбрана в данный момент в качестве текущей точки отсчета, закрашена черным (на рис. 9.7 — верхняя левая). Это значит, что изначально точкой отсчета всех фрагментов является их верхний левый угол.

Чтобы изменить точку отсчета с помощью переключателя, щелкните мышью по одной из доступных точек. Она тотчас станет черной, т. е. выделенной. Правда, для этого придется хорошо прицелиться, т. к. точки на переключателе очень малы.

В нашем случае (см. рис. 9.7) в качестве точки отсчета вы можете выбрать или верхнюю левую, или центральную точку фрагмента. Какую именно из них выбрать — зависит от ваших предпочтений.

В нижней половине панели **Info** находятся информационные текстовые поля. Поля **R**, **G**, **B** и **A** показывают параметры цвета точки, находящейся в данный момент под курсором мыши, — соответственно долю красной, зеленой, синей составляющих и прозрачность. Если под курсором мыши находится свободное пространство листа, в этих полях отображаются прочерки. А текстовые поля **X** и **Y** показывают текущие координаты курсора мыши.

Зеркальное отражение

Часто бывает нужно создать зеркальное отражение какого-либо графического фрагмента без его перемещения. Flash предоставляет такую возможность.

Выберите нужный фрагмент на листе. После этого выберите пункты **Flip Horizontal** или **Flip Vertical** подменю **Transform** меню **Modify**. Первый пункт выполняет отражение фрагмента по горизонтальной оси, второй — по вертикальной.

Более сложные манипуляции

От простого — к сложному. Это вечный путь познания. Последуем им и мы.

Но сначала давайте познакомимся с одним очень интересным и мощным инструментом Flash — "трансформатором". С его помощью вы можете как угодно манипулировать с выделенным фрагментом графики, не прибегая к другим инструментам и модификаторам. Более того, Flash предусматривает модификаторы и для "трансформатора", чтобы ограничить его возможности специально для начинающих пользователей. Поэтому мы сначала рассмотрим именно модификаторы этого инструмента, а уж потом попробуем совладать с "трансформатором" без их помощи.

Чтобы выбрать "трансформатор", щелкните в инструментарии кнопку, показанную на рис. 9.8. Вы также можете нажать клавишу <Q>. Курсор мыши примет вид стрелки, похожей на "стрелку выделения", но имеющей другой значок. После этого щелкните по нужному графическому фрагменту, чтобы его выделить.



Рис. 9.8. Кнопка включения инструмента "трансформатор"

Изменение размеров

Вы уже узнали, как можно изменить размеры графического фрагмента. Для этого достаточно указать новые значения в полях ввода **W** и **H**, находящихся в

редакторе свойств (см. рис. 9.5) и на панели **Info** (см. рис. 9.6), и нажать клавишу <Enter>. Однако это можно сделать и намного удобнее, мышью. Для этого используется модификатор "изменение размера". Проверьте, выбран ли у вас инструмент "трансформатор", выделите на листе нужный графический фрагмент и щелкните кнопку-выключатель, показанную на рис. 9.9.



Рис. 9.9. Кнопка модификатора "изменение размера"

Если у вас нет на экране инструментария, и, таким образом, кнопка модификатора "изменение размеров" недоступна, вы можете воспользоваться меню. Для этого вам даже не нужно будет выбирать инструмент "трансформатор". Просто выберите пункт-выключатель **Scale** подменю **Transform** меню **Modify**. Также вы можете выбрать пункт **Scale** контекстного меню выделенного фрагмента.

Посмотрите теперь на лист, точнее, на выделенный вами фрагмент графики. Он должен выглядеть так, как показано на рис. 9.10. Как видите, он словно вписан в невидимый прямоугольник выделения, на сторонах и углах которого расположены небольшие светлые квадраты — маркеры изменения размеров.

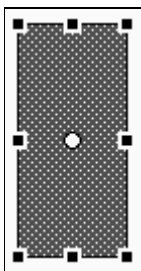


Рис. 9.10. Выделенный графический фрагмент при включенном модификаторе "изменение размера"

Маркеры изменения размеров вам уже знакомы. Мы описали их, когда говорили о правке заливок (о работе с заливками см. главу 7). Для этого случая они работают так же. На всякий случай кратко опишем их назначение.

Перемещение мышью маркеров, расположенных на сторонах воображаемого прямоугольника, изменяет один из размеров фрагмента: горизонтальный или вертикальный. Если перетаскивать мышью маркеры, расположенные на горизонтальных сторонах этого прямоугольника, меняться будет вертикальный размер, т. е. высота. Если же перетаскивать мышью маркеры на вертикальных сторонах прямоугольника, изменится горизонтальный размер — ширина. В обоих этих случаях искажаются пропорции фрагмента.

Перемещение мышью маркеров, расположенных в углах воображаемого прямоугольника, изменяет сразу оба размера фрагмента: и горизонтальный, и вертикальный. Пропорции фрагмента при этом сохраняются. Если же вы хотите менять размеры, не сохраняя пропорции, "захватите" фрагмент за угловой маркер, начните перетаскивание, нажмите клавишу <Shift> и удерживайте ее, пока не закончите перетаскивание.

Есть еще один способ изменения размеров выделенного графического фрагмента — воспользоваться панелью **Transform**. Использование этой панели чем-то напоминает использование панели **Info** — вы также указываете нужные значения в полях ввода и в конце нажимаете клавишу <Enter>, чтобы Flash обновил графику на рабочем листе. В некоторых случаях, например, если вам нужно точно подогнать размеры фрагмента, без панели **Transform** не обойтись.

Чтобы вывести на экран панель **Transform**, выберите пункт **Transform** в меню **Window**. Также вы можете выбрать пункт **Transform** подменю **Panels** контекстного меню выделенного фрагмента или нажать комбинацию клавиш <Ctrl>+<T>. Выведенная вами панель показана на рис. 9.11.

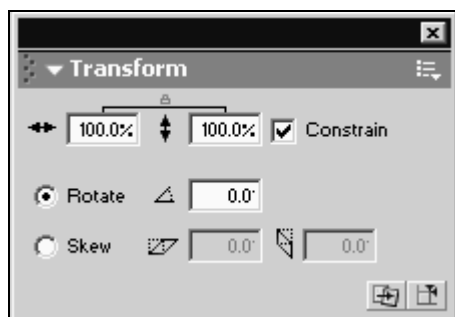


Рис. 9.11. Панель **Transform**

Нужные вам размеры фрагмента вводятся в два верхних поля: горизонтальный размер — в левое поле ввода, вертикальный — в правое. Имейте в виду, что в этом случае они указываются не в обычных единицах измерения, заданных вами при установке параметров документа Flash (см. главу 2), а в процентах относительно текущего размера. Вы можете вводить значения от 1 до 1000%. После ввода не забудьте нажать клавишу <Enter>.

Если вы включите флажок **Constrain**, расположенный правее этих полей ввода, Flash будет сохранять пропорции фрагмента. При этом если вы введете в одно поле ввода новое значение, содержимое второго поля ввода изменится, Flash сам вычислит новое значение, чтобы соблюсти эти пропорции.

Вращение и сдвиг

Вращение и сдвиг выделенного графического фрагмента осуществляются также с помощью инструмента "трансформатор". Однако модификатор для этого используется другой — "вращение и сдвиг". Кнопка, с помощью которой он включается, показана на рис. 9.12. Вы также можете выбрать пункт-выключатель **Rotate and Skew** в подменю **Transform** меню **Modify** или одноименный пункт контекстного меню выделенного фрагмента.



Рис. 9.12. Кнопка модификатора "вращение"

Выделенный фрагмент графики будет иметь при этом вид, показанный на рис. 9.10, т. е. такой же, как при выбранном модификаторе "изменение размеров". Он снова вписан в прямоугольник выделения, на углах которого расположены такие же квадратные маркеры поворота, а на сторонах — маркеры сдвига.

Чтобы повернуть фрагмент, перетаските мышью один из угловых маркеров, не важно, какой. Если вы хотите повернуть фрагмент на угол, кратный 45°, то при перетаскивании маркера удерживайте нажатой клавишу <Shift>. Чтобы выполнить сдвиг, перетаските маркер, находящийся на стороне прямоугольника.

В центре прямоугольника, в который вписан ваш фрагмент, находится белая точка. Это центр вращения фрагмента, вокруг него осуществляется поворот. (Его также называют точкой фиксации.) Вы можете перетаскать его в другое место, даже вынести за пределы прямоугольника. Чтобы вернуть его обратно в центр фрагмента, дважды щелкните по нему мышью.

Если вам нужно быстро повернуть выделенный фрагмент на 90°, воспользуйтесь меню. Пункт **Rotate 90° CW** в подменю **Transform** меню **Modify** поворачивает фрагмент на 90° по часовой стрелке, а пункт **Rotate 90° CWW** — против часовой стрелки. Вместо выбора этих пунктов вы можете нажимать комбинации клавиш <Ctrl>+<Shift>+<9> и <Ctrl>+<Shift>+<7> соответственно.

И, конечно же, вы можете повернуть или сдвинуть фрагмент, воспользовавшись панелью **Transform** (см. рис. 9.11). Выведите ее на экран. После этого включите переключатель **Rotate** и введите в расположенное справа от него поле ввода нужный угол поворота. Чтобы выполнить сдвиг, вам нужно будет включить переключатель **Skew** и ввести в расположенные справа от него поля ввода нужные углы сдвига: горизонтального — в левое поле ввода, вертикального — в правое. Теперь остается нажать клавишу <Enter> — и дело сделано.

Если вам нужно одновременно повернуть и изменить масштаб графического фрагмента, вы можете использовать особую функцию **Flash**. Конечно, можно сделать это и по очереди, сначала изменив размеры фрагмента, а потом повернув его, или наоборот. Но за один раз сделать это, согласитесь, удобнее.

Выберите пункт **Scale and Rotate** в подменю **Transform** меню **Modify** или нажмите комбинацию клавиш <Ctrl>+<Alt>+<S>. На экране появится диалоговое окно **Scale and Rotate**, показанное на рис. 9.13. Задайте в поле ввода **Scale** новый масштаб фрагмента в процентах, а в поле ввода **Rotate** — угол поворота в градусах. После этого останется только нажать кнопку **OK**, если же вы хотите отказаться от манипуляции над фрагментом, нажмите кнопку **Cancel**.

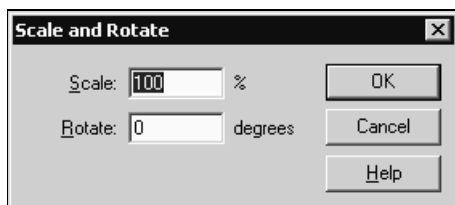


Рис. 9.13. Диалоговое окно **Scale and Rotate**

Искажение формы

На наш взгляд, эта возможность во Flash MX включена зря, и модификатор "искажение формы" на самом деле не нужен. Все, что он предоставляет в распоряжение пользователя, можно сделать, используя старую добрую "стрелку выделения" (см. главу 5). Хотя, может быть, этот модификатор позволяет начинающим пользователям более наглядно изменять форму контура. Поэтому мы его все же рассмотрим.

Чтобы включить модификатор "искажение формы", нажмите кнопку-выключатель, показанную на рис. 9.14. Вы также можете выбрать пункт-выключатель **Distort** в подменю **Transform** меню **Modify** или одноименный пункт контекстного меню выделенного фрагмента. Выделенный вами графический фрагмент примет вид такой как на рис. 9.15. Как видите, центра вращения здесь уже нет, поскольку он и не нужен, — мы не собираемся вращать фрагмент.



Рис. 9.14. Кнопка модификатора "искажение формы"

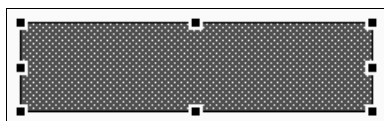


Рис. 9.15. Графический фрагмент, выделенный при включенном модификаторе "искажение формы"

Что же вы можете делать с выделенным таким образом фрагментом? Можно перемещать его углы и стороны. При этом фрагмент будет соответственно "вытягиваться" или "втягиваться", изменяя свою форму. Например, из обычного квадрата можно, "вытянув" один его угол и "втянув" противоположный, получить фигуру такую, как на рис. 9.16.



Рис. 9.16. Результат искажения формы квадрата (слева — изначальный квадрат, справа — деформированный)

С помощью этого модификатора можно исказить только "обычную" графику. Экземпляры, группы и текстовые блоки, а также градиентные заливки не могут быть искажены.

Если же вы все-таки хотите исказить экземпляр, группу или текстовый блок, сначала преобразуйте их в "обычную" графику. Для этого выделите нужный фрагмент и либо выберите пункт **Break Apart** в меню **Modify** или в контекстном меню выделенного фрагмента, либо просто нажмите комбинацию клавиш <Ctrl>+.

Деформация

С появлением Flash MX художникам, работающим с этим программным продуктом, стали доступны средства, позволяющие исказить форму выделенного фрагмента графики. Ранее для этого часто нужно было использовать внешний векторный редактор, например, Macromedia Freehand, после чего импортировать результат во Flash. Теперь же для этого достаточно воспользоваться модификатором "оггибающая" инструмента "трансформатор". С помощью этого модификатора вы можете менять форму фигуры, как вам заблагорассудится.

Для модификатора "оггибающая" действует то же ограничение, что и для "искажения формы". С его помощью вы можете исказить только "обычную" графику. Экземпляры, группы и текстовые блоки, а также градиентные заливки не могут быть искажены.

Чтобы включить модификатор "оггибающая", нажмите кнопку-выключатель, показанную на рис. 9.17. Вы также можете выбрать пункт-выключатель **Envelope** в подменю **Transform** меню **Modify** или одноименный пункт контекстного меню выделенного фрагмента. Выделенный вами графический фрагмент примет вид, показанный на рис. 9.18.



Рис. 9.17. Кнопка модификатора "огигающая"

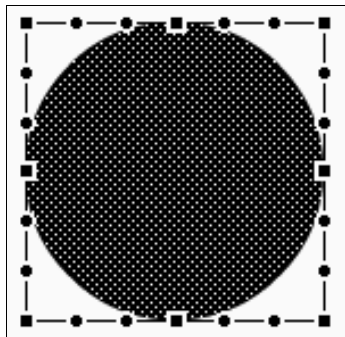


Рис. 9.18. Графический фрагмент, выделенный при включенном модификаторе "огигающая"

Как видите, прямоугольник выделения имеет много маркеров. Принцип работы с графическим фрагментом в этом случае можно выразить одной фразой: выбирайте нужный маркер и тащите его мышью. В результате этого выделенный фрагмент будет искажаться. Например, показанный на рис. 9.18 круг примет вид такой, как на рис. 9.19.

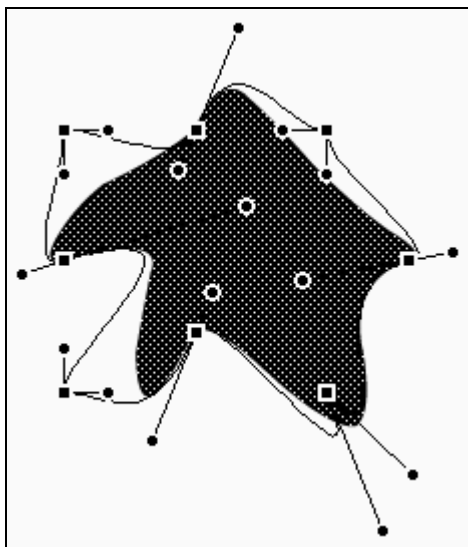


Рис. 9.19. Графический фрагмент, показанный на рис. 9.18, после искажения

Вообще, вам стоит поэкспериментировать с модификатором "огibaющая", чтобы понять, как он работает. Нарисуйте несколько простейших примитивов и испытайте этот модификатор в действии.

Свободная трансформация

Если при выбранном инструменте "трансформатор" не включен ни один модификатор, то фактически все эти модификаторы включены одновременно. В этом случае говорят о так называемой *свободной трансформации*. При свободной трансформации вы можете как угодно изменять выделенный графический фрагмент, не заботясь о том, какой модификатор нужно включить. Однако для того, чтобы применить к выделенному фрагменту требуемое преобразование, следует очень точно позиционировать курсор мыши над нужным маркером.

Чтобы выбрать свободную трансформацию, проще всего отключить все кнопки-выключатели, "ответственные" за модификаторы инструмента "трансформатор". Все эти кнопки находятся в области **Options** инструментария. Также вы можете выбрать пункт **Free Transform** подменю **Transform** меню **Modify** или контекстного меню выделенного фрагмента. Выделенный фрагмент примет вид, показанный на рис. 9.10.

Чтобы выполнить то или иное преобразование, поместите курсор мыши над нужным маркером прямоугольника выделения. Курсор мыши примет при этом "говорящую" форму. Скорее всего, вам придется некоторое время подвигать курсор возле разных маркеров, прежде чем он примет нужную форму. После этого "захватите" маркер и переместите его на новое место, как и в случае других модификаторов.

По нашему мнению, свободная трансформация более пригодна для достаточно опытных художников. Начинающим же будет очень полезно потренироваться, чтобы овладеть этим мощным инструментом Flash MX.

Дополнительные возможности

Ниже описаны некоторые дополнительные возможности по работе с графическими фрагментами, которые могут вам пригодиться.

Преобразования копии графического фрагмента

Часто нужно не просто выполнить какое-либо преобразование над выбранным графическим фрагментом, а создать его преобразованную копию в буфере обмена Windows. После этого вы можете вставить измененную копию этого фрагмента куда угодно. К сожалению, это можно сделать только с теми преобразованиями, которые доступны в панели **Transform** (см. рис. 9.11). К ним относятся изменение размеров, вращение и сдвиг.

Воспользовавшись панелью **Transform**, задайте нужные преобразования. После этого нажмите кнопку, показанную на рис. 9.20, — она расположена в правом нижнем углу этой панели. В буфере обмена будет создана преобразованная копия выделенного фрагмента, в то время как сам выделенный фрагмент не изменится. После этого вы можете вставить эту копию в любое место рабочего листа или даже поместить в другое приложение.



Рис. 9.20. Кнопка создания преобразованной копии выделенного графического фрагмента в буфере обмена

Сброс всех преобразований графического фрагмента

В то время как вы выполняете какие-то манипуляции с выделенным графическим фрагментом — вращаете его, сдвигаете, меняете его размеры, искажаете его форму — Flash сохраняет в памяти его изначальные параметры. Поэтому, если вы захотите вернуться к изначальному фрагменту, такому, каким он был до всех преобразований, вам достаточно только нажать кнопку, расположенную в правом нижнем углу панели **Transform**. Эта кнопка показана на рис. 9.21, она вызывает немедленный сброс всех преобразований. Также вы можете выбрать пункт **Remove Transform** подменю **Transform** меню **Modify** или просто нажать комбинацию клавиш **<Ctrl>+<Shift>+<Z>**. И вы сможете сказать своему графическому фрагменту слова старой песни: "каким ты был, таким остался".



Рис. 9.21. Кнопка сброса всех преобразований

Блокировка фрагмента

Очень часто нужно выполнить какую-либо манипуляцию над графическим фрагментом, являющимся частью сложного изображения. При этом другие части этого изображения, расположенные рядом, мешают выполнить эти манипуляции; если вы попытаетесь, скажем, сдвинуть этот фрагмент, то рискуете сдвинуть и кого-либо из его "соседей". Бывает также наоборот: нужно выполнить манипуляции над "соседями" какого-либо графического фрагмента, не затронув сам этот фрагмент. А поскольку изображение очень сложное, сделать это бывает весьма трудно.

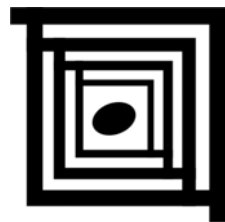
Специально для таких случаев Flash предоставляет возможность *заблокировать* какой-либо фрагмент, а именно, сделать его не выделяемым "стрелкой" и немодифицируемым. Заблокированный фрагмент ведет себя как фон рабочего листа: вы можете щелкать по нему мышью без всякого риска для себя и для него.

Чтобы заблокировать какой-либо фрагмент, прежде всего, преобразуйте его в группу, т. е. сгруппируйте. Как это сделать, описано в *главе 5*, но мы кратко напомним эту несложную процедуру. Выделите нужный графический фрагмент и выберите пункт **Group** меню **Modify** или нажмите комбинацию клавиш <Ctrl>+<G>. Фрагмент примет вид, показанный на рис. 5.38, т. е. будет окружен тонким синим прямоугольником.

Теперь выберите пункт **Lock** подменю **Arrange** меню **Modify**. Но, вероятно, проще и быстрее для вас будет нажать комбинацию клавиш <Ctrl>+<Alt>+<L>. Выделение группы пропадет, и фрагмент будет заблокирован. Попробуйте щелкнуть по нему мышью — он никак не прореагирует.

Закончив нужные действия, разблокируйте этот фрагмент. Для этого выберите пункт **Unlock** подменю **Arrange** меню **Modify** или нажмите комбинацию клавиш <Ctrl>+<Alt>+<Shift>+<L>. И разгруппируйте его, выбрав пункт **Ungroup** All меню **Modify** или нажав комбинацию клавиш <Ctrl>+<Shift>+<G>.

Заметьте, что пункт разблокировки фрагмента носит название **Ungroup All**, а не **Ungroup**. Так как заблокированный фрагмент нельзя выделить, вы не можете разблокировать отдельно взятый фрагмент, оставив другие заблокированными. Поэтому Flash разблокирует сразу все фрагменты — помните об этом.



Глава 10

Образцы и библиотеки. Проводник Flash

В предыдущих главах мы много рисовали, редактировали, стирали, модифицировали, преобразовывали, импортировали, в общем, не теряли времени даром. В этой главе мы рисовать не будем, Хватит, научились и рисовать, и править нарисованное, и стирать ненужное. Поговорим об управлении графикой. Да-да, именно так: будем учиться управлять нашей графикой.

А именно, преобразовывать графические фрагменты в образцы и помещать их в библиотеки. А впоследствии извлекать нужный образец из библиотеки и создавать экземпляры этого образца на рабочем листе. Ну и, конечно, управлять содержащимися в библиотеке экземплярами: править, удалять и т. п.

Но зачем? Кому нужны эти библиотеки и образцы? Зачем нам эта лишняя головная боль?

Давайте возьмем гипотетическое изображение Flash, содержащее множество мелких одинаковых элементов. Не будем придумывать, что это за изображение, просто представим себе его. Каждый его элемент описывается некоторым набором графических примитивов, а, значит, занимает место в памяти и на диске. Десять таких элементов занимает в десять раз больше места, сто элементов — в сто раз больше и т. п. Представляете, во сколько раз увеличится результирующий файл Shockwave/Flash?

Теперь давайте рассуждать так. Что представляет собой этот огромный файл? Повторяющийся набор описаний абсолютно одинаковых элементов — и больше ничего. Мы описываем много раз одно и то же! Нельзя ли нарисовать этот повторяющийся элемент один раз, а потом просто ставить в нужных местах ссылку на его описание? В этом случае мы сэкономим уйму места: ссылка на описание графического элемента занимает несравнимо меньше места, чем сам этот графический элемент. Возможно ли такое сделать во Flash?

Конечно. Если бы не было возможно, мы бы об этом не говорили. В этом случае описание графического элемента помещается в особое хранилище, называемое *библиотекой*, а сам этот элемент получает название *образца*

(в терминологии Flash — symbol). Каждому образцу дается при создании уникальное имя, по которому его можно опознать, и определенный тип, задающий его свойства. Теперь в нужном месте на рабочем листе мы ставим ссылку на этот образец, создавая тем самым *экземпляр образца* или просто *экземпляр* (в терминологии Flash — instance).

Такой подход имеет еще одно преимущество. Вы можете изменять все экземпляры одного образца, просто отредактировав этот образец в библиотеке. После этого Flash сам обновит все экземпляры. Видите, как удобно? Единственный недостаток: вы не сможете изменять созданный таким образом экземпляр так же свободно, как обычную графику. Но этим во многих случаях можно пожертвовать, не так ли?

Более того — вы можете менять некоторые параметры этого экземпляра. Например, сменить его цвет, прозрачность, повернуть его, исказить или изменить его размеры. И, тем не менее, он останется экземпляром. Если вы отредактируете образец в библиотеке, этот измененный экземпляр также обновится.

Все файлы, что вы импортировали в документ Flash, кроме изображений формата Shockwave/Flash, также оказываются в библиотеке. Все изображения и фильмы, созданные в других программах, все звуки оказываются там и превращаются в образцы. (Кроме изображений в формате Shockwave/Flash — эти помещаются прямо на рабочий лист, и вам придется вручную добавить их в библиотеку.) Это очень удобно: вы можете помещать экземпляры импортированного изображения или звука на лист сколько угодно раз, а размер файла при этом увеличиваться не будет. (Хотя, конечно, все-таки увеличится — при самом импорте.)

И еще. Если вы собираетесь создавать во Flash анимированные изображения (фильмы), вам так или иначе не избежать работы с образцами и библиотеками. Ведь некоторые виды анимации могут быть применены только к экземплярам образцов, хранящихся в библиотеке. Подробнее об анимации вы можете узнать в *части 3* этой книги.

Flash также позволяет вам создавать разделяемые библиотеки. Такие библиотеки вы можете выкладывать в Интернет, чтобы другие пользователи Flash могли использовать их содержимое. При этом проигрыватель Flash, загрузив изображение, в котором был использован образец из разделяемой библиотеки, автоматически загружает файл библиотеки и извлекает из нее нужный образец. Таким образом, размер файла Shockwave/Flash уменьшится еще сильнее. Однако если проигрыватель Flash не сможет загрузить файл разделяемой библиотеки, изображение не будет показано правильно.

А напоследок мы поговорим о мощном инструменте организации вашей Flash-графики. Это так называемый Проводник Flash, аналогичный по назначению Проводнику Windows. С его помощью вы сможете быстро оты-

скивать образцы, экземпляры и прочие графические элементы, содержащиеся в вашем творении.

Итак, приступим...

Работа с образцами и экземплярами

Здесь мы изучим работу с образцами и экземплярами. Но сначала нужно узнать о типах образцов, поддерживаемых Flash.

Типы образцов

Flash поддерживает непосредственное создание образцов трех различных *типов*. Тип образца задается при его создании и впоследствии может быть изменен. Каждый из этих типов обладает своими особыми свойствами и имеет свою особую область применения. Давайте их перечислим и опишем, дав в скобках наименование, принятое в терминологии Flash.

Графический образец (graphic) представляет собой обычное статичное или анимированное изображение, сделанное во Flash или импортированное из другой программы. Это, вероятно, наиболее часто используемый тип образцов.

Образец-кнопка (button) — это особый образец, представляющий собой обычную командную кнопку. Такие образцы используются для создания графического пользовательского интерфейса для написанных на Flash программ и рассматриваются во всех подробностях в *главе 20*.

Образец-клип (movie clip) представляет собой настоящий фильм, созданный во Flash или другом программном пакете, импортированный в текущий документ Flash и помещенный на рабочий лист как часть изображения. Такой образец "работает" совершенно независимо от основного изображения, частью которого является. Образцы-клипы применяются для создания очень сложной анимации. Кроме того, по сравнению с графическими образцами, образцы-клипы предоставляют дополнительные возможности для программирования своего поведения, поэтому часто используются при создании пользовательского интерфейса. Мы изучим образцы-клипы во время рассмотрения создания анимации в среде Flash (*часть 3 книги*).

Flash предоставляет также возможность переопределить тип на уровне не образца, а экземпляра. Это значит, что вы можете поместить на лист экземпляр какого-либо образца и переопределить его — экземпляра — тип; тип образца при этом не изменится. Сейчас это не принесет нам пользы, но в дальнейшем, при изучении анимации и программирования (*части 2 и 3*), может пригодиться.

Кроме перечисленных выше трех типов, Flash поддерживает еще три: *образец-растровое изображение* (bitmap), *образец-звук* (sound) и *образец-импортированный клип* (embedded video). Их также собирательно называют

импортированными образцами. Как вы уже поняли, образцами-растровыми изображениями становятся импортированные в документ растровые изображения, а образцами-звуками — импортированные звуки. Эти два типа образцов не могут быть созданы непосредственно во Flash, в отличие от графических образцов, кнопок и образцов-клипов.

Существует еще один — седьмой по счету — тип образцов. Это *образец-шрифт* (font). Подробнее мы его рассмотрим в конце этой главы.

Создание образцов

Разобравшись с тремя типами образцов, давайте приступим к их созданию. Как говорится, от теории к практике.

Есть два пути создания образца. Во-первых, вы можете нарисовать на рабочем листе какой-либо графический элемент, а потом превратить его в образец и поместить в библиотеку. Во-вторых, вы можете создать "пустой" образец и поместить его в библиотеку, а уже потом "наполнить" его графическим содержимым. Выбирайте сами, каким путем пойти. Если вы рисуете образец "с нуля", то, вероятно, идеальным будет второй путь. Если же вы хотите преобразовать в образец уже нарисованный графический фрагмент, идите первым путем.

Предположим, что у нас на рабочем листе уже что-то нарисовано (рис. 10.1). Давайте преобразуем это "что-то" в образец.



Рис. 10.1. Графический элемент, который будет преобразован в образец

Выделите графический элемент на листе и выберите пункт **Convert to Symbol** в меню **Insert**, в контекстном меню выделенного графического фрагмента или нажмите клавишу <F8>. На экране появится небольшое диалоговое окно **Convert to Symbol**, показанное на рис. 10.2.

Каждый образец, помещенный в библиотеку, должен иметь уникальное имя, по которому его можно однозначно идентифицировать. Это имя вводится в поле ввода **Name**; проследите, чтобы оно было уникальным, иначе Flash не сможет создать образец, о чем и предупредит вас. Имейте в виду, что такое имя может включать в себя только буквы латинского алфавита, цифры и знак подчеркивания, причем начинаться имя должно на букву.

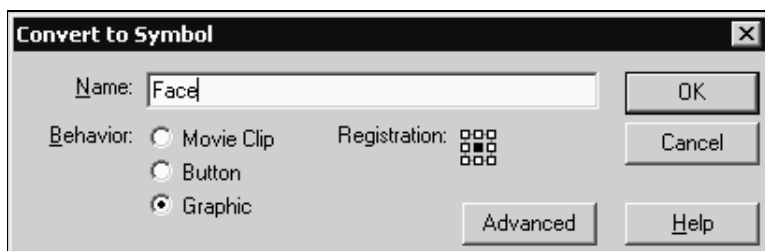


Рис. 10.2. Диалоговое окно **Convert to Symbol**

А с помощью набора переключателей **Behavior** задается тип образца: графический (переключатель **Graphic**), кнопка (**Button**) или клип (**Movie Clip**).

Правее набора переключателей **Behavior** находится элемент управления **Registration**. С его помощью задается местоположение *точки фиксации* образца. Точка фиксации — это точка, относительно которой будет вестись отсчет местоположения экземпляра этого образца и будет выполняться вращение экземпляра. Как видите, существует девять мест, куда может быть помещена точка фиксации образца, по умолчанию она находится в его центре. Щелкните мышью по одной из белых точек элемента управления **Registration**, чтобы задать новое местоположение для точки фиксации или оставьте ее в центре образца.

Закончив задание параметров образца, нажмите кнопку **OK**, чтобы поместить его библиотеку. Если вы передумали создавать новый образец, нажмите кнопку **Cancel**.

После того, как созданный нами образец будет добавлен в библиотеку, графический элемент, на основе которого он был создан, станет его экземпляром. Так что ваше изображение не будет испорчено.

Теперь, чтобы посмотреть на созданный нами образец, нужно открыть окно библиотеки. Для этого выберите пункт **Library** в меню **Window** или нажмите клавишу <F11>. Окно библиотеки показано на рис. 10.3.

Кратко опишем окно библиотеки. Оно разделено на две части. В нижней части находится список уже созданных образцов, вы можете выбирать эти образцы, щелкая мышью по их именам. Левее имен в этом списке находятся небольшие изображения, обозначающие тип образца. В верхней части находится панель предварительного просмотра, где отображается содержимое образца, выбранного в нижней части окна. Вы можете менять размеры этих частей, перетаскивая мышью серую полосу, разделяющую их. И, конечно же, вы можете менять размеры самого окна библиотеки.

Если у вас на экране открыто окно библиотеки, можно просто перетаскивать в него выделенные фрагменты графики, чтобы превратить их в образцы. В этом случае на экране также появится диалоговое окно **Convert to Symbol** (см. рис. 10.2), где вы будете должны задать параметры вновь создаваемого образца.



Рис. 10.3. Окно библиотеки образцов

Подробнее окно библиотеки и работу с ним мы опишем далее в этой главе. Сейчас же давайте рассмотрим создание образца "с нуля".

Чтобы поместить в библиотеку новый "пустой" образец, сделайте следующее. Выберите пункт **New Symbol** в меню **Insert** или нажмите комбинацию клавиш <Ctrl>+<F8>. Если у вас открыто окно библиотеки, вы также можете выбрать пункт **New Symbol** в дополнительном меню этого окна или нажать кнопку, показанную на рис. 10.4, эта кнопка находится в нижнем левом углу окна библиотеки.



Рис. 10.4. Кнопка **New Symbol** окна библиотеки

После этого рабочий лист очистится, и вы сможете нарисовать нужный вам образец. В центре листа виден небольшой черный крестик — это точка фиксации образца. К сожалению, вы не сможете ее сместить на другое место — вместо этого вам придется сместить сам нарисованный образец.

Закончив рисование образца, выберите пункт **Edit Document** в меню **Edit** или нажмите комбинацию клавиш <Ctrl>+<E>. Также вы можете нажать кнопку, расположенную над рабочим листом слева (см. рис. 5.39). Flash вернется

в обычный режим работы — редактирование всего изображения. А в окне библиотеки появится новый образец.

Ну и, конечно, вы можете создать новый образец, импортировав в документ растровое изображение или звук. Для этого можно выбрать пункт **Import** в меню **File**, в результате этого импортированное вами изображение (а на самом деле, его экземпляр) появится на рабочем листе. Если же вы хотите этого избежать, то выберите в меню **File** пункт **Import to Library**, в этом случае на рабочем листе не появится ничего нового.

Внимание!

При импорте вновь созданный образец получает имя, совпадающее с именем исходного файла, из которого он был импортирован.

Создание экземпляров

Итак, мы изучили, как создаются образцы и вкратце познакомились с окном библиотеки. Теперь рассмотрим, как создавать экземпляры этих самых образцов и работать с ними.

Экземпляры создаются очень просто. Для этого достаточно вывести на экран окно библиотеки, выбрать в нем нужный образец и перетащить его на рабочий лист. На листе будет создан новый экземпляр выбранного вами образца (рис. 10.5).

Обратите внимание, что созданный вами экземпляр выделен тонким синим прямоугольником, как группа графических фрагментов. Это значит, что вы не можете его править, как обычную графику. Чтобы изменить экземпляр, вам нужно будет отредактировать образец, на основе которого он был создан. Также обратите внимание, что в его центре небольшим кружком помечена точка фиксации.

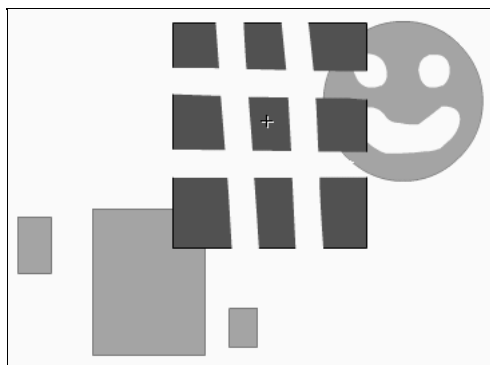


Рис. 10.5. Образец в окне библиотеки и созданный на его основе экземпляр

Преобразование экземпляров

Мы уже говорили, что Flash позволяет изменить некоторые параметры созданных экземпляров. В частности, вы можете изменять их цвета, задавать различные преобразования и пр. Давайте посмотрим, что же реально вы можете делать.

Изменение цвета экземпляров

Изменение цвета экземпляра выполняется с помощью редактора свойств. Если выделить экземпляр, то он примет вид, показанный на рис. 10.6.

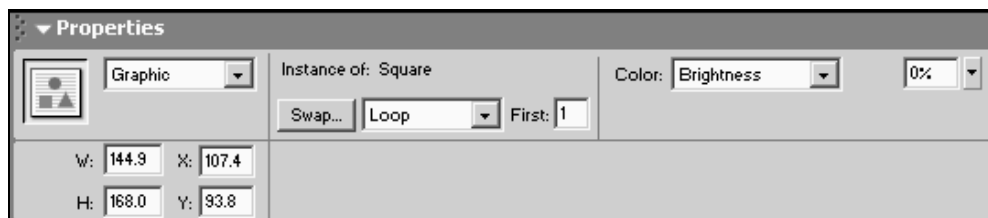


Рис. 10.6. Вид редактора свойств при выделенном экземпляре

Цветовой эффект задается с помощью раскрывающегося списка **Color**, расположенного в правой части редактора свойств. В этом списке доступны пять пунктов: **None**, **Brightness**, **Tint**, **Alpha** и **Advanced**. В зависимости от того, какой пункт в этом списке выбран, редактор свойств меняет свой вид, предоставляя пользователю дополнительные элементы управления для задания дополнительных параметров.

Давайте рассмотрим все пункты списка **Color** по порядку.

При выборе пункта **None**, никаких дополнительных элементов управления в редакторе свойств не появляется. А все потому, что этот пункт отключает все цветовые эффекты, примененные к экземпляру. Можете считать, что этот пункт списка возвращает экземпляр в первоначальное состояние.

Если выбран пункт **Brightness**, как на рис. 10.6, вы можете изменить общую яркость экземпляра, т. е. общую яркость всех цветов всех графических фрагментов, составляющих экземпляр. При выборе этого пункта правее раскрывающегося списка эффектов появится поле ввода с регулятором. Введите в это поле нужное значение от -100% до 100% ; отрицательные значения делают экземпляр темнее, положительные — ярче.

Когда выбран пункт **Alpha**, вы можете изменить общую прозрачность экземпляра. При выборе этого пункта левее раскрывающегося списка эффектов появится поле ввода с регулятором. Введите в это поле нужное значение от 0 до 100%. Чем меньше это значение, тем прозрачнее экземпляр, значение равное нулю делает его совсем невидимым.

Если выбран пункт **Tint**, вы можете задать оттенок для всего экземпляра, т. е. цвет, в который окрасится экземпляр целиком. При выборе этого пункта в редакторе свойств появится несколько элементов управления, с помощью которых вы сможете задать этот цвет (рис. 10.7).

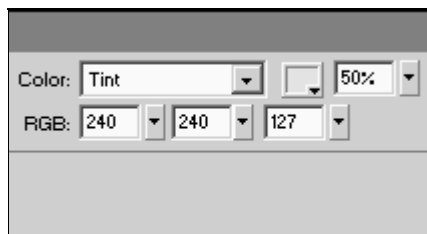


Рис. 10.7. Фрагмент редактора свойств (выбран пункт **Tint** списка цветовых эффектов)

Вы можете задать оттенок, выбрав его в расположенном правее списка **Color** селекторе цвета. Задать новый оттенок можно и другим способом, введя соответственно его красную, зеленую и синюю составляющие в поля ввода с регуляторами **R**, **G** и **B**.

Правее селектора цветов находится еще одно поле ввода с регулятором. С его помощью вы можете задать степень окрашенности экземпляра выбранным вами цветом. Допустимый диапазон значений от 0 до 100%. Чем больше это значение, тем сильнее преобладает заданный вами оттенок, значение равное нулю убирает его совсем, а значение равное 100% закрашивает им весь экземпляр, полностью "забивая" его изначальные цвета.

Последний пункт раскрывающегося списка цветовых эффектов — **Advanced** — позволяет задать более сложные цветовые эффекты. При его выборе в редакторе свойств появится кнопка **Settings**, при нажатии которой на экране появится диалоговое окно **Advanced Effect** (рис. 10.8).

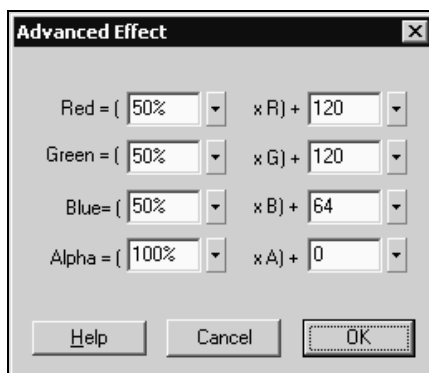


Рис. 10.8. Диалоговое окно **Advanced Effect**

Диалоговое окно **Advanced Effect** содержит четыре группы из двух полей ввода с регулятором каждая: **Red**, **Green**, **Blue** и **Alpha**. Они служат для изменения исходных цветов (красного, зеленого, синего и прозрачности) экземпляра. Это изменение задается следующим образом.

Возьмем для примера самую верхнюю группу полей ввода, задающую изменение красного цвета. Обозначим исходную долю красной составляющей R_0 , значения, введенные в левое и правое поля ввода, — R и r , а окончательную долю красной составляющей — R_1 . Тогда эта самая окончательная доля будет вычисляться по формуле

$$R_1 = R_0 \times R + r.$$

Например, предположим, что изначальная доля красной составляющей цвета экземпляра была 128, в левое поле ввода было введено значение 80%, а в правое — 24. Тогда окончательная доля красной составляющей будет равна

$$128 \times 0,8 + 24 = 126,4 \approx 126.$$

Как видите, вы можете регулировать цвета экземпляра достаточно точно. Это может очень пригодиться, если вы собираетесь в дальнейшем превратить это изображение в фильм. Ну, а сейчас что ж, тоже полезно.

Преобразования экземпляра

Вы можете выполнять над экземпляром те же действия, что доступны для любого графического фрагмента: вращение, изменение размеров, сдвиг. Также можно изменить точку привязки экземпляра. Как выполняются эти операции описано в *главе 9*. Однако вы не сможете ни деформировать экземпляр, используя модификатор "огibaющая" инструмента "трансформатор", ни исказить его с помощью модификатора "искажение формы".

Изменение типа экземпляра

Вам уже известно, что каждый образец в библиотеке может принадлежать к одному из трех типов: графический образец, кнопка или образец-клип. И вы также знаете, что можно сменить тип экземпляра того или иного образца, не меняя типа самого образца. Эта операция выполняется с помощью редактора свойств. Выделите нужный экземпляр и выберите в раскрывающемся списке, расположенном ниже имени образца, новый тип. В этом списке доступны три пункта: **Graphic** (графический образец), **Button** (образец-кнопка) и **Movie Clip** (образец-клип), их назначение должно быть понятно без дополнительного разъяснения.

Смена экземпляра

Иногда бывает необходимо заменить один экземпляр на другой, основанный на другом образце, сохранив при этом все преобразования, выполненные над этим экземпляром. Для такого случая в редакторе свойств располо-

жена кнопка **Swap** (см. рис. 10.6). При нажатии на нее на экране появляется диалоговое окно **Swap Symbol** (рис. 10.9). Если у вас на экране нет редактора свойств, вы можете выбрать пункт **Swap Symbol** в меню **Modify**.

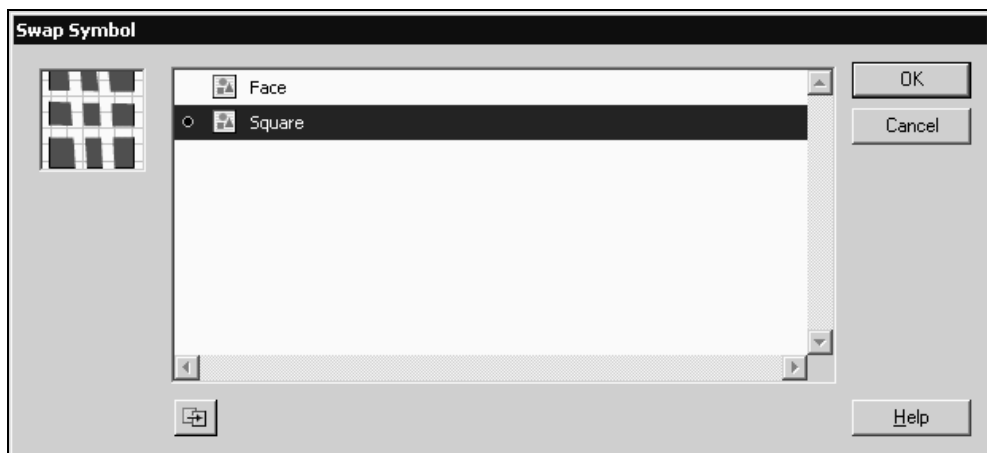


Рис. 10.9. Диалоговое окно **Swap Symbol**

В списке, занимающем большую часть этого окна, перечислены все имеющиеся в библиотеке текущего документа Flash образцы. Образец, на основе которого был создан выделенный на листе экземпляр, помечен хорошо заметной черной точкой. Выберите нужный образец. В небольшом поле предварительного просмотра, расположенном левее и выше списка, будет показано его содержимое. После этого вам останется только нажать кнопку **ОК**, чтобы выполнить смену образца, или кнопку **Cancel** для отказа от этого.

Имейте в виду, что новый экземпляр унаследует все преобразования старого. Так, если вы сделали старый экземпляр полупрозрачным и повернули его на какой-то угол, новый тоже будет полупрозрачным и повернутым.

Внимание!

Если вы заменили один экземпляр другим, основанном на другом образце, в ключевом кадре анимации, то вам нужно будет сделать то же самое и во всех остальных ключевых кадрах этой анимации, в которых присутствует этот экземпляр. (Об анимации и ключевых кадрах см. главу 12.)

Если вы хотите заменить экземпляр растрового изображения другим того же типа, то вы можете также воспользоваться диалоговым окном **Swap Bitmap** (см. рис. 8.11). Чтобы вызвать его на экран, выделите нужное изображение и либо нажмите кнопку **Swap** в редакторе свойств, либо выберите пункт **Swap Bitmap** меню **Modify**.

Преобразование экземпляра в обычный графический элемент

Одним из ограничений системы библиотечных образцов и экземпляров является некоторая ее негибкость. Вы не можете править экземпляры с той же легкостью, как обычные, независимые графические фрагменты. Эту негибкость лучше всего описать на примере.

Предположим, что вы создали образец, а на его основе — несколько экземпляров. После этого вам понадобилось очень сильно изменить один экземпляр этого образца, например, с использованием модификатора "оггибающая" инструмента "трансформатор". Flash не позволит вам этого сделать, т. к. этот экземпляр — фактически копия библиотечного образца с возможностью очень ограниченной правки. И, чтобы изменить эту копию, вам придется изменять сам образец.

Здесь возможны два пути. Первый путь — создание нового образца и порождение от него нового экземпляра. Этот путь оправдан, если вам нужно создать несколько экземпляров такого образца, тогда налицо выигрыш в размере файла и временных затратах. Если же вам нужно создать всего один экземпляр, лучше пойти по второму пути — нарисовать обычный графический элемент. В этом случае размер файла останется неизменным, а затраты времени и труда на создание изображения будут существенно ниже за счет того, что вам не придется создавать библиотечный образец.

Во многих случаях проще не рисовать элемент заново, а взять уже существующий экземпляр образца и преобразовать его в обычную, независимую графику. Для этого выделите нужный экземпляр и либо выберите пункт **Break Apart** в меню **Modify**, либо нажмите комбинацию клавиш <Ctrl>+. После этого вы можете работать с получившимся элементом как с обычной графикой.

Работа с образцами

Мы научились работать с экземплярами библиотечных образцов. Давайте теперь выясним, как работать с самими образцами.

Изменение образцов

Если вам нужно изменить сразу все экземпляры, лучший способ сделать это — изменить сам образец. Сразу после того, как образец будет изменен, Flash обновит все созданные на его основе экземпляры.

Flash предоставляет целых три способа изменить образец:

- ☐ в обычном режиме редактирования образца;
- ☐ "на месте";
- ☐ в отдельном окне Flash.

Вы можете выбрать тот способ, который вам более удобен.

Проще всего переключиться в режим редактирования образца "на месте". При этом вы сможете изменить образец прямо на рабочем листе, где находится ваше изображение. Вся окружающая графика будет закрашена серым и недоступна для изменения — вы можете изменять только сам образец (рис. 10.10). Возможно, не всем из вас это покажется удобным: окружающая графика может перекрывать его, мешая работе. С другой стороны, можно оценить, как исправленный вами образец будет "смотреться" на своем месте в изображении.

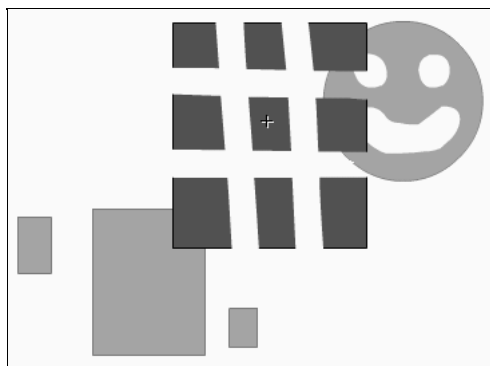


Рис. 10.10. Образец (выделен черным) в режиме редактирования "на месте"

Чтобы переключиться в режим редактирования образца "на месте", дважды щелкните по любому его экземпляру. Вы также можете выбрать пункт **Edit in Place** меню **Edit** или контекстного меню нужного экземпляра или пункт **Edit Selected** меню **Edit**.

Режим редактирования образца в отдельном окне Flash имеет то преимущество, что вы можете править образец, одновременно имея перед глазами основное изображение. Конечно, для этого вам придется изменить размеры открытых окон Flash, но вы уже знаете, как это сделать (см. главу 1). Недостатком может служить то, что не все любят манипулировать несколькими окнами, открытыми в одном приложении.

Окно, в котором открыт образец, имеет заголовок вида

<Имя файла документа>.fla – Library:<Порядковый номер образца>,

так что вы сразу увидите, в каком окне находитесь.

Чтобы переключиться в режим редактирования образца в новом окне, выберите пункт **Edit in New Window** контекстного меню нужного экземпляра. Других способов сделать это нет.

Обычный режим редактирования образцов — это когда Flash скрывает изображение, выводя на его место выбранный образец. При этом во Flash оста-

ется открытым одно окно (в смысле, дополнительных окон для текущего документа не открывается). Пользователь видит перед собой только образец, который он выбрал для изменения, — и больше ничего.

Переключиться в обычный режим правки образца можно восемью разными способами. Вы можете:

- ❑ выбрать пункт **Edit Symbol** меню **Edit**;
- ❑ выбрать пункт **Edit Selected** того же меню, выделив экземпляр нужного образца;
- ❑ нажать комбинацию клавиш <Ctrl>+<E>;
- ❑ в окне библиотеки (см. рис. 10.3) дважды щелкнуть либо иконку слева от названия нужного образца, либо его изображение в панели предварительного просмотра;
- ❑ выбрать пункт **Edit** в контекстном меню списка образцов окна библиотеки;
- ❑ выбрать пункт **Edit** в контекстном меню нужного экземпляра образца на рабочем листе;
- ❑ щелкнуть расположенную справа над рабочим листом кнопку **Edit Symbols** и выбрать имя нужного образца в появившемся на экране подменю (рис. 10.11).

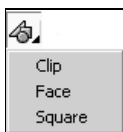


Рис. 10.11. Кнопка и раскрытое подменю **Edit Symbols**

Выбирайте, какой способ для вас самый удобный.

Для редактирования образца можно применять все уже знакомые вам приемы работы с графикой. Единственное исключение: вам не удастся изменить точку фиксации образца штатными средствами. Для этого вам придется выделить весь образец и переместить его так, чтобы точка фиксации пришлась на другое, нужное вам, место образца. (Однако штатными средствами вы можете изменить точку фиксации у экземпляра этого образца.)

Чтобы выйти из режима редактирования образца, проще всего выбрать пункт **Edit Document** меню **Edit**, нажать комбинацию клавиш <Ctrl>+<E> или щелкнуть кнопку, расположенную над рабочим листом слева (см. рис. 5.39). Если вы находитесь в режиме редактирования образца "по месту", вы также можете дважды щелкнуть мышью по листу где-нибудь вне образца. Если же вы находитесь в режиме редактирования образца в отдельном окне Flash, просто закройте это окно.

Дублирование образцов

Мы уже рассказывали, как преобразовать экземпляр в обычный графический элемент. Этот подход выгоден тогда, когда вам нужно создать графический элемент, отличающийся от имеющегося в библиотеке образца. (Вспомните, ведь Flash не позволяет править экземпляры, за исключением самых элементарных преобразований. С обычной же графикой вы можете делать все, что угодно.)

Однако если нужно создать не один такой графический элемент, а несколько, описанный подход не работает. Выгоднее, опять же, создать образец, а на его основе — соответствующее количество экземпляров. К счастью, Flash предоставляет возможность продублировать образец, создав его копию под другим именем. После этого вы сможете немного подредактировать новый образец, и создать на его основе нужное количество экземпляров.

Чтобы создать копию какого-либо образца, сначала выберите на листе любой созданный на его основе экземпляр. После этого выберите пункт **Duplicate Symbol** меню **Modify**. На экране появится диалоговое окно **Symbol Name** (рис. 10.12).

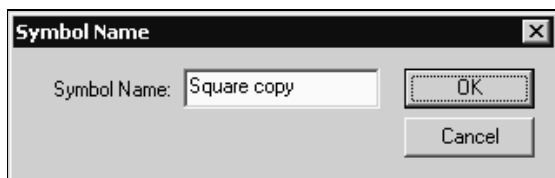


Рис. 10.12. Диалоговое окно **Symbol Name**

Что делать с этим диалоговым окном, понятно. Введите в единственное поле **Symbol Name** имя создаваемого образца и нажмите кнопку **OK**. Если вы передумали создавать новый образец, нажмите кнопку **Cancel**.

После этого Flash выполнит следующие действия. Сначала он создаст новый образец, являющийся полной копией того, на основе которого был создан выделенный на листе экземпляр. Далее он автоматически заменит выделенный экземпляр другим, созданным на основе уже нового образца, и также его выделит. Так что вам останется только дважды щелкнуть по этому экземпляру и отредактировать вновь созданный образец.

Если у вас на экране открыто окно библиотеки, вы можете выделить нужный образец в списке и выбрать пункт **Duplicate** в контекстном меню или дополнительном меню окна. На экране появится диалоговое окно **Symbol Properties**, аналогичное по виду диалоговому окну **Convert to Symbol** (см. рис. 10.2), где вы можете задать не только имя создаваемого образца, но и его тип. Нажмите кнопку **OK** — и новый образец будет создан. Однако создавать новые экземпляры на основе этого образца вам уже придется самим.

И, наконец, вы можете создать копию образца, нажав кнопку, расположенную ниже списка образцов в окне **Swap Symbol** (см. рис. 10.9). После этого опять же на экране появится диалоговое окно **Symbol Name**. Введя в него имя нового образца и нажав кнопку **OK**, не забудьте также нажать **OK** в самом окне **Swap Symbol**. Одновременно вы сможете заменить выделенный экземпляр другим, созданным на основе нового образца, для этого просто выберите в списке образцов нужный пункт.

Работа с библиотекой

Вот мы и узнали основные приемы работы с образцами и экземплярами. Теперь настала пора выяснить, как управлять образцами в библиотеке: изменять их параметры, удалять, искать и т. п. Давайте же это выясним.

Окно библиотеки

Прежде всего, еще раз, но уже более внимательно, посмотрим на окно библиотеки (см. рис. 10.3). Что оно в себе содержит?

Известно что, скажете вы. Список образцов и панель предварительного просмотра. Все знакомо.

Все, да не все.

Если вы внимательно посмотрите на список образцов, расположенный в нижней части окна библиотеки, то увидите, что он представляет собой многоколоночную таблицу. Перечислим все имеющиеся в нем колонки:

- **Name** — имя образца;
- **Kind** — тип образца (Graphic, Button, Movie Clip, Bitmap, Sound или Font);
- **Use Counts** — количество экземпляров, созданных на основе этого образца, если ни одного экземпляра не было создано, отображается прочерк;
- **Linkage** — состояние внешнего образца (о внешних образцах см. ниже в этой главе);
- **Date Modified** — дата последнего изменения образца.

Вы можете изменять ширину любой колонки списка, перетаскивая мышью ее правую границу. Этот прием одинаков во всех Windows-приложениях, использующих подобные списки. Также можно сортировать значения в списке, щелкая по заголовку нужной колонки. А кнопка, расположенная правее списка вровень с заголовками колонок, позволит вам изменить порядок сортировки на противоположный (рис. 10.13).



Рис. 10.13. Кнопка, изменяющая порядок сортировки строк списка на противоположный

Нетрудно заметить, что значения, отображаемые в колонке **Use Counts**, не всегда соответствуют действительности. Например, вы можете создать на листе несколько экземпляров какого-либо образца и обнаружить, что в этой колонке против его имени стоит прочерк. В этом случае выберите пункт **Update Use Counts Now** дополнительного меню окна библиотеки — и список будет обновлен.

Для включения (выключения) автоматического обновления списка образцов при каждом создании и удалении экземпляров служит пункт-выключатель **Keep Use Counts Updated**. По умолчанию этот пункт включен, но если вы находите, что постоянное обновление счетчиков замедляет работу вашего компьютера, можете его выключить.

Ниже кнопки изменения порядка сортировки находятся еще две кнопки, показанные на рис. 10.14. Это кнопки быстрого изменения размера окна библиотеки. Верхняя кнопка расширяет окно так, чтобы был виден весь список образцов. Прежде чем нажимать ее, подумайте — вполне возможно, окно библиотеки в этом случае займет по ширине весь экран. Нижняя же кнопка сужает окно так, чтобы была видна только колонка имен образцов. И, конечно, вы можете сами менять размер окна библиотеки, как любого другого окна Windows.



Рис. 10.14. Кнопки быстрого изменения размера окна библиотеки

Над панелью предварительного просмотра отображается небольшая строка статуса. В ней показывается общее количество образцов в библиотеке. (В это количество входят также все папки, о которых рассказывается ниже в этой главе.)

Щелкнув по панели предварительного просмотра правой кнопкой мыши, вы вызовете небольшое контекстное меню. Выбрав пункт-выключатель **Show Grid**, вы включите или отключите показ координатной сетки. А два других пункта — **Movie's Background** и **White Background** — работают как двухпозиционные переключатели. Первый пункт отображает содержимое панели просмотра на том фоне, что был задан для рабочего листа, а второй — на белом фоне.

И, наконец, вы можете выделять в списке сразу несколько образцов. Для этого щелкните первый образец, нажмите клавишу <Ctrl>, удерживая ее, и продолжайте щелкать по остальным образцам, которые вы хотите выделить. Можно также щелкнуть по первому образцу, нажать клавишу <Shift> и, удерживая ее, щелкнуть по другому, — все образцы, находящиеся в списке между этими двумя, будут выделены.

Управление образцами

Вы уже знаете, как добавлять в библиотеку новые образцы, как их править и изменять их тип. Давайте поговорим о других действиях, которые можно выполнить над ними.

Вы можете изменить имя любого образца, независимо от того, созданы ли на его основе экземпляры или нет. Для этого дважды щелкните по имени (не по иконке) нужного образца в списке. Можно также воспользоваться пунктом **Rename** контекстного меню списка или дополнительного меню окна. После этого в списке вместо имени этого образца появится небольшое поле ввода, в котором будет подставлено старое имя. Введите новое имя и нажмите клавишу <Enter> для его сохранения или <Esc> для отказа от изменения имени.

Кроме того, вы можете изменить имя образца, воспользовавшись диалоговым окном **Symbol Properties** (см. рис. 10.2). Выберите пункт **Properties** контекстного меню списка или дополнительного меню окна (нужный образец должен быть выбран в списке). Также можно щелкнуть кнопку **Properties**, расположенную на нижней границе окна библиотеки (рис. 10.15). В появившемся на экране диалоговом окне **Symbol Properties** введите в поле **Name** новое имя образца, а с помощью набора переключателей **Behavior** выберите его тип. После этого нажмите кнопку **OK** для сохранения сделанных установок или **Cancel** — для отказа от этого.



Рис. 10.15. Кнопка **Properties** окна библиотеки

Окно задания параметров импортированных растровых изображений было показано на рис. 8.12. Оно называется **Bitmap Properties** и также позволяет задать имя образца. Также оно позволяет установить параметры сжатия изображения, обо всем этом подробно рассказывалось в *главе 8*.

Если вы переименовали образец, на основе которого уже были созданы экземпляры, то эти экземпляры все равно останутся привязанными к нему. Flash корректно изменит все ссылки.

Чтобы исправить образец, созданный в самом Flash, дважды щелкните по его значку, отображаемому слева от имени (не по самому имени!). Вы также можете воспользоваться пунктом **Edit** контекстного меню списка или дополнительного меню окна. Это все вам знакомо.

Кроме того, если вы вызовете окно **Symbol Properties** для уже существующего в библиотеке образца, в нем будет доступна кнопка **Edit**. Нажав эту кнопку, вы сможете изменить образец в обычном режиме правки.

Но что делать, если нужно исправить образец, полученный в результате импорта растрового изображения или звука? В этом случае Flash предлагает воспользоваться программой, зарегистрированной в системе для обработки файла данного типа. Стало быть, вы можете выбрать в контекстном меню списка или дополнительном меню окна пункт **Edit with <Имя зарегистрированной программы>**.

Часто, однако, бывает по-другому. А именно, для какого-то типа файлов не зарегистрировано никакой программы по умолчанию. Тогда вам нужно воспользоваться пунктом **Edit with**. В этом случае на экране появится стандартное диалоговое окно открытия файла Windows, выберите исполняемый файл нужной программы и нажмите кнопку открытия.

Вы также можете отредактировать исходный файл изображения или звука, используя любую программу, поддерживающую его формат. Недостатком такого подхода является то, что после редактирования вам необходимо будет обновить основанный на этом файле образец в библиотеке. Для этого выделите в списке образцов нужный и в контекстном или дополнительном меню выберите пункт **Update**. На экране появится диалоговое окно **Update Library Items** (рис. 10.16).

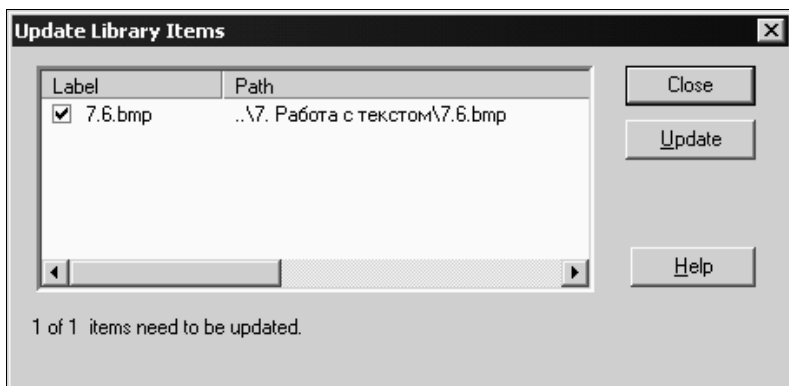


Рис. 10.16. Диалоговое окно **Update Library Items**

Большую часть этого диалогового окна занимает список образцов, созданных из импортированных файлов. Этот список представляет собой таблицу, состоящую из двух колонок: **Label** (имя образца, оно же — имя исходного файла) и **Path** (путь к исходному файлу). Левее значения, стоящего в колонке **Label**, находится флажок, если он включен, данный файл будет обновлен. Включите флажки против нужных файлов и нажмите кнопку **Update**. После обновления образцов нажмите кнопку **Close**.

Если в списке образцов выбран звук, вы можете проиграть его прямо в окне библиотеки. Для этого выделите нужный образец и выберите пункт **Play** контекстного или дополнительного меню. Чтобы остановить проигрывание,

выберите пункт **Stop**. Но удобнее будет воспользоваться кнопками, появляющимися в верхнем правом углу панели предварительного просмотра (рис. 10.17). Правая кнопка запускает проигрывание, а левая — останавливает. К сожалению, Flash не может проигрывать таким же образом импортированные фильмы.



Рис. 10.17. Кнопки запуска и остановки проигрывания, находящиеся в панели предварительного просмотра

Чтобы удалить ненужный образец, выделите его в списке и выберите пункт **Delete** в контекстном или дополнительном меню или нажмите кнопку **Delete** (рис. 10.18), находящуюся на нижней кромке окна библиотеки. После этого Flash выдаст грозное предупреждение (рис. 10.19), говорящее о том, что все экземпляры, созданные на основе этого образца, также будут удалены, и вернуть их обратно невозможно. Если вы хотите, чтобы при удалении образцы также были удалены все его экземпляры, включите флажок **Delete Symbol Instances**, в противном случае экземпляры останутся, и вы сможете заменить их на экземпляры другого образца. После этого нажмите кнопку **Delete**, если это вас не испугало, или **Cancel**, если вы передумали удалять образец.



Рис. 10.18. Кнопка **Delete** окна библиотеки

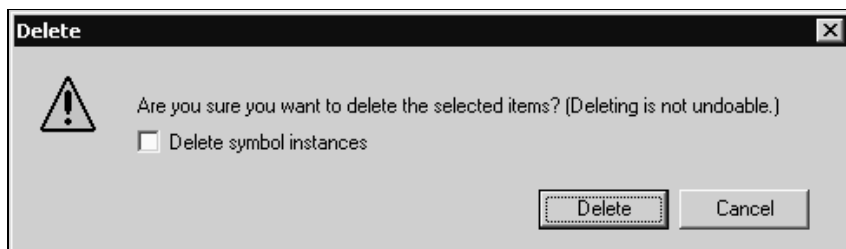


Рис. 10.19. Предупреждение об удалении образца

Flash предоставляет весьма удобную возможность выбора всех образцов, на основе которых не было создано ни одного экземпляра. Это может быть полезно, если вы хотите вычистить из документа весь мусор. Просто выберите пункт **Select Unused Items** в дополнительном меню — и все неиспользованные образцы будут выделены.

Использование папок

Для организации файлов на дисках вашего компьютера вы используете папки. Файлы "раскладываются" по папкам, исходя из какого-либо критерия. Точно таким же образом вы можете организовать образцы в вашей библиотеке — "разложив" их по папкам.

Структура папок в библиотеке может быть разной. Вы можете "разложить" по отдельным папкам файлы, относящиеся к разным этапам работы, разным типам или разным частям изображения. Для каждого случая здесь может быть свой подход. Все зависит, насколько удобно работать с той или иной иерархией папок. Но большинство художников, работающих с Flash, все-таки распределяют образцы по папкам, исходя из их типа и назначения (то же самое рекомендуется и в руководстве по Flash). Например, такая:

графика первого плана

элемент 1

элемент 2

. . .

графика второго плана

элемент 1

элемент 2

. . .

кнопки основных функций

кнопка 1

кнопка 2

. . .

кнопки дополнительных функций

кнопка 1

кнопка 2

. . .

фильмы

фильм 1

фильм 2

. . .

Использовать ли папки или обойтись без них? Это зависит от ваших предпочтений. Однако дадим один совет. Если библиотека содержит небольшое количество образцов (десяток-полтора), папки излишни — все ее содержимое видно как на ладони. Если библиотека разрастается до нескольких десятков образцов, лучше "разбросать" ее содержимое по папкам. Причем создать такую структуру папок, чтобы в каждой папке помещалось не более десяти—пятнадцати образцов. Если же получается больше, внутри каждой

папки создайте сложенные папки второго уровня и переместите образцы в них. Например, так:

графика

первый план

элемент 1

элемент 2

. . .

второй план

элемент 1

элемент 2

. . .

. . .

кнопки

основные функции

кнопка 1

кнопка 2

. . .

дополнительные функции

кнопка 1

кнопка 2

. . .

. . .

фильмы

фильм 1

фильм 2

. . .

Но не переусердствуйте.

Попробуйте работать с разными иерархиями папок. Выясните, какая из них лучше всего вам подходит. И остановитесь на ней.

С теорией покончили. Пора переходить к практике. Рассмотрим работу с папками.

Вероятно, проще всего переместить образец в новую папку, выделив его и выбрав пункт **Move to New Folder** контекстного или дополнительного меню. После этого на экране появится диалоговое окно **New Folder** (рис. 10.20). Введите в единственное поле **Name** имя новой папки и нажмите кнопку **ОК**. Если вы передумали создавать новую папку и перемещать в нее образец, нажмите кнопку **Cancel**.

К сожалению, Flash в этом случае не отличается сообразительностью. Если вы захотите переместить в ту же папку другой файл, введете в поле **Name**

имя уже существующей папки, Flash откажется ее создавать. Поэтому для "разбрасывания" файлов по уже созданным папкам нужно пользоваться другими приемами.

После создания папки вы увидите, что в списке образцов появилось нечто, аналогичное папке из левой панели Проводника Windows (рис. 10.21). Это и есть папка библиотеки Flash. В колонке **Kind** списка против имени папки стоит слово "Folder". Щелкая по значку папки, вы можете раскрыть или закрыть ее. А дважды щелкнув мышью по ее имени, можно ее переименовать, как это делалось с образцами.



Рис. 10.20. Диалоговое окно **New Folder**

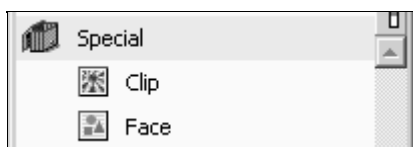


Рис. 10.21. Папка в списке образцов

Чтобы поместить образец в папку, перетащите его на значок папки или на один из образцов, уже находящихся в ней. При этом курсор мыши изменит свой вид, говоря, что в данное место вы можете "сбросить" перетаскиваемый образец. Чтобы "вынести" образец из папки, перетащите его куда-нибудь вне папки.

Чтобы создать новую папку, вы также можете нажать кнопку **New Folder** (рис. 10.22), находящуюся на нижней кромке окна библиотеки. Вы также можете выбрать пункт **New Folder** в дополнительном меню окна библиотеки. В списке образцов появится небольшое поле ввода; введите в него имя новой папки и нажмите клавишу <Enter>. Если вы нажмете клавишу <Esc>, новая папка получит имя вида "untitled folder <Номер>".



Рис. 10.22. Кнопка **New Folder** окна библиотеки

Вы уже знаете, что Flash позволяет создавать папки, находящиеся в других папках (рис. 10.23). Для этого, прежде всего, выделите любой образец, нахо-

дящийся в папке, в которой вы хотите создать новую папку. Впоследствии вы можете перетащить ее в любое другое место в иерархии папок и "наполнить" ее образцами и другими папками.

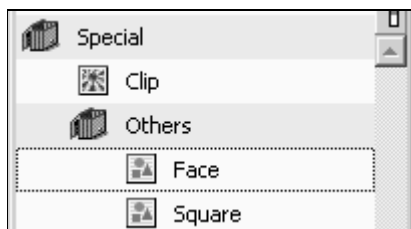


Рис. 10.23. Вложенные папки

Если вы собираетесь удалить папку, имейте в виду следующее. Дело в том, что все находящиеся в папке образцы считаются Flash ее полной собственностью. И при удалении папки они также удаляются. Поэтому, прежде чем удалять папку, вам нужно обязательно "вынести" из нее все содержащиеся в ней файлы.

Вы можете создать новый образец сразу в какой-либо папке. Для этого перед тем, как создать этот образец, выберите в нужной папке любой из существующих образцов.

Ранее мы сказали, что каждый образец в библиотеке должен иметь уникальное имя. Это не совсем верно. На самом деле, уникальные имена должны иметь все образцы, находящиеся в одной папке. Здесь опять прослеживается аналогия с файловой системой компьютера: на диске может находиться сколько угодно файлов с одинаковыми именами, если они при этом "разбросаны" по разным папкам. Это может быть очень удобно, скажем, образцы одинакового назначения, находящиеся в разных папках, могут иметь одинаковые имена.

Но что случится, если вы переместите в какую-либо папку образец, имеющий то же имя, что и уже существующий в этой папке? В этом случае возникнет так называемый *конфликт имен* образцов. Flash выдаст вам предупреждение, показанное на рис. 10.24.



Рис. 10.24. Предупреждение о конфликте имен образцов

Если вы хотите оставить оба этих образца, выберите переключатель **Don't Replace Existing Items**. В этом случае перемещенный вами образец получит имя вида "<Старое имя> сору". Если вы хотите, чтобы перемещенный образец заменил существовавший в папке, выберите переключатель **Replace Existing Items (Not Undoable)**. Учтите только, что существовавший в папке образец будет удален безвозвратно. После этого остается нажать кнопку **OK**, чтобы выполнить перенос образца, или **Cancel**, чтобы отказаться от него.

Для работы с папками Flash также предоставляет следующие пункты контекстного и дополнительного меню:

- ☐ **Expand Folder** — раскрывает выделенную в списке папку;
- ☐ **Collapse Folder** — закрывает выделенную в списке папку;
- ☐ **Expand All Folders** — раскрывает все папки в списке;
- ☐ **Collapse All Folders** — закрывает все папки в списке.

Совместное использование образцов и библиотек

Важнейшей чертой современного мира является международное разделение труда. Не вдаваясь в длинные и нудные объяснения, скажем, что компьютер, который стоит перед вами, — наглядный тому пример. Если вы когда-нибудь откроете его корпус, то поразитесь, сколько людей, живущих в самых разных странах, принимало участие в его создании. Вы найдете детали из Японии, Китая, Сингапура, США, Ирландии и даже — кто бы мог подумать! — Российской Федерации.

Вызвано это разделение труда многими причинами. Первая из них: каждый занимается тем делом, которое лучше всего знает. Остальные причины незначительны. В самом деле, зачем самим пытаться изготовить какую-то деталь (процессор, жесткий диск, корпус, провода питания), если кто-то где-то может сделать это лучше, быстрее и дешевле. А современные средства связи, в том числе, компьютерной и развитый транспорт полностью снимают проблему больших расстояний.

Так же и с программным обеспечением, и с Web-дизайном, и с компьютерной графикой. Существуют успешно работающие организации, сотрудники которых разбросаны по всему миру. В случае Web-дизайна, например, один из них делает графические элементы оформления, другой — готовит тексты, третий — отвечает за администрирование Web-сервера, принадлежащего четвертому и т. д. Пятый делает фильмы в формате Flash для размещения на сайте, а шестой занимается тем, что готовит образцы для этих фильмов. А когда подготовит, создаст разделяемую библиотеку и выложит ее в Интернет, чтобы его коллеги могли эти образцы использовать.

Давайте сначала поговорим о самом простом способе позаимствовать чужие образцы — элементарно скопировать их из чужого файла. Потом речь у нас

пойдет о так называемых обновляемых образцах. И закончим эту главу рассказом о разделяемых библиотеках — более сложном, но и более действенном способе сделать свое общим.

Копирование образцов из документа в документ

Flash предоставляет удобные средства для переноса образцов из одной библиотеки в другую. В частности, вы можете открыть несколько документов, выделить на рабочем листе одного из них нужные экземпляры, скопировать их в буфер обмена и вставить на рабочий лист другого документа. Также можно перетащить экземпляры из одного документа в другой (в этом случае вам следует позаботиться о том, чтобы окна обоих документов были видны на экране). Теперь Flash не только выполнит копирование экземпляров во второй документ, но и поместит в его библиотеку новые образцы.

Есть еще один способ переноса образцов из одного документа в другой. Выберите пункт **Open as Library** в меню **File** или нажмите комбинацию клавиш <Ctrl>+<Shift>+<O>. На экране появится стандартное диалоговое окно открытия файла Windows. Выберите файл документа Flash, из которого вы хотите позаимствовать образцы, и нажмите кнопку открытия. После этого откроется второе окно библиотеки, принадлежащей открытому документу, и вы сможете копировать из нее образцы простым перетаскиванием (рис. 10.25). Воспользовавшись пунктом **Open as Library** меню **File**, можно открыть любое количество библиотек.

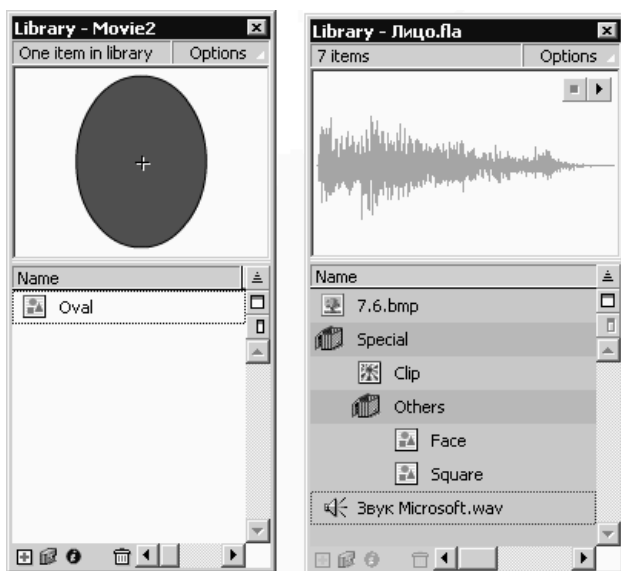


Рис. 10.25. Слева — окно библиотеки активного документа Flash, справа — окно библиотеки, открытой выбором пункта **Open as Library** меню **File**

Отметьте еще при этом, что списки образцов всех открытых таким образом библиотек закрашены серым, т. е. недоступны для изменения. Это сделано специально, чтобы вы не смогли случайно ничего там изменить. (Впрочем, это не мешает вам при необходимости открыть исходный файл библиотеки как обычный документ Flash и изменить все, что угодно.)

Обновляемые образцы

Обновляемые образцы — это образцы, которые могут быть заменены другими образцами из других библиотек, так называемыми *образцами-источниками*. При этом оригинальное имя и параметры образца сохраняются, а меняется только его содержимое. И замена эта может выполняться как вручную, так и автоматически, при каждом экспорте или публикации изображения.

Зачем это может понадобиться? Например, в следующем случае. Вы можете поместить в ваше изображение образец, который еще не закончен. То есть, художник, ответственный за этот образец, к данному моменту сделал только грубый набросок, а закончить собирается позднее. Вы помещаете этот набросок в ваше изображение, а окончательную замену (на готовый образец) выполняете позднее. Это только пример, на самом деле, возникают и другие ситуации, в которых могут помочь обновляемые образцы.

Как сделать образец обновляемым? Очень просто. Выведите на экран окно библиотеки. Далее выделите в списке нужный образец и вызовите диалоговое окно **Symbol Properties**. Для этого выберите пункт **Properties** контекстного меню списка или дополнительного меню окна (нужный образец должен быть выбран в списке). Также вы можете щелкнуть кнопку **Properties**, расположенную на нижней границе окна библиотеки (см. рис. 10.15). Нажмите кнопку **Advanced**, расположенную в нижней части этого диалогового окна. Окно **Symbol Properties** развернется в свой "полный" вид (рис. 10.26).

В правом нижнем углу этого диалогового окна видна кнопка **Browse**. Нажмите ее. На экране появится стандартное диалоговое окно открытия файла **Windows**. Выберите файл документа, в котором находится образец-источник, и нажмите кнопку открытия. После этого на экране появится диалоговое окно **Select Source Symbol**, внешне похожее на окно **Swap Symbol** (см. рис. 10.9). Выберите в его списке нужный образец-источник и нажмите кнопку **OK**.

После этого вам останется нажать кнопку **OK** и в окне **Symbol Properties**. Перед этим вы можете включить флажок **Always Update before Publishing**, если хотите, чтобы обновляемый образец обновлялся при каждом экспорте и публикации изображения.

Впоследствии вы можете изменить как сам образец-источник, так и файл документа Flash, откуда он будет взят. Чтобы изменить образец-источник, нажмите кнопку **Symbol**, выберите нужный образец в диалоговом окне **Select Source Symbol** и нажмите кнопку **OK**. Чтобы изменить файл документа,

нажмите кнопку **Browse** и выберите нужный файл в стандартном диалоговом окне открытия файла Windows.

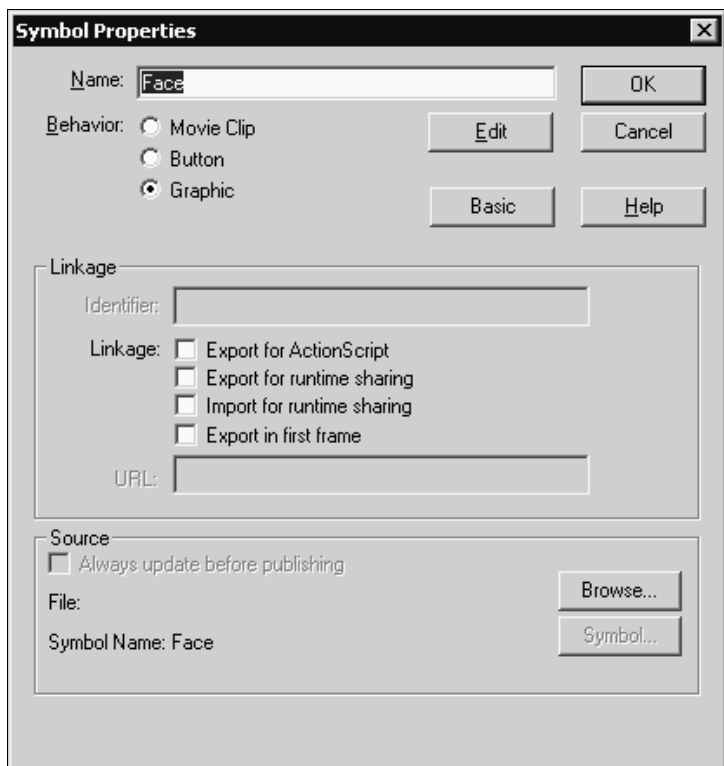


Рис. 10.26. Диалоговое окно **Symbol Properties** ("полный" вид)

Если вы хотите обновить образец вручную, выберите пункт **Update** в контекстном меню списка или дополнительном меню библиотеки. В этом случае на экране появится диалоговое окно **Update Library Items** (см. рис. 10.16). Включите против имени обновляемого образца (он будет один в списке) флажок и нажмите кнопку **Update**. После этого закройте диалоговое окно, нажав кнопку **Close**.

К несчастью, Flash никак не помечает в списке обновляемые образцы.

Разделяемые образцы

Что такое *разделяемые образцы*? Это образцы, которые могут быть использованы любым изображением Flash, возможно, даже созданным другим художником. Файлы Shockwave/Flash, содержащие разделяемые образцы, часто называют *разделяемыми библиотеками*. Такие файлы могут содержать ка-

кие-либо изображения или фильм, а могут включать в себя только библиотеку с разделяемыми образцами. Они выкладываются в Интернет, чтобы все желающие могли получить к ним доступ.

Каждый разделяемый образец должен иметь уникальное имя. Имейте в виду, что имя разделяемого образца — это не то же самое, что имя образца. Хотя они могут совпадать, но имена разделяемых образцов должны быть действительно уникальными.

Если нужно воспроизвести изображение, использующее разделяемые образцы, проигрыватель Flash загрузит нужный файл изображения Shockwave/Flash и позаимствует эти образцы из его библиотеки. Это будет выполнено только тогда, когда разделяемый образец будет действительно нужен, поэтому если изображение не использует ни одного разделяемого образца, то разделяемая библиотека загружена не будет. Конечный потребитель ничего при этом не заподозрит, если, конечно, у проигрывателя не возникнет проблем с загрузкой разделяемой библиотеки.

Выгоды такого подхода очевидны. Во-первых, пресловутое разделение труда (его преимущества были описаны выше). Во-вторых, размеры файлов Shockwave/Flash, использующих разделяемые библиотеки, значительно меньше аналогичных файлов, которые "все свое несут с собой". В-третьих, становится возможным стандартизировать некоторые элементы изображений, например, дорожные знаки или какие-либо обозначения. Такие библиотеки, содержащие "стандартные" образцы, могут использовать, например, работники одной виртуальной фирмы или просто друзья, помешанные на Flash-графике.

О самом главном недостатке разделяемых библиотек мы уже говорили. Если проигрыватель Flash почему-то не сможет загрузить такую библиотеку, изображение, использующее образцы из этой библиотеки, не сможет правильно отобразиться. Далее, размеры разделяемых библиотек, как правило, достаточно велики, а значит, разделяемые библиотеки долго загружаются. И последний недостаток: трудность контроля за такими библиотеками. В самом деле, если какой-либо из образцов такой библиотеки содержит ошибку, вам остается только ждать, пока автор ее не исправит. А такое ожидание может затянуться...

Имейте также в виду, что разделяемая библиотека всегда загружается целиком. А если пользователь имеет медленный канал доступа в Интернет, то загрузка может отнять много времени. Поэтому делайте ваши разделяемые библиотеки как можно меньше: включайте в них минимум необходимых образцов и, желательно, не создавайте в них никаких изображений или фильмов. Экономьте время пользователя.

Но хватит бродить вокруг да около! Давайте же все-таки выясним, как создаются разделяемые образцы.

Создание разделяемых образцов

Разделяемый образец создается очень просто. Вы открываете нужный документ Flash, выбираете образец в его библиотеке и задаете для него параметры экспорта. Рассмотрим этот процесс подробнее: сначала для образцов, созданных в самом Flash, а потом — для импортированных из других программ.

Итак, откройте нужный документ Flash. Выведите на экран окно библиотеки. Затем выделите в списке нужный образец (это должен быть образец, созданный в самом Flash) и вызовите диалоговое окно **Symbol Properties**. Щелкните кнопку **Advanced**, чтобы развернуть его в "полный" вид (см. рис. 10.26). И давайте рассмотрим элементы управления, расположенные в группе **Linkage**.

Чтобы превратить образец в разделяемый, включите флажок **Export for runtime sharing**. Flash автоматически подставит в поле ввода **Identifier** имя разделяемого образца, совпадающее с обычным именем, под которым вы создали этот образец. Вы можете оставить это имя или ввести другое.

Теперь о поле ввода **URL**. В нем вводится интернет-адрес, по которому будет располагаться файл разделяемой библиотеки. Если вы его точно знаете, лучше введите. В противном случае оставьте это поле пустым: вы сможете задать адрес разделяемой библиотеки потом, при включении разделяемого образца в изображение.

Задав нужные параметры, нажмите кнопку **OK**. Если вы передумали разделять этот образец, нажмите кнопку **Cancel**. Повторите эту операцию для всех образцов, которые вы хотите сделать разделяемыми.

В дальнейшем, если вы хотите сделать разделяемый образец неразделяемым (обычным), просто отключите флажок **Export for runtime sharing** в диалоговом окне **Symbol Properties**.

Рассмотрим теперь, как разделяются импортированные образцы. Выделите в списке нужный образец и выберите пункт **Linkage** контекстного меню списка или дополнительного меню окна библиотеки. На экране появится диалоговое окно **Linkage Properties** (рис. 10.27).

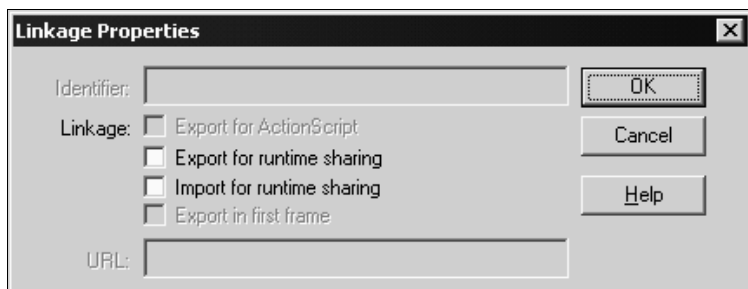


Рис. 10.27. Диалоговое окно **Linkage Properties**

Как видите, все элементы в этом окне уже вам знакомы. Включите флажок **Export for runtime sharing**, укажите нужные значения в полях ввода **Identifier** и **URL** и нажмите кнопку **OK**.

Учтите, что в списке образцов разделяемые и неразделяемые образцы показывались почти одинаково. Единственное отличие: в колонке **Linkage** у неразделяемого образца стоит прочерк, а у разделяемого — слово "Export: <Имя разделяемого образца>". Поэтому, чтобы выяснить, что вы экспортировали, вам придется часто прокручивать список по горизонтали или расширять окно библиотеки.

Сохраните документ Flash. Все, файл разделяемой библиотеки готов. Остается его экспортировать в формат Shockwave/Flash (об этом см. *главу 11*).

Дальнейшие действия по созданию разделяемой библиотеки уже не связаны с Flash. Они включают в себя, прежде всего, размещение разделяемой библиотеки (файла в формате Shockwave/Flash) на каком-либо Web-сервере. Если же вы собираетесь распространять свои Flash-творения другим способом, например, на компакт-дисках, вам нужно будет записать файл разделяемой библиотеки на эти диски. Одним словом, вы должны сделать свою разделяемую библиотеку доступной.

Кроме того, вам придется каким-либо способом распространить сведения о вашей разделяемой библиотеке среди коллег, иначе говоря, заняться рекламой. Для этого хороши все средства: публикация Web-страницы, электронная почта, личное общение и т. п.

Использование разделяемых образцов

Чтобы использовать разделяемый образец, вам понадобится файл разделяемой библиотеки в формате Shockwave/Flash, содержащей этот образец. Этот файл, как правило, выкладывается на Web-сервере или делается доступным для использования в изображениях Flash иным способом, например, для распространения на компакт-дисках. Причем, вам не обязательно иметь доступ к нему при создании изображения с использованием разделяемых образцов. Хотя, конечно, это желательно.

Очень желательно также получить доступ к исходному FLA-файлу разделяемой библиотеки, образцы которой вы хотите использовать. Конечно, не всегда этот файл доступен, но иметь его при себе не помешает. В этом случае вы сможете извлекать разделяемые образцы прямо из него, таким образом, доступ к SWF-файлу разделяемой библиотеки вам уже не нужен. Более того, вам даже не нужно знать ни имен разделяемых элементов, ни интернет-адреса библиотеки.

Предположим этаким идеальный вариант развития событий. Вы раздобыли исходный файл разделяемой библиотеки и точно знаете, где находится сама библиотека. Как теперь "позаимствовать" из нее нужный образец и поместить его в новый документ Flash?

Очень просто.

Выберите пункт **Open as Library** в меню **File** или нажмите комбинацию клавиш <Ctrl>+<Shift>+<O>. На экране появится стандартное диалоговое окно открытия файла Windows. Выберите исходный файл нужной разделяемой библиотеки и нажмите кнопку открытия. На экране появится еще одно окно библиотеки, содержащее все образцы открытой разделяемой библиотеки (см. рис. 10.25). Расположите оба эти окна на экране так, чтобы они не перекрывали друг друга.

Теперь перетащите нужный образец из окна разделяемой библиотеки в окно библиотеки нового документа. Вы также можете перетащить его прямо на рабочий лист, в этом случае Flash сам поместит его в библиотеку. Новый образец будет помечен как импортированный: в колонке **Linkage** списка против его имени будет стоять слово "Import <Имя разделяемого образца>".

Вы можете затем добавлять на рабочий лист импортированные из разделяемой библиотеки образцы, как если бы они были созданы в самом этом документе. При этом описания данных образцов в новый документ помещаться не будут, а будут загружаться из разделяемой библиотеки при просмотре готового изображения.

Если же вам не удалось раздобыть исходный файл разделяемой библиотеки, не беда. Создайте новый "пустой", т. е. не имеющий содержимого образец. Выделите его в списке образцов и вызовите на экран диалоговое окно **Symbol Properties** (для образцов, созданных в самом Flash) или **Linkage Properties** (для импортированных образцов). Включите флажок **Import for Runtime Sharing**, если же он недоступен, то отключите все остальные флажки в группе **Linkage**. Теперь осталось задать нужные значения в полях ввода **Identifier** и **URL** и нажать кнопку **OK**.

Разрешение конфликта имен

Часто случается так, что вы копируете в библиотеку образец с именем, которое совпадает с именем уже существующего образца. Неважно, каким образом при этом вы скопировали новый образец: вместе с переносом его описания или путем использования импортированного образца. Как вы помните, такую ситуацию называют конфликтом имен.

Flash требует, чтобы все образцы, находящиеся в одной папке, независимо от их типа, имели уникальные имена. По имени Flash определяет, на основании какого образца создан тот или иной экземпляр. Если же одно и то же имя будут иметь два образца, он не сможет выбрать из них нужный. Поэтому конфликты имен нужно разрешать.

Делается это следующим образом.

Допустим, вы скопировали в библиотеку образец, чье имя конфликтует с именем уже существующего в той же папке образца. И предположим, что эти образцы имеют один тип, например, графический. В этом случае на

экране появится окно предупреждения, показанное на рис. 10.24. И переключатели, расположенные в этом окне, ведут себя точно так же.

Если вы хотите оставить оба этих образца, выберите переключатель **Don't Replace Existing Items**, в этом случае перемещенный вами образец получит имя вида "<Старое имя> сору". Впоследствии вы сможете переименовать эти образцы или удалить ненужный.

Если же вы хотите, чтобы перемещенный образец заменил существовавший в папке, выберите переключатель **Replace Existing Items (Not Undoable)**. Учтите только, что существовавший в папке образец будет удален безвозвратно.

Чтобы выполнить операцию копирования образца, после выбора нужного переключателя нажмите кнопку **ОК**. Если вы хотите прервать операцию копирования образцов в библиотеку, то сразу нажмите кнопку **Cancel**. В этом случае операция копирования будет тотчас прервана, и никаких образцов в библиотеку скопировано больше не будет.

Образцы-шрифты

В главе 7, говоря о работе с текстом в среде Flash, мы упомянули о проблеме, возникающей при экспорте готового изображения в формат Shockwave/Flash. В этом случае в результирующий SWF-файл помещается описание всех шрифтов, использованных в этом изображении, что может вызвать значительное увеличение размера этого файла. В качестве решения этой проблемы предлагалось три пути: уменьшение количества использованных в изображении шрифтов до необходимого минимума, использование шрифтов-псевдонимов или преобразование текста в графику.

Здесь предлагается четвертый вариант решения этой проблемы: создание *разделяемого образца-шрифта* в разделяемой библиотеке. Это значит, что вы можете поместить в разделяемую библиотеку любой шрифт, который вам нужен. Если впоследствии вы используете такой шрифт в любом другом изображении Flash, проигрыватель Flash просто загрузит нужную разделяемую библиотеку и извлечет из нее образец-шрифт.

Разделяемый образец-шрифт создается в два этапа. На первом этапе создается обычный, неразделяемый образец-шрифт. На втором этапе он делается разделяемым.

Внимание!

Создание обычных образцов-шрифтов лишено смысла. Шрифт все равно помещается в результирующий SWF-файл при экспорте готового изображения. Имеет смысл создавать только разделяемые образцы-шрифты, которые могут быть использованы в других изображениях.

Давайте создадим образец-шрифт. Для этого выберем в дополнительном меню окна библиотеки пункт **New Font**. На экране появится диалоговое окно **Font Symbol Properties** (рис. 10.28).



Рис. 10.28. Диалоговое окно **Font Symbol Properties**

Введите в поле **Name** имя создаваемого образца-шрифта. Это имя может совпадать с оригинальным именем шрифта, уберите только из него пробелы. В раскрывающемся списке **Font** выберите шрифт, на основе которого создается образец. Включите флажки **Bold** и **Italic**, если хотите создать соответственно полужирный шрифт и курсив. И нажмите кнопку **OK**. Если же вы передумали создавать образец-шрифт, нажмите кнопку **Cancel**.

После создания образец-шрифт появится в списке образцов библиотеки. К сожалению, Flash не предоставляет возможности для его просмотра. Чтобы увидеть, как выглядит этот шрифт, вам придется создать текстовый блок, ввести в нем какой-либо текст и отформатировать этим шрифтом.

Чтобы преобразовать обычный образец в разделяемый, выделите в списке только что созданный образец-шрифт и выберите пункт **Linkage** контекстного меню списка или дополнительного меню окна библиотеки. На экране появится диалоговое окно **Linkage Properties** (см. рис. 10.27). Задайте в нем нужные параметры разделения и не забудьте нажать кнопку **OK**.

Чтобы использовать в другом изображении разделяемый шрифт, поступите так же, как и в случае обычного образца. То есть, откройте исходный FLA-файл, воспользовавшись пунктом **Open as Library** в меню **File**, и перетащите образец-шрифт в окно библиотеки нового изображения.

Зачастую Flash ведет себя в этом случае очень странно. А именно, вместо того, чтобы использовать разделяемый образец-шрифт, он просто копирует его целиком в библиотеку нового документа. Тогда вам придется вызвать диалоговое окно **Linkage Properties** и проверить, включен ли в нем флажок **Import for Runtime Sharing**. Если же он отключен, включите его, но прежде отключите флажок **Export for Runtime Sharing**. После этого укажите интернет-адрес разделяемой библиотеки в поле ввода **URL** и нажмите кнопку **OK**.

Далее вы можете, как обычно, форматировать текст с использованием этого шрифта, просто выбрав его в списке шрифтов редактора свойств (см. рис. 7.5). Использованный образец-шрифт будет помечен звездочкой (рис. 10.29).

Созданный образец-шрифт может быть переименован, удален или перемещен в другую папку. Также вы можете изменять его параметры в окне **Font Symbol Properties**, выбрав пункт **Properties** в контекстном или дополнитель-

ном меню, и параметры разделения в окне **Linkage Properties**, выбрав пункт **Linkage**.

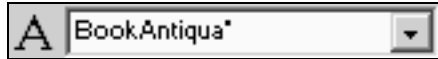


Рис. 10.29. Образец-шрифт помечен в списке шрифтов звездочкой

Превращение разделяемого образца в обычный

Может получиться так, что вы захотите превратить использованный разделяемый образец обратно в обычный, описание которого включается в библиотеку использующего его документа. Сейчас мы опишем, как это делается.

Вызовите на экран диалоговое окно **Symbol Properties** или **Linkage Properties** — в этом случае не имеет значения, какое окно вы вызовете. Отключите переключатель **Import for Runtime Sharing**. И нажмите кнопку **OK**, если вы уверены в себе. В противном случае нажмите кнопку **Cancel**.

Библиотеки общего использования

В составе Flash поставляется три *библиотеки общего использования*, содержащие образцы, которые вы можете свободно применять в своих изображениях и фильмах. Чтобы загрузить такую библиотеку, выберите соответствующий ей пункт в подменю **Common Libraries** меню **Window**. Все доступные пункты этого подменю перечислены в табл. 10.1.

Таблица 10.1. Библиотеки общего использования, поставляемые в составе Flash

Название	Описание
Buttons	Разнообразные кнопки (о кнопках см. главу 20)
Learning Interactions	Различные примеры реализации элементов управления средствами Flash (об элементах управления см. главу 23)
Sounds	Множество полезных звуков

Библиотеки общего использования — это обычные документы Flash (файлы с расширением fla). Если вы работаете в Windows 95/98/Me, то можете найти их в подкаталоге Application Data/Macromedia/Flash MX/Configuration/Libraries, расположенном в каталоге Windows. Если же вы предпочитаете Windows NT/2000/XP, откройте подкаталог Application Data/Macromedia/Flash MX/Configuration/Libraries, расположенный в каталоге вашего пользовательского профиля. Если вы хотите добавить во Flash свои библиотеки, просто поместите нужные файлы в этот подкаталог.

Проводник Flash

Наши изображения все усложняются и усложняются. Теперь, по прошествии семи глав, они содержат не только векторную, но и импортированную растровую графику, и текстовые блоки, и образцы, и экземпляры этих образцов. Как разобраться во всем этом богатстве?

Специально для такого случая Flash предлагает вам весьма мощное средство, называемое *Проводником* (в терминологии Flash — *Movie Explorer*). Аналогично Проводнику Windows, он позволяет вам просматривать структуру вашего документа, искать и находить различные графические элементы и выполнять над ними необходимые действия. Если ваше изображение достигло такого уровня сложности, когда вы не можете разобраться в нем без посторонней помощи, — смело обращайтесь к Проводнику!

Вызвать на экран Проводник можно несколькими способами. Самый быстрый — нажать комбинацию клавиш <Alt>+<F3>. Также вы можете выбрать пункт **Movie Explorer** меню **Window** или одноименный пункт контекстного меню листа. Окно Проводника показано на рис. 10.30.

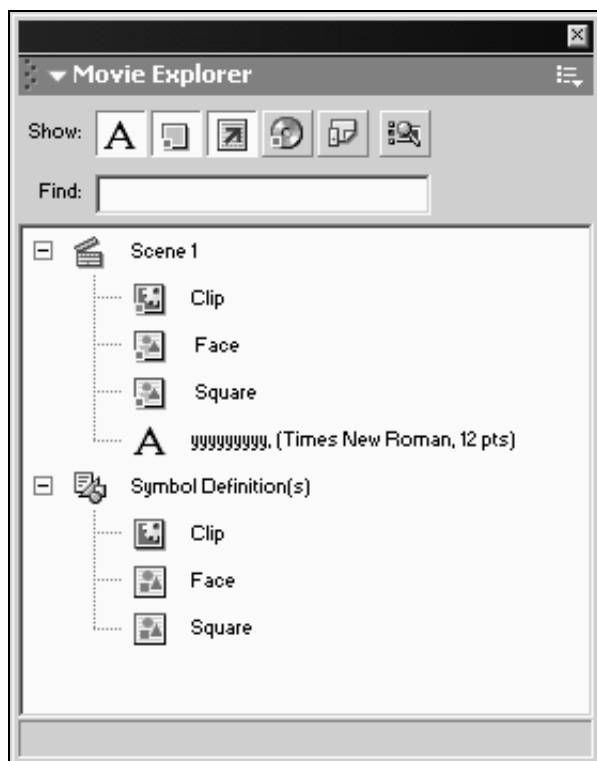


Рис. 10.30. Окно Проводника Flash

Большую часть окна Проводника занимает обширный иерархический список всех имеющихся в составе изображения Flash элементов. (На самом деле, не всех, но об этом чуть ниже.) Как только вы добавляете или удаляете что-то на рабочем листе, содержимое этого списка изменяется.

Вы можете разворачивать и сворачивать ветви этого списка, щелкая "узелки", расположенные левее значков папок, или выбирая соответственно пункты **Expand Branch** или **Collapse Branch** в контекстном или дополнительном меню. А пункт **Collapse Others** сворачивает все ветви, кроме выбранной в списке.

Вы можете выбирать любые пункты этого списка, просто щелкая по ним мышью, как и в списках Проводника Windows. Вы также можете выбирать одновременно несколько пунктов, удерживая нажатой клавишу <Ctrl>. Правда, вряд ли вам это пригодится.

Графические элементы, составляющие изображение, отображаются в ветви **Scene 1** (номер может быть другим). Если вы выберете любой из этих элементов, Flash сделает его выбранным на рабочем листе. Если выбранный элемент все равно не виден, выберите пункт **Goto Location** контекстного или дополнительного меню. А пункт **Goto Symbol Definition** заставит Flash выбрать в списке Проводника образец, на основе которого создан этот экземпляр.

Образцы, внесенные в библиотеку, отображаются в ветви **Symbol Definitions**. Если вы выберете пункт **Find in Library** контекстного или дополнительного меню, Flash откроет окно библиотеки и выделит в списке выбранный вами в Проводнике образец. Пункт **Select Symbol Instances** заставит Flash выбрать в Проводнике и на рабочем листе все экземпляры этого образца.

Внимание!

К сожалению, в окне Проводника почему-то не отображаются образцы, созданные из импортированных файлов, и образцы-шрифты. Однако экземпляры этих образцов в списке Проводника отображаются.

Остальные пункты, доступные в контекстном и дополнительном меню окна Проводника, перечислены ниже.

Пункт **Rename** позволит вам переименовать выбранный в списке элемент. (Правда, в настоящее время это актуально лишь для образцов — давать имена экземплярам мы будем значительно позднее.) Если выбран текстовый блок, этот пункт позволяет изменить его содержимое. После выбора этого пункта вместо имени элемента в списке появится небольшое поле ввода, в котором будет подставлено старое имя. Введите новое имя и нажмите клавишу <Enter> для его сохранения или <Esc> — для отказа от изменения имени.

Пункты **Edit**, **Edit in Place** и **Edit in New Window** позволят вам изменить выбранный образец: первый пункт — в обычном режиме редактирования, второй — "на месте", третий — в отдельном окне. Чтобы изменить выбранный образец "на месте", вы также можете дважды щелкнуть по нему в списке.

Если выбран экземпляр или текстовый блок, в меню становятся доступными пункты **Cut**, **Copy**, **Paste** и **Clear**. Первые три пункта предназначены для работы с буфером обмена Windows: вырезания, копирования и вставки соответствующего графического элемента. Последний пункт позволяет удалить выделенный в списке элемент.

Пункт **Copy Text to Clipboard** выполняет очень странную операцию. Он копирует в буфер обмена текстовое представление списка Проводника. В таком тексте имитируется вложенность пунктов списка в ветви, это достигается отступами разной длины. Зачем и кому это может понадобиться, непонятно. Аналогично, пункт **Print** позволяет распечатать список Проводника.

Если ваше изображение настолько сложно, что список в окне Проводника оказывается слишком большим, чтобы в нем разобраться, вы можете воспользоваться весьма мощными функциями поиска. Если вы знаете имя экземпляра, который хотите найти, введите его в поле ввода **Find**, расположенном над списком (см. рис. 10.30). Flash выведет в списке только те экземпляры, чье имя содержит введенную вами последовательность символов. К сожалению, образцы при этом в любом случае показаны не будут.

Если вы хотите просмотреть в списке Проводника только образцы, включите пункт-выключатель **Show Symbol Definitions** в контекстном или дополнительном меню, а пункт-выключатель **Show Movie Elements** выключите. Если вы, наоборот, хотите просмотреть только экземпляры и другие элементы изображения, пункт **Show Movie Elements** включите, а **Show Symbol Definitions** выключите. Если же вы хотите просмотреть и образцы, и экземпляры, включите оба этих пункта.

В верхней части окна расположены кнопки-выключатели, позволяющие более точно выбрать графические элементы, которые будут выводиться в списке Проводника. Перечислим их слева направо:

- ☐ показ текстовых блоков;
- ☐ показ образцов, созданных в самом Flash, и их экземпляров;
- ☐ показ сценариев, написанных на языке программирования ActionScript (о программировании см. *часть 4*);
- ☐ показ экземпляров импортированных образцов (растровых изображений, фильмов и звуков);
- ☐ показ слоев и кадров (о кадрах см. *главу 13*, а о слоях — *главу 15*).

Последняя — шестая кнопка — выводит на экран диалоговое окно **Movie Explorer Settings**, где параметры вывода элементов задаются более точно. Это окно показано на рис. 10.31.

Набор флажков **Show** позволяет задать объекты, которые нужно показывать в списке Проводника:

- ☐ **Text** — текстовые блоки;

- ☐ **Buttons** — кнопки (т. е. экземпляры образцов-кнопок, сами образцы-кнопки выводятся всегда);
- ☐ **Movie Clips** — клипы (т. е. экземпляры образцов-клипов, сами образцы-клипы выводятся всегда);
- ☐ **Video** — импортированные клипы (т. е. экземпляры образцов-импортированных клипов, сами образцы-импортированные клипы не выводятся никогда);
- ☐ **ActionScripts** — сценарии на языке программирования **ActionScript**;
- ☐ **Bitmaps** — растровые изображения (сами образцы-растровые изображения не выводятся никогда);
- ☐ **Graphics** — обычные графические экземпляры (сами графические образцы выводятся всегда);
- ☐ **Sounds** — звуки, точнее, экземпляры звуковых образцов (сами образцы-звуки не выводятся никогда, о работе со звуком см. главу 17);
- ☐ **Layers** — слои;
- ☐ **Frames** — кадры.

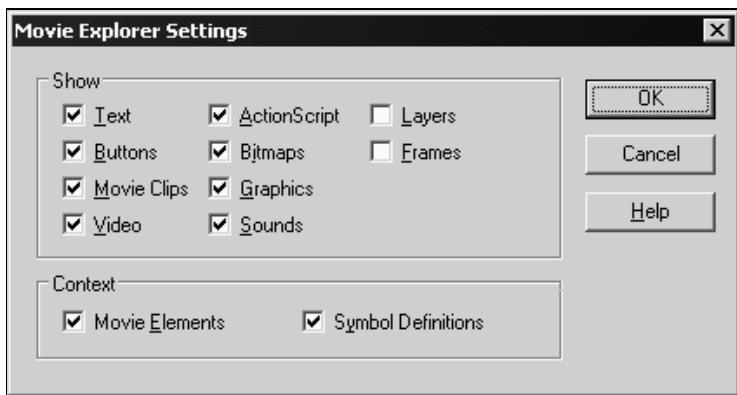
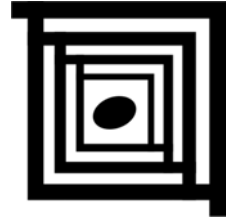


Рис. 10.31. Диалоговое окно **Movie Explorer Settings**

Набор флажков **Context** позволяет выбрать показ образцов (флажок **Symbol Definitions**) или элементов изображения (флажок **Movie Elements**).

Вам, скорее всего, придется поэкспериментировать, прежде чем вы достигнете нужного результата. К сожалению, Flash зачастую ведет себя очень странно при задании некоторых параметров в этом окне. Вероятно, это ошибки в программе.

Задав нужные параметры отображения списка, нажмите кнопку **ОК**, чтобы немедленно их применить. Если вы не хотите их применять, нажмите кнопку **Cancel**.



Глава 11

Публикация и экспорт статичной графики

Вот и закончен наш рисунок

Добавлены последние штрихи, исправлены последние ошибки, выполнено последнее сохранение, создана резервная копия. Что дальше?

А дальше — *публикация*, сохранение изображения Flash в одном из популярных форматов для распространения. И, конечно, само распространение. Но о распространении мы говорить не будем — это уже проблемы не Flash. Лучше поговорим о публикации, о завершающем этапе создания в среде Flash любого графического изображения.

При публикации Flash не просто сохраняет изображение в другом формате. Он также выполняет весьма сильную оптимизацию, удаляя ненужные цвета, "обрезая" лишнее пустое пространство, сжимая массив точек согласно выбранному вами алгоритму. Благодаря этому графический файл, подготовленный для распространения в Интернете, по электронной почте или иным способом, имеет минимальный размер.

Кроме публикации, вы можете выполнить экспорт изображения. *Экспорт* отличается от публикации тем, что при создании изображения Flash не проводит никакой оптимизации. Поэтому экспортированные файлы плохо подходят для распространения графики, но зато прекрасно редактируются в других графических программах.

Давайте рассмотрим публикацию и экспорт изображений Flash.

Публикация изображения

Итак, работа над изображением закончена. Теперь нам нужно сохранить ее в одном из широко распространенных форматов и передать потребителю. Причем сохранить так, чтобы оно выглядело как можно лучше, и "весило" как можно меньше. Иначе говоря, мы займемся публикацией изображения.

Выбор формата публикации

Но, прежде всего, давайте выясним, в каком формате мы будем распространять наше творение. Правильный выбор формата публикации в данном случае решает многое. В самом деле, если вы выберете для распространения изображения какой-нибудь малоупотребительный формат, мало кто сможет его просмотреть. Поэтому нужно ориентироваться на самые популярные форматы графики.

Практически всегда неподвижные, статические изображения сохраняются для дальнейшего распространения в двух форматах: GIF и JPEG. В последнее время, в связи с событиями вокруг формата GIF, постепенно, хотя и очень медленно, набирает популярность формат PNG. (Иногда используется также формат BMP и некоторые другие, но довольно редко.)

Вы, конечно же, заметили, что все три этих формата — растровые. Растровая графика традиционно лидирует, когда дело доходит до распространения изображений. Это происходит потому, что растровая графика требует существенно меньше системных ресурсов для вывода ее на экран. Конечно, она проигрывает векторной в возможности обработки, но скажите честно, вы часто занимаетесь тем, что вращаете, искажаете и перекрашиваете полученные из Сети картины в стиле "фэнтези"? Вы их просто смотрите, не так ли?

Векторная графика, в отличие от растровой, используется практически только при создании изображений. Точнее, использовалась, пока не появился пакет Macromedia Flash и, вместе с ним, одноименный формат векторной графики, предназначенный именно для ее распространения. Можно сказать, что Flash осуществил прорыв, принес векторную графику конечному потребителю графических изображений. Но даже графика Flash имеет достаточно узкую область применения, остальные области прочно удерживает за собой растровая графика.

Так какой же формат выбрать? На каком из них остановиться?

Формат GIF, как вы уже знаете из *главы 4*, прекрасно подходит для сохранения и распространения штриховых рисунков с небольшим количеством цветов, таких, как схемы, карты, карандашные рисунки, элементы оформления Web-страниц и т. п. При сохранении графики в формате GIF выполняется сжатие массива данных по алгоритму, исключающему потери данных. Так что файлы GIF получаются весьма компактными.

Формат PNG — прекрасная альтернатива GIF. Он особенно хорош тем, что разработан группой независимых исследователей и абсолютно бесплатен. GIF же — увы! — стоит денег. Фирма UniSys, владеющая патентом на алгоритм сжатия, на редкость нехотела вспомнить об этом. Так что если вы озабочены возможным переходом формата GIF в разряд платных продуктов, сохраняйте свою графику в файлах PNG. Тем более, что PNG предоставляет несравнимо больше "наворотов", чем старенький GIF.

Что касается формата JPEG, то главный его недостаток — алгоритм сжатия, вызывающий потерю данных. Этот формат хорошо использовать для распространения полноцветных полутоновых изображений: картин, фотографий, графики с большим количеством градиентов и т. п. При сохранении в формате JPEG штриховой графики возможны сильные потери качества, да и файлы могут получиться больше, чем их GIF-аналоги. По сравнению с тем же BMP, JPEG обеспечивает достаточную компактность, а при создании JPEG-файлов можно выбрать необходимое качество изображения.

Итак, наш выбор: GIF, PNG и JPEG. Точнее, один из этих форматов.

Задание формата публикации

Итак, мы выбрали формат, в котором будем распространять наши изображения. Сразу же сообщим Flash о своем выборе!

Выберем пункт **Publish Settings** в меню **File** или нажмем комбинацию клавиш <Ctrl>+<Shift>+<F12>. На экране появится диалоговое окно **Publish Settings**, показанное на рис. 11.1. Если надо, переключитесь на вкладку **Formats**. В ней вы сможете выбрать формат экспорта и задать параметры публикуемого изображения.

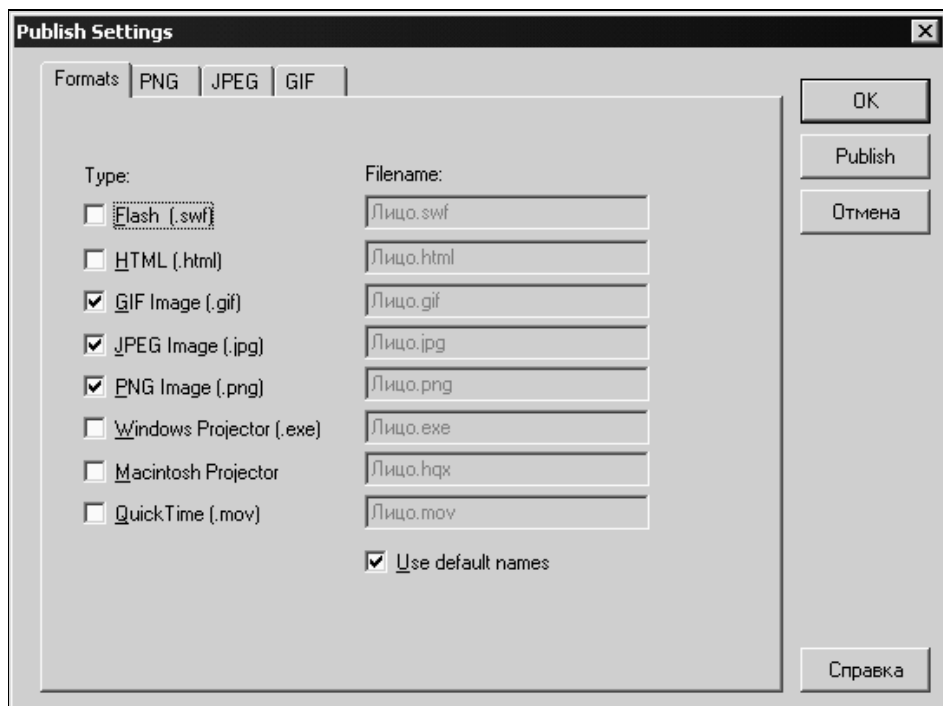


Рис. 11.1. Диалоговое окно **Publish Settings** (вкладка **Formats**)

Как вы уже поняли, формат публикации изображения задается с помощью набора флажков **Type** в левой части этого окна. Флажков довольно много, поэтому мы не будем рассматривать их все. Ограничимся тремя: **GIF**, **JPEG** и **PNG**. Понятно, что они означают.

Итак, чтобы сохранить ваше изображение в нужном формате, вы просто включаете соответствующий флажок. Если вам нужно сохранить изображение сразу в нескольких форматах, создав несколько разных файлов, вы можете включить несколько флажков — Flash это позволяет.

В правой части диалогового окна находится группа полей ввода **Filename** — по одному полю на каждый флажок и, следовательно, на каждый формат. В них вы можете задать имя результирующего файла. По умолчанию Flash берет имя изначального документа Flash и добавляет к нему соответствующее формату расширение (gif, jpg и png). Результирующие файлы сохраняются в той же папке, где лежит исходный документ. Как правило, эти установки изменять не требуется. Если вы все же хотите изменить имена результирующих файлов, отключите флажок **Use default names** в нижней части диалогового окна и введите новые имена.

Само сохранение осуществляется после нажатия кнопки **Publish**. Во время экспорта Flash показывает индикатор процесса его выполнения.

Опубликовав изображение, нажмите кнопку **ОК**, чтобы сохранить заданные вами установки для использования в дальнейшем. Если вы не хотите сохранять их, нажмите кнопку **Cancel**. Однако мы все-таки настоятельно рекомендуем нажать **ОК**. Сохранив параметры публикации вашего изображения, вы в дальнейшем сможете публиковать его, выбрав пункт **Publish** в меню **File** или нажав комбинацию клавиш <Shift>+<F12>.

Настройки форматов публикации

Кроме собственно задания форматов публикации и имен результирующих файлов, Flash позволяет вам точно настроить параметры того или иного формата. Например, вы можете задать размеры изображения, количество цветов и выбрать алгоритм его сжатия. Рассмотрим эти настройки во всех подробностях.

GIF

Если на вкладке **Formats** диалогового окна **Publish Settings** включен флажок **GIF**, становится доступной вкладка **GIF**. Ее содержимое показано на рис. 11.2. Рассмотрим его, пропустив элементы управления, задающие настройки анимации.

Группа **Dimensions** из полей ввода **Width** и **Height** служит для задания соответственно ширины и высоты результирующего изображения. Если включен флажок **Match Movie**, эти размеры совпадают с размерами рабочего листа.

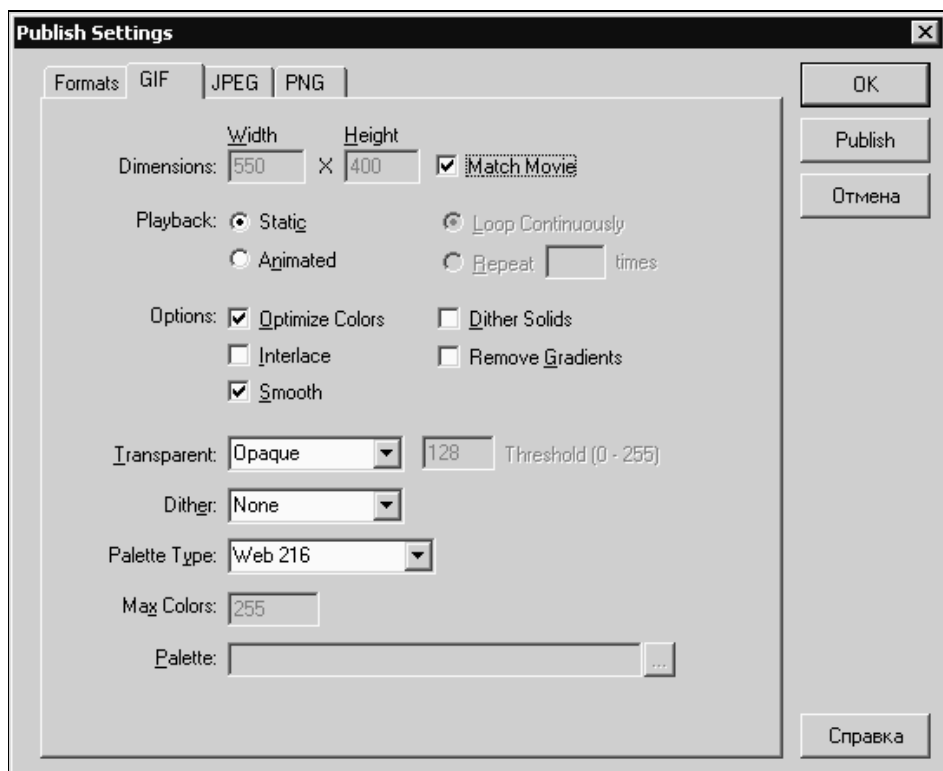


Рис. 11.2. Диалоговое окно **Publish Settings** (вкладка **GIF**)

Проверьте, включен ли в группе **Playback** переключатель **Static**. Если нет, включите его. Этот переключатель задает создание обычного, статичного изображения. Если же включен переключатель **Animated**, Flash создаст анимированный GIF.

Группа флажков **Options** управляет параметрами оптимизации изображения, которую проводит при его сохранении Flash. Опишем их подробно.

Флажок **Optimize Colors** включает или отключает оптимизацию палитры цветов изображения. Если она включена, Flash при сохранении удаляет неиспользуемые цвета. Это позволяет уменьшить размер результирующего файла на 1000—1500 байт, однако требует для хранения изображения больше оперативной памяти. Поскольку современные компьютеры имеют большой объем памяти, а каналы доступа в Интернет все еще медленны, лучше держать этот флажок включенным.

Флажок **Interlace**, будучи включенным, вызывает создание изображения GIF с чередованием. Такой файл отображается Web-обозревателем по частям, постепенно "проявляясь" в процессе загрузки. Если вы создаете изображение для публикации в Интернете, лучше включите этот флажок.

Флажок **Smooth** включает или отключает сглаживание контуров результирующего изображения. Сглаженные изображения выглядят лучше, но в некоторых случаях при этом возможно ухудшение качества, также увеличивается размер файла. "Поиграйтесь" с этой настройкой, чтобы выбрать лучшее качество изображения.

Флажок **Dither Solids** включает или отключает использование составных цветов. При создании изображения GIF может возникнуть такая ситуация, когда для сохранения цвета какого-либо фрагмента изображения не хватит позиции в палитре. В этом случае Flash может либо подобрать цвет из уже существующих, что может вызвать искажение цветов, либо составить нужный цвет, смешав пиксели разных цветов. Такие цвета, составленные из пикселей других цветов, называются *составными*. Использование составных цветов передает цвета изображения более-менее точно, но вызывает увеличение размера файла. Чтобы выяснить, идет ли использование составных цветов на пользу вашему изображению, вам также придется поэкспериментировать.

Флажок **Remove Gradients** включает или отключает удаление градиентов в результирующем изображении. Градиенты в изображении формата GIF, как правило, выглядят очень грубо, к тому же увеличивают размер файла. Если вы включите флажок **Remove Gradients**, Flash преобразует градиентные заливки в сплошные, для закраски которых будет использован ключевой цвет градиента. Включать или не включать этот флажок, решать вам.

Раскрывающийся список **Transparent** позволяет сделать цвет фона рабочего листа прозрачным. Этот список имеет три пункта:

- ☐ **Opaque** — непрозрачный фон;
- ☐ **Transparent** — прозрачный фон;
- ☐ **Alpha** — полупрозрачный фон.

Если выбран третий пункт, становится доступным поле ввода, расположенное правее списка **Transparent**. В нем задается степень прозрачности от 0 (полная прозрачность) до 255 (полная непрозрачность).

Раскрывающийся список **Dither** позволяет выбрать способ создания составных цветов для градиентных заливок. Он имеет три пункта:

- ☐ **None** — составные цвета вообще не используются, для отображения градиентов берутся наиболее близкие цвета из палитры;
- ☐ **Ordered** — создаются составные цвета приемлемого качества, размер результирующего файла слегка увеличивается;
- ☐ **Diffusion** — создаются высококачественные составные цвета, размер результирующего файла увеличивается сильнее.

Пункт **Diffusion** работает только тогда, когда для файла задана безопасная палитра цветов Web.

Раскрывающийся список **Palette Type** служит для выбора цветовой палитры, которая будет создана для GIF-файла. В нем доступны четыре пункта, которые мы рассмотрим ниже.

Пункт **Web 216** включает использование безопасной палитры цветов Web. В этом случае достигаются очень малый размер файла и высокая скорость его обработки. Однако возможно искажение цветов графики за счет того, что цвета безопасной палитры могут не полностью совпадать с цветами, используемыми в изображении. Выберите этот пункт, если изображение предназначается для систем, отображающих максимум 256 цветов.

Пункт **Adaptive** имеет в своем составе *адаптивную палитру цветов*. Такая палитра содержит только цвета, реально используемые в изображении. Если выбран этот пункт списка, становится доступно поле ввода **Max Colors**, где задается максимальное количество доступных в палитре цветов. При работе с адаптивной палитрой не происходит никакого искажения цветов, однако размер файла и время обработки могут вырасти. Выберите этот пункт, если изображение будет отображаться на компьютерах с современными мощными видеоподсистемами, выводящими одновременно тысячи и миллионы цветов.

Пункт **Web Snap Adaptive** аналогичен пункту **Adaptive**, за тем исключением, что при любой возможности будут использоваться цвета из безопасной палитры Web. Благодаря этому размер файла уменьшится, но, возможно, появятся искажения цветов. Этот пункт — компромисс между **Web 216** и **Adaptive**.

Пункт **Custom** позволяет вам выбрать палитру цветов самому. После выбора этого пункта становится доступно поле ввода **Palette**, где вы сможете ввести имя нужного файла. Поддерживается импорт из файлов палитр АСТ и различных графических файлов.

JPEG

Если на вкладке **Formats** диалогового окна **Publish Settings** включен флажок **JPEG**, становится доступной вкладка **JPEG**. Ее содержимое показано на рис. 11.3. Рассмотрим расположенные на ней немногочисленные элементы управления.

Группа **Dimensions** из полей ввода **Width** и **Height** служит для задания соответственно ширины и высоты результирующего изображения. Если включен флажок **Match Movie**, эти размеры совпадают с размерами рабочего листа.

Регулятор **Quality** позволяет задать качество результирующего изображения от 0 до 100. Чем больше это значение, лучше выглядит изображение и тем больше получается его файл. А в поле ввода, расположенном правее регулятора, можно ввести значение качества вручную.

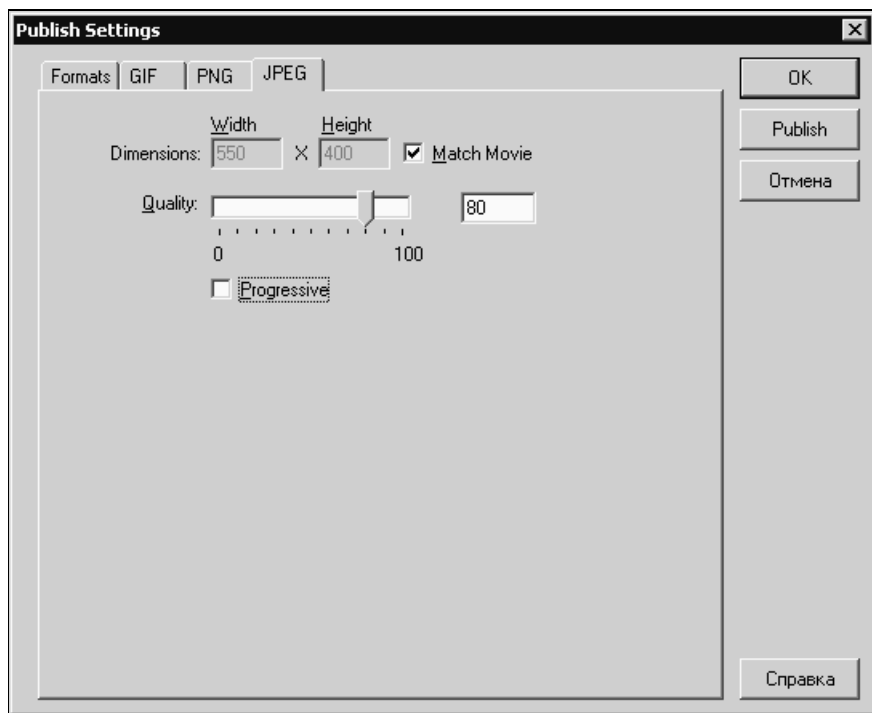


Рис. 11.3. Диалоговое окно **Publish Settings** (вкладка **JPEG**)

Флажок **Progressive**, будучи включенным, вызывает создание изображения JPEG с чередованием. Такой файл отображается Web-обозревателем, постепенно "проявляясь" в процессе загрузки. Если вы создаете изображение для публикации в Интернете, лучше включите этот флажок.

PNG

Если на вкладке **Formats** диалогового окна **Publish Settings** включен флажок **PNG**, становится доступной вкладка **PNG**. Ее содержимое показано на рис. 11.4. Рассмотрим расположенные на ней элементы управления.

Группа **Dimensions** из полей ввода **Width** и **Height** служит для задания соответственно ширины и высоты результирующего изображения. Если включен флажок **Match Movie**, эти размеры совпадают с размерами рабочего листа.

Раскрывающийся список **Bit Depth** позволяет выбрать количество бит, отводимых на информацию о цвете, иначе говоря, цветовой режим. Здесь доступны три пункта:

- ☐ **8-bit** — восьмибитный цвет, 256 доступных цветов;
- ☐ **24-bit** — 24-битный цвет, 16,7 миллионов доступных цветов, TrueColor;
- ☐ **24-bit with Alpha** — 24-битный цвет с каналом прозрачности (альфа-каналом).

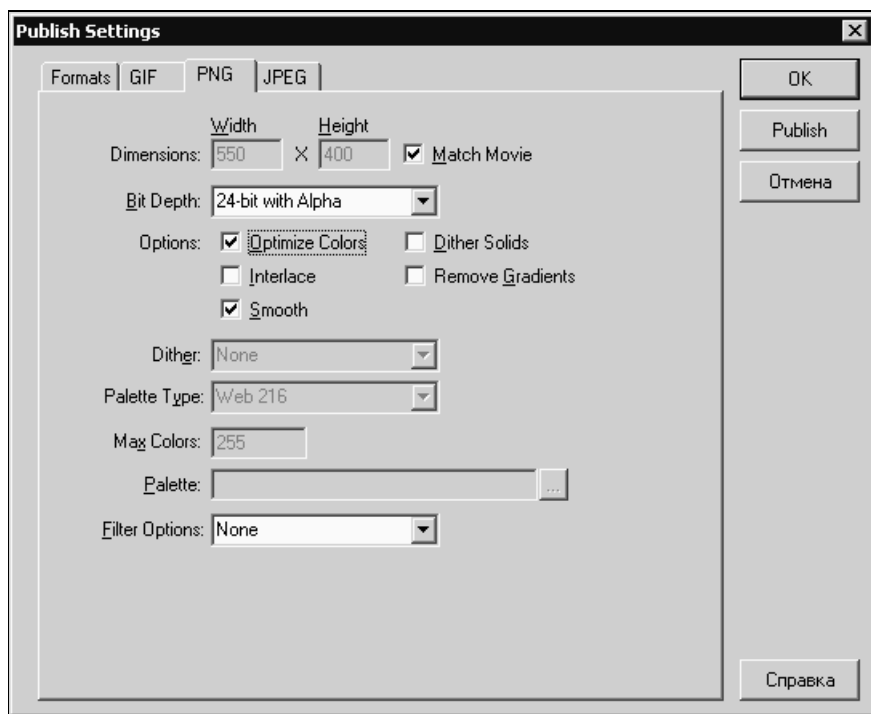


Рис. 11.4. Диалоговое окно **Publish Settings** (вкладка **PNG**)

Группа флажков **Options** управляет параметрами оптимизации изображения, которую проводит при его сохранении Flash. Опишем их подробно.

Флажок **Optimize Colors** включает или отключает оптимизацию палитры цветов изображения. Если она включена, Flash при сохранении удаляет неиспользуемые цвета. Это позволяет уменьшить размер результирующего файла на 1000—1500 байт, однако требует для хранения изображения больше оперативной памяти. Этот флажок лучше держать включенным.

Флажок **Interlace**, будучи включенным, вызывает создание изображения PNG с чередованием. Такой файл отображается Web-обозревателем, постепенно "проявляясь" в процессе загрузки. При создании изображения для публикации в Интернете этот флажок также лучше включить.

Флажок **Smooth** включает или отключает сглаживание контуров результирующего изображения. Сглаженные изображения выглядят лучше, но в некоторых случаях при этом возможно ухудшение качества, также увеличивается размер файла. "Поиграйтесь" с этой настройкой, чтобы выбрать лучшее качество изображения.

Флажок **Dither Solids** включает или отключает использование составных цветов. Подробнее о составных цветах см. в описании вкладки **GIF**.

Флажок **Remove Gradients** позволяет удалить градиенты в результирующем изображении. Как правило, градиенты в изображении формата PNG выглядят очень грубо и увеличивают размер файла. Но включать или не включать этот флажок, решать вам.

Если в раскрывающемся списке **Bit Depth** выбран пункт **8-bit**, то становится доступным раскрывающийся список **Dither**. Он позволяет выбрать способ создания составных цветов для градиентных заливок и предлагает для этого три пункта:

- ☐ **None** — составные цвета не используются, для отображения градиентов берутся наиболее близкие цвета из палитры;
- ☐ **Ordered** — создаются составные цвета приемлемого качества, размер результирующего файла слегка увеличивается;
- ☐ **Diffusion** — создаются высококачественные составные цвета, размер результирующего файла увеличивается еще сильнее.

Пункт **Diffusion** работает только в том случае, если была выбрана безопасная палитра цветов **Web**.

Раскрывающийся список **Palette Type** служит для выбора палитры. В нем доступны следующие четыре пункта:

- ☐ **Web 216** — используется безопасная палитра цветов **Web**;
- ☐ **Adaptive** — используется адаптивная палитра цветов. Если выбран этот пункт списка, становится доступно поле ввода **Max Colors**, где задается максимальное количество доступных в палитре цветов;
- ☐ **Web Snap Adaptive** — аналогичен пункту **Adaptive**, за тем исключением, что при случае будут использоваться цвета из безопасной палитры **Web**;
- ☐ **Custom** — позволяет вам выбрать палитру цветов самому. После выбора этого пункта становится доступно поле ввода **Palette**, где вы сможете ввести имя нужного файла. Поддерживается импорт из файлов палитр АСТ и различных графических файлов.

Последний раскрывающийся список — **Filter** — позволяет задать способ дополнительной обработки массива точек с целью увеличить степень его сжатия. Здесь доступны шесть пунктов:

- ☐ **None** — дополнительная обработка отсутствует;
- ☐ **Sub** — в массиве сохраняется разность между цветом текущего и предшествующего ему пикселей;
- ☐ **Up** — в массиве сохраняется разность между цветом текущего и последующего пикселей;
- ☐ **Average** — для определения цвета текущего пикселя используется среднее значение цветов двух соседних пикселей;
- ☐ **Path** — для определения цвета текущего пикселя берется среднее значение цветов трех соседних пикселей;

- ❑ **Adaptive** — самый сложный способ обработки массива. В сочетании с адаптивной палитрой цветов позволяет сильно уменьшить размер PNG-файла. Рекомендуется к использованию, если изображение будет отображаться на компьютерах с современными мощными видеоподсистемами.

Предварительный просмотр публикуемой графики

Flash предоставляет возможность просмотреть публикуемый файл перед собственно публикацией. Для этого выберите пункт **Publish Preview** в меню **File**. Откроется подменю, содержащее несколько пунктов, соответствующих каждому формату публикации, который вы выбрали. Если вы выберете один из этих пунктов, Flash выполнит экспорт вашего изображения в соответствующем формате и тут же откроет его в программе, зарегистрированной для этого типа файлов. Для файлов GIF, JPEG и PNG такой программой будет установленный на вашем компьютере Web-обозреватель. Когда вы закроете эту программу, созданный Flash файл будет удален.

Вы также можете быстро просмотреть результаты экспорта изображения в так называемый формат по умолчанию. Таким форматом Flash считает тот, что выбран в окне **Publish Settings** (см. рис. 11.1) первым, если считать сверху вниз. В нашем случае таким форматом будет GIF. Чтобы просмотреть опубликованное в таком формате по умолчанию изображение, выберите пункт **Default** — (<Название формата>) в подменю **Publish Preview** меню **File** или нажмите клавишу <F12>.

Экспорт изображения

Разобравшись с публикацией, перейдем к экспорту.

Напомним, чем отличается публикация от экспорта. При публикации результирующее изображение максимально оптимизируется, чтобы достичь максимального качества отображения и минимального размера файла. Такое изображение может передаваться по компьютерным сетям, электронной почте, на дискетах и другими путями, выводиться на экран и распечатываться. Редактирование подобного изображения в других графических программах обычно не предусматривается. При экспорте, наоборот, изображение подготавливается именно к правке в других программах, поэтому оптимизация не проводится.

Как правило, любое изображение, созданное в любой программе векторной графики (не только Flash), требует дополнительной правки. Поэтому, вероятно, не стоит сразу публиковать его, а лучше сначала экспортировать и подправить в другой программе. Впрочем, все это — лишь наши субъективные взгляды, которые вы совсем не обязаны разделять. Поступайте так, как вам удобно.

Форматы экспорта, поддерживаемые Flash

Перечислим все графические форматы, в которые Flash может экспортировать графику. Сразу скажем, что не будем учитывать форматы хранения анимированной графики, т. е. фильмов — экспортировать в них статичную графику все равно бесполезно. Результаты своих изысканий сведем в табл. 11.1.

Таблица 11.1. Форматы экспорта графики, поддерживаемые Flash

Название формата	Расширение файлов
Adobe Illustrator	ai
AutoDesk AutoCAD	dxf
Encapsulated PostScript	eps
FutureSplash	spl
GIF (обычный и анимированный)	gif
JPEG	jpg, jpe, jpeg
Macromedia Shockwave/Flash	swf
PNG	png
Метафайлы Windows	wmf
Растровые файлы Windows	bmp
Расширенные метафайлы Windows	emf

Экспорт графики

Для того чтобы экспортировать графику Flash в другой формат, выберите пункт **Export Image** в меню **File**. На экране появится стандартное диалоговое окно сохранения файла Windows. Выберите нужный формат файла в раскрывающемся списке в нижней части этого окна, задайте имя файла и нажмите кнопку сохранения.

В некоторых случаях Flash просто сохранит созданный в результате экспорта файл на диске. Но чаще на экране появится новое диалоговое окно, где вам будет предложено задать некоторые параметры сохраняемого файла. Задав их, нажмите кнопку **ОК** — и файл будет сохранен. Если вы передумали экспортировать графику, нажмите кнопку **Cancel**.

Ниже будут описаны все эти диалоговые окна и особенности экспорта в различные графические форматы. Если же какой-то формат не будет описан, значит, никаких параметров для его экспорта Flash не потребует.

Adobe Illustrator

Если вы хотите отредактировать созданную во Flash графику в другой программе, экспортируйте ее в формат Adobe Illustrator. Flash поддерживает создание файлов версий 88, 3.0, 5.0, 6.0 и 8.0 этого формата. Имейте только в виду, что градиентные заливки поддерживаются, начиная с версии 5.0, а внедрение растровых изображений — с версии 6.0.

После нажатия кнопки сохранения в диалоговом окне сохранения файла Windows на экране появится новое диалоговое окно, показанное на рис. 11.5. В нем вы можете выбрать версию файла Illustrator.

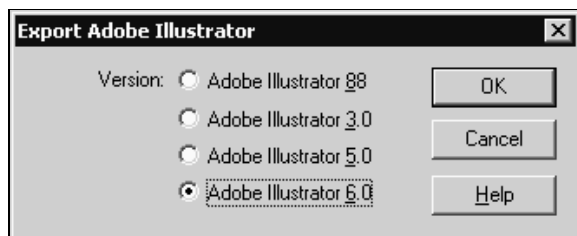


Рис. 11.5. Диалоговое окно **Export Adobe Illustrator**

Большую часть этого диалогового окна занимает группа переключателей **Version**. Всего переключателей в ней четыре: **Adobe Illustrator 88**, **Adobe Illustrator 3.0**, **Adobe Illustrator 5.0** и **Adobe Illustrator 6.0**. Нет нужды объяснять их назначение. Однако, очень странно, что среди них нет переключателя, задающего экспорт в формат **Adobe Illustrator 8.0**. Неужели, это очередная ошибка в этом прекрасном пакете?

GIF

После нажатия кнопки сохранения в диалоговом окне сохранения файла Windows на экране появится другое диалоговое окно, показанное на рис. 11.6. В нем вы можете задать параметры создаваемого файла GIF. Как видите, их не очень много.

Группа **Dimensions** из полей ввода **Width** и **Height** служит для задания соответственно ширины и высоты результирующего изображения. В поле ввода **Resolution** задается разрешающая способность изображения в точках (пикселах) на дюйм. Если вы нажмете кнопку **Match Screen**, Flash установит эти параметры сам, основываясь на размерах рабочего листа и разрешении экрана.

Раскрывающийся список **Include** задает размеры свободного пространства, которое будет включено в состав изображения. Пункт **Minimum Image Area**, выбранный изначально, заставляет Flash включить в изображение минимум пустого пространства. Если же вы выберете пункт **Full Document Size**, Flash включит в состав изображения все свободное пространство, имеющееся на листе.

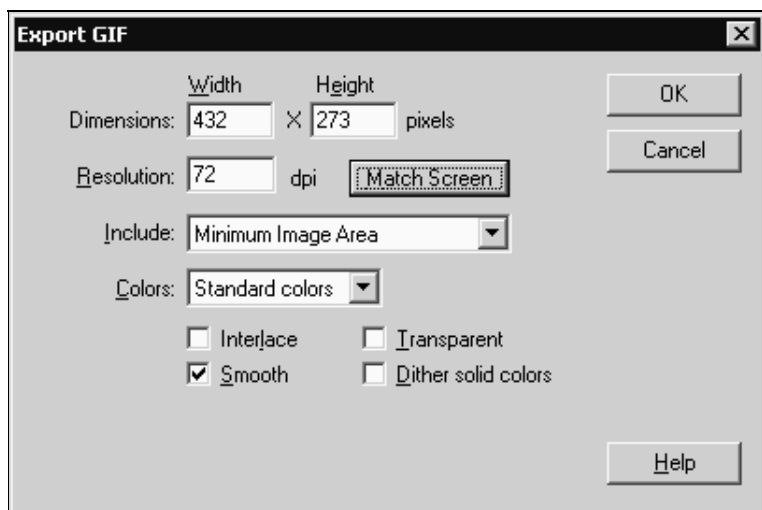


Рис. 11.6. Диалоговое окно **Export GIF**

Раскрывающийся список **Colors** задает количество доступных цветов в цветовой палитре изображения. Доступно восемь пунктов:

- ☐ **Black & White** — черно-белое изображение, два цвета;
- ☐ **4, 8, 16, 32, 64, 128, 256 colors** — соответствующее количество цветов;
- ☐ **Standard colors** — безопасная палитра Web.

Группа флажков, расположенных ниже этого раскрывающегося списка, вам уже большей частью знакома. Флажок **Interlace**, будучи включенным, вызывает создание изображения GIF с чередованием. Флажок **Smooth** включает или отключает сглаживание контуров результирующего изображения. Флажок **Transparent** при включении делает цвет фона рабочего листа прозрачным. А флажок **Dither solid colors** включает или отключает использование составных цветов.

BMP

После нажатия кнопки сохранения в диалоговом окне сохранения файла Windows на экране появится диалоговое окно, показанное на рис. 11.7. В нем вы можете задать параметры создаваемого файла BMP.

Группа **Dimensions** из полей ввода **Width** и **Height** служит для задания соответственно ширины и высоты результирующего изображения. В поле ввода **Resolution** задается разрешающая способность изображения в точках (пикселах) на дюйм. Если вы нажмете кнопку **Match Screen**, Flash установит эти параметры сам, основываясь на размерах рабочего листа и разрешении экрана.

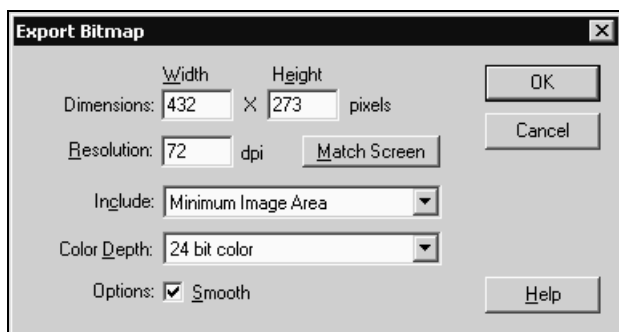


Рис. 11.7. Диалоговое окно **Export Bitmap**

Раскрывающийся список **Include** задает размеры свободного пространства, которое будет включено в состав изображения. Пункт **Minimum Image Area**, выбранный изначально, заставляет Flash включить в изображение минимум пустого пространства. Если же вы выберете пункт **Full Document Size**, Flash включит в состав изображения все свободное пространство, имеющееся на листе.

Раскрывающийся список **Color Depth** задает цветовой режим изображения. Доступно четыре пункта:

- ☐ **8 bit grayscale** — черно-белое изображение с 256 градациями серого цвета;
- ☐ **8 bit color** — 256-цветное изображение;
- ☐ **24 bit color** — изображение с 24-битным цветом (TrueColor);
- ☐ **32 bit color w/ alpha** — изображение с 24-битным цветом и каналом прозрачности (альфа-канал).

Флажок **Smooth** включает или отключает сглаживание контуров результирующего изображения.

AutoDesk AutoCAD

Flash позволяет экспортировать изображения в формате AutoCAD версии 10 или более поздней.

JPEG

После нажатия кнопки сохранения в диалоговом окне сохранения файла Windows на экране появится другое диалоговое окно, показанное на рис. 11.8. В нем вы можете задать параметры создаваемого файла JPEG. Как видите, их не очень много.

Группа **Dimensions** из полей ввода **Width** и **Height** служит для задания соответственно ширины и высоты результирующего изображения. В поле ввода **Resolution** задается разрешающая способность изображения в точках (пикселах) на дюйм. Если вы нажмете кнопку **Match Screen**, Flash установит эти параметры сам, основываясь на размерах рабочего листа и разрешении экрана.

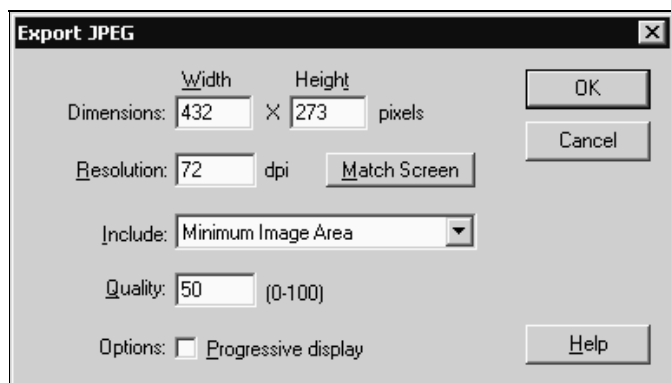


Рис. 11.8. Диалоговое окно **Export JPEG**

Раскрывающийся список **Include** задает размеры свободного пространства, которое будет включено в состав изображения. Пункт **Minimum Image Area**, выбранный изначально, заставляет Flash включить в изображение минимум пустого пространства. Если же вы выберете пункт **Full Document Size**, Flash включит в состав изображения все свободное пространство, имеющееся на листе.

Поле ввода **Quality** позволяет задать качество результирующего изображения от 0 до 100. Чем больше это значение, лучше выглядит изображение и тем больше получается его файл.

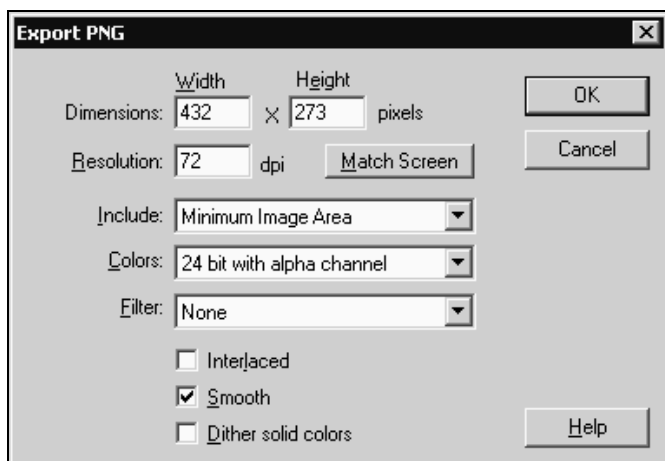
Флажок **Progressive display**, будучи включенным, вызывает создание JPEG-файла с чередованием.

PNG

После нажатия кнопки сохранения в диалоговом окне сохранения файла Windows на экране появится другое диалоговое окно, показанное на рис. 11.9. В нем вы можете задать параметры создаваемого файла PNG.

Группа **Dimensions** из полей ввода **Width** и **Height** служит для задания соответственно ширины и высоты результирующего изображения. В поле ввода **Resolution** задается разрешающая способность изображения в точках (пикселах) на дюйм. Если вы нажмете кнопку **Match Screen**, Flash установит эти параметры сам, основываясь на размерах рабочего листа и разрешении экрана.

Раскрывающийся список **Include** задает размеры свободного пространства, которое будет включено в состав изображения. Пункт **Minimum Image Area**, выбранный изначально, заставляет Flash включить в изображение минимум пустого пространства. Если же вы выберете пункт **Full Document Size**, Flash включит в состав изображения все свободное пространство, имеющееся на листе.

Рис. 11.9. Диалоговое окно **Export PNG**

Раскрывающийся список **Color Depth** задает цветовой режим изображения. Доступно четыре пункта:

- ☐ **8 bit** — 256-цветное изображение;
- ☐ **24 bit** — изображение с 24-битным цветом (TrueColor);
- ☐ **24 bit with alpha channel** — изображение с 24-битным цветом и каналом прозрачности (альфа-канал).

Раскрывающийся список **Filter** позволяет задать способ дополнительной обработки массива точек с целью увеличить степень его сжатия. Здесь доступны шесть пунктов:

- ☐ **None** — дополнительная обработка отсутствует;
- ☐ **Sub** — в массиве сохраняется разность между цветом текущего и предшествующего ему пикселей;
- ☐ **Up** — в массиве сохраняется разность между цветом текущего и последующего пикселей;
- ☐ **Average** — среднее значение цветов двух соседних пикселей используется для определения цвета текущего пикселя;
- ☐ **Path** — среднее значение цветов трех соседних пикселей используется для определения цвета текущего пикселя;
- ☐ **Adaptive** — самый сложный способ обработки массива. В сочетании с адаптивной палитрой цветов позволяет сильно уменьшить размер PNG-файла.

Флажок **Interlace**, будучи включенным, вызывает создание PNG-файла с чередованием. Флажок **Smooth** включает или отключает сглаживание контуров результирующего изображения. А флажок **Dither solid colors** включает или отключает использование составных цветов.

Shockwave/Flash

В этот формат статичная графика, как правило, не экспортируется. Этот формат используется для сохранения разделяемых библиотек. Если же вы все-таки захотите экспортировать в него статичное изображение, обратитесь к *главе 19*.

После нажатия кнопки сохранения в диалоговом окне сохранения файла Windows на экране появится другое диалоговое окно, показанное на рис. 11.10. В нем вы можете задать параметры создаваемого файла Shockwave/Flash.

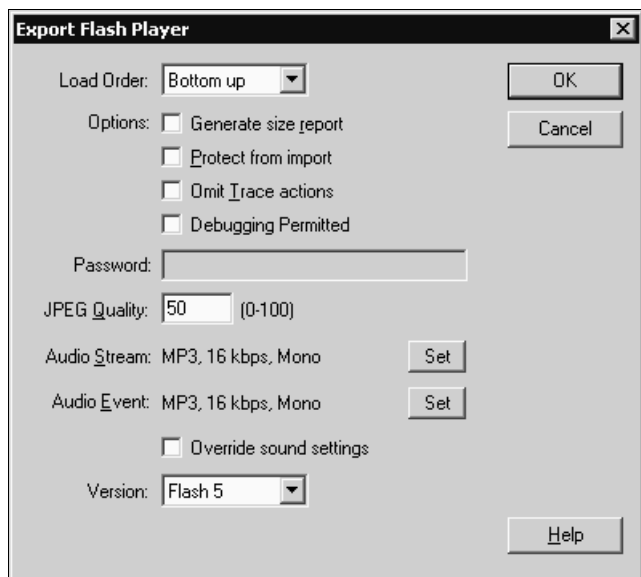
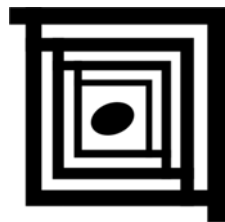


Рис. 11.10. Диалоговое окно **Export Flash Player**

В этом окне нас интересует только раскрывающийся список **Version**. С помощью этого списка вы можете задать версию создаваемого файла Shockwave/Flash. Этот список содержит пять пунктов: **Flash 1**, **Flash 2**, **Flash 3**, **Flash 4**, **Flash 5** и **Flash 6** (то есть, Flash MX). Нет нужды объяснять, что они означают. Учтите только, что ранние версии Flash могут не поддерживать некоторые возможности, которые вы использовали в своей библиотеке.

Если в вашей библиотеке присутствует импортированная растровая графика, вам также может пригодиться поле ввода **JPEG Quality**. Оно позволяет задать качество растровой графики, которое будет достигнуто в результате сжатия в формат JPEG. Вы можете ввести в него значение от 0 до 100, чем оно больше, лучше выглядит изображение и тем больше получится результирующий файл Shockwave/Flash. Однако, на наш взгляд, лучше выбрать степень и алгоритм сжатия для каждого растрового изображения отдельно, это подробно описывалось в *главе 8*.

Часть III



Работа с анимацией

Глава 12. Форматы анимированной графики и видео

Глава 13. Покадровая анимация

Глава 14. Трансформационная анимация

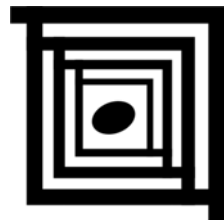
Глава 15. Слои

Глава 16. Импорт анимации и видео

Глава 17. Работа со звуком

Глава 18. Подготовка к экспорту

Глава 19. Публикация и экспорт анимации



Глава 12

Форматы анимированной графики и видео

Вот и добрались мы наконец-то до анимации. Выяснив, как в среде Flash создается статичная, неподвижная графика, перейдем к рассмотрению графики анимированной. Именно в этом — в создании анимированной графики, видеофильмов, "флэшек", как называют их фанаты — и силен Flash. Именно для этого он и создавался.

Но этим мы займемся чуть позже. А пока что — небольшой вводный курс, рассказывающий об анимированной графике и компьютерном видео.

История компьютерного видео насчитывает всего лет десять. И это понятно: чтобы хранить полноразмерные фильмы, нужно очень много дискового пространства, а чтобы его нормально обрабатывать, нужны исключительно мощные компьютеры. Если вы смотрите фильмы в формате MPEG 4 (это могут быть как фильмы на дисках DVD, так и фильмы, записанные на обычных CD в формате DivX), то знаете, что, не имея современного компьютера, о компьютерном видео лучше забыть. Иначе вы рискуете смотреть не фильм, а слайд-шоу со скоростью один кадр секунд этак в пять.

Как раз десять лет назад появились достаточно быстродействующие компьютеры и достаточно емкие жесткие диски, чтобы обрабатывать и хранить компьютерное видео. И именно в то время возникло понятие "мультимедиа" (multimedia), обозначающее совокупность текстовых, цифровых, звуковых и видеоданных, объединенных в единое целое.

Давайте выясним, почему для хранения и обработки видео нужно больше ресурсов, чем для обычной компьютерной графики. Вероятно, вы знаете, что кино- или видеофильм можно представить как набор множества неподвижных изображений (кадров), очень быстро, несколько раз в секунду, сменяющих друг друга. Так как человеческий глаз не может уследить за сменой одного изображения другим, субъективно этот непрерывный "поток" изображений выглядит как одна движущаяся картинка. Для достижения такого эффекта *частота кадров* фильма должна быть довольно велика. В табл. 12.1 приведены стандартные значения частоты кадров, применяемые в кино и на телевидении.

Таблица 12.1. Стандартные значения частоты кадров

Значение частоты, кадров/с	Где применяется
12	Web-анимация, японские мультфильмы "анимэ"
24	Обычное кино
25	Телевидение стандартов SECAM и PAL
30	Телевидение стандарта NTSC

Как видите, каждый фильм фактически представляет собой огромное количество статичных изображений. Чтобы сохранить их в цифровом виде, нужен носитель весьма большой емкости. Поэтому для сохранения фильмов используют алгоритмы сжатия с потерями. Но здесь первопроходцев компьютерного кино подстерегает другой подводный камень: для распаковки таких фильмов и вывода изображения на экран нужны огромные вычислительные мощности. А такие мощности появились в распоряжении рядовых пользователей только десять лет назад. Да и то, тогдашние фильмы не блистали качеством.

Вероятно, самым первым форматом записи фильмов был QuickTime, разработанный фирмой Apple для компьютеров Macintosh и позднее перенесенный на Microsoft Windows. Фирма Microsoft в ответ разработала программный пакет Video for Windows и формат записи фильмов AVI, позволяющий использовать различные алгоритмы сжатия. Несколькоми годами позднее, когда мощности персональных компьютеров еще возросли, независимая группа MPEG разработала очень сильный для своего времени алгоритм сжатия MPEG 1. Именно в этом формате записывались диски VideoCD — первый и не слишком удачный опыт создать цифровое кино "для народа".

Вероятно, вы видели эти диски, не могли не видеть. VideoCD — это обычный компакт-диск, на который записывался один огромный файл формата MPEG 1 с расширением dat; на одном таком диске помещалось около часа видео, поэтому полноразмерные фильмы поставлялись на двух дисках. Качество видео было очень плохим, фильмы смотрели на компьютерах с помощью специальных программ, открывая их в маленьком окошке, или на особых проигрывателях, похожих на видеоманитофоны. Бизнес по продаже дисков VideoCD был развит довольно сильно: автор припоминает, что видел в Волгограде магазин, торгующий фирменными дисками с отечественным кино. Но все же VideoCD — один из тех многочисленных блинов, что, будучи первыми, выходят комом.

Настоящую жизнь компьютерному кино дали появившиеся во второй половине девяностых годов лазерные диски DVD. Эти лазерные диски высокой плотности позволяют записывать до 6 гигабайт информации, чего вполне хватает не только на полноразмерный фильм высокого качества, но и на

несколько звуковых дорожек, интерактивные меню и прочие "бонусы". Для записи фильмов на такие диски используется алгоритм MPEG 2.

Существуют также альтернативные форматы записи видео на обычные компакт-диски. Разработан SuperVideoCD, на который помещается до двух часов видео, сжатого по алгоритму MPEG 2. Создан великий и ужасный DivX — альтернативная реализация алгоритма MPEG 4 — позволяющий записывать полноразмерный фильм отличного качества на один CD. Эти форматы, в основном DivX, составляют серьезную конкуренцию DVD, будущее которого сегодня выглядит не так уж и безоблачно.

Давайте же подробно разберем наиболее популярные из существующих на сегодняшний день форматов записи видео. Но сначала поговорим о двух способах создания анимированного изображения: покадровом и трансформационном.

Покадровая и трансформационная анимация

Эти два способа создания движущегося изображения почти аналогичны двум разновидностям компьютерной графики, описанных нами в *главе 4*. Рассмотрим их подробнее.

Покадровая анимация

Покадровая анимация (ее еще называют классической) представляет собой набор изображений различных фаз движения — *кадров*, прокручиваемых с большой скоростью. Это самый старый и самый надежный способ сохранения движущегося изображения на каком-либо носителе (пленке, бумаге, магнитной ленте, жестком диске, CD). Пример покадровой анимации из пяти кадров показан на рис. 12.1.

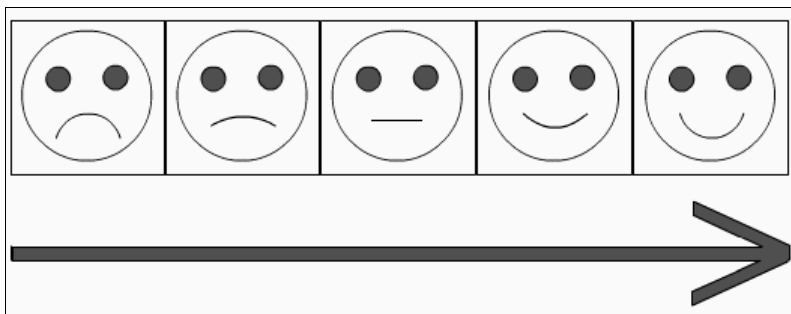


Рис. 12.1. Пример покадровой анимации из пяти кадров (стрелка показывает направление проигрывания)

В чем-то покадровая анимация сродни растровому изображению. И здесь, и там данные сохраняются в виде массива точек, позднее подвергаемого сжатию. В самом деле, отдельные кадры такой анимации сохраняются именно в виде растровых изображений. Векторная графика для этого, насколько нам известно, для создания покадровой анимации не применяется.

Абсолютно все фильмы, накопленные к данному моменту времени трудолюбивым человечеством, представляют собой покадровую анимацию. Еще бы, ведь сам принцип действия киноаппарата основан на фиксации на светочувствительной пленке множества неподвижных изображений, каждое — через определенный промежуток времени. Двадцать четыре (стандартная частота кадров "большого" кино) раза в секунду киноаппарат приказывает: "Остановись, мгновенье". Из многих тысяч таких вот "остановившихся мгновений" и состоит любая кинолента.

Аналогично, путем покадровой фиксации изображения на носителе, работает и видеокамера. Правда, в этом случае процесс создания последовательности кадров не так очевиден: информация записывается в электронном виде на магнитный носитель, и невооруженным глазом ее не увидишь. Но, можете поверить, здесь все абсолютно так же, как в случае с кинокамерой.

А если взять рисованные и кукольные анимационные фильмы, так там покадровая анимация в чистом виде! Каждый кадр фильма рисуется или выстраивается на сцене, после чего кинокамерой делается один-единственный кадр. После этого готовится следующий кадр и т. д., пока не будет готов весь фильм. Адская работа. Конечно, сейчас появилось множество технических новинок, облегчающих труд аниматора, в том числе, и компьютеры. Но принцип остается тот же.

Чем же полюбилась человечеству покадровая анимация? Рассмотрим все ее преимущества.

- ❑ Относительная очевидность (но никак не легкость) создания. В самом деле, чтобы изготовить анимационный фильм, нужно "всего лишь" нарисовать все входящие в него кадры и перевести их на какой-нибудь информационный носитель. (В случае обычного кинофильма это делает сам киноаппарат.) На наш взгляд, достаточно очевидно. (Но отнюдь не просто и совсем не легко.)
 - ❑ Широкие возможности для создателя. Ну, тут уж и говорить не о чем!..
- К несчастью, на этом преимущества покадровой анимации кончаются. И начинаются недостатки.
- ❑ Большая трудоемкость создания фильмов. Если каждый кадр рисуется вручную, и при этом не применяется никаких технических средств, облегчающих работу, процесс создания фильма может затянуться на многие месяцы, а то и годы. (Обычные кинофильмы делаются значительно бы-

стрее, т. к. в них не нужно рисовать кадры — оператор просто фиксирует реальную сцену.) Да и пресловутые технические средства ненамного ускоряют этот процесс.

□ Большие проблемы, возникающие при сохранении покадровой анимации в цифровом виде.

Вот здесь давайте остановимся. И поговорим о переводе фильмов в цифровой вид (оцифровке) и их хранении.

Каждый из множества кадров, составляющих фильм, занимает при хранении определенное пространство на диске. Предположим, что это пространство составляет 100 килобайт — для хранения полноцветного изображения высокого разрешения в формате JPEG этого даже маловато. Теперь предположим, что таких изображений у нас 100 000 — такой длинный у нас фильм. Умножаем 100 на 100 000 и получаем 10 000 000, т. е. примерно 10 гигабайт. (Примерно, потому что гигабайт — это не 1 000 000 000, а 1 073 741 824 байт.) Выходит, для хранения фильма нам нужен целый жесткий диск, а уж сколько для этого понадобится компакт-дисков, просто страшно подумать. Этот фильм даже на DVD не влезет!

Что делать?

Конечно же, надо сжать фильм посильнее. И, если можно, сжать не только сами кадры, но и саму их последовательность, выбросив часть информации, без которой можно и обойтись. И, разумеется, сжать звуковое сопровождение.

Для сжатия фильмов используются специальные алгоритмы, реализующие сжатие с потерями. В этом случае при сжатии какая-то часть информации, не очень нужная при воспроизведении, действительно выбрасывается из фильма, за счет чего последний становится заметно меньше. Более того, эти алгоритмы анализируют каждый кадр фильма и сохраняют в результирующем файле только данные о разнице между соседними кадрами. Это еще сильнее уменьшает размер сжатого фильма.

Вот поэтому для сохранения покадровой анимации и применяются только растровые форматы. Если же сохранить такую анимацию в векторном формате, ее будет очень трудно, практически невозможно сжать с применением одного из обычно применяемых для этого алгоритмов. Да и процессорных ресурсов во время вывода на экран такая анимация будет потреблять значительно больше растровой.

Такие алгоритмы, как MPEG 4 и DivX, позволяют поместить полноразмерный фильм на обычный компакт-диск, т. е. размер сжатого с их помощью видеофайла составляет 600—720 мегабайт. Это уже вполне приемлемые параметры, учитывая объемы современных жестких дисков. Именно эти два алгоритма, как нам кажется, и совершили "компьютерно-киношную" революцию, создав высококачественное цифровое кино "для народа". То есть, сделали то, чего не удалось сделать VideoCD.

Но здесь возникает другая проблема. Сжатые сильными алгоритмами фильмы могут "осилить" только достаточно мощные компьютеры. Если вы попытаетесь просмотреть фильм DivX на компьютере пятилетней давности, то увидите не нормальный фильм, а слайд-шоу. Это происходит потому, что слабосильный процессор не успевает распаковывать данные и выдавать их на экран и поэтому вынужден пропускать целые кадры. К счастью, никому в голову не приходит запускать цифровое кино на старых компьютерах. А современная компьютерная техника сейчас достаточно дешева.

Вот и все о покадровой анимации. Теперь поговорим о ее конкуренте.

Трансформационная анимация

Взглянем еще раз на рис. 12.1. И хорошенько подумаем.

Как вы знаете, отдельные кадры покадровой анимации сохраняются в виде растровых изображений. В векторном виде сохранять их, вроде бы, неудобно — об этом уже говорилось. Теперь давайте предположим, что мы не вняли голосу разума и все-таки сохранили каждый кадр такой анимации в векторном виде. (Скажем честно, такая анимация, какая изображена на рис. 12.1, сама просится в векторный вид. Сами посмотрите — ведь простейшая графика, линии и заливки.) Далее, предположим, что мы можем описывать с помощью формул не только форму кривых и прочих графических примитивов, но и их поведение. Следовательно, мы можем изменить форму "рта", просто вызвав нужную формулу и подставив в нее нужные параметры. Что у нас получится?

А получится у нас *трансформационная анимация*. От покадровой она отличается тем, что не описывает каждый кадр последовательности отдельно, а сразу задает поведение того или иного графического примитива. А собственно созданием анимации, движения, занимается программное обеспечение, поддерживающее соответствующий формат файлов (рис. 12.2).



Рис. 12.2. Пример трансформационной анимации из пяти кадров (стрелка показывает направление проигрывания, серые кадры формируются программой)

Как вы видите, два кадра анимации, показанной на рис. 12.2, мы все-таки должны создать сами. Эти кадры, собственно, и описывают анимацию, а именно, что же в ее результате меняется в нашем изображении. Давайте назовем эти два кадра, созданные нами, *ключевыми* — в дальнейшем этот термин будет применяться очень часто. Сформированные же программой кадры назовем *промежуточными* (на рис. 12.2 они показаны серым цветом).

Внимание!

Данные нами два термина имеют смысл только в контексте трансформационной анимации. В покадровой же анимации все кадры будут ключевыми, а промежуточных кадров не будет вовсе.

Как вы уже поняли, векторная графика и трансформационная анимация — братья навек. Как трансформационная анимация немыслима без векторной графики (к растровому изображению ее применить просто невозможно), так и векторной графике сам Бог велел добавить трансформационную анимацию. Однако хоть векторная графика как способ представления изображений существует довольно давно, трансформационная анимация возникла только в последние годы. Фактически трансформационную анимацию создал пакет Flash, если до него и существовали какие-то аналогичные разработки, то они остались неизвестными широкой публике.

По аналогии с покадровой, перечислим все достоинства и недостатки трансформационной анимации. Начнем, конечно же, с достоинств.

- **Исключительная простота создания.** Вам нужно только растолковать программе, что и как вы хотите изменить, задать промежуток времени, в течение которого будут происходить эти изменения, и некоторые дополнительные параметры — и программа сама сформирует за вас нужную анимацию. Вам не придется кропотливо рисовать все входящие в ваш фильм кадры, как это было в случае с покадровой анимацией.
- **Исключительная компактность получающегося массива данных.** Как вы помните, векторная графика занимает меньше места, чем растровая, так и трансформационная анимация занимает меньше места, чем покадровая. Ведь — согласитесь! — для хранения нескольких параметров функции, задающей анимацию, нужно меньше места, чем для множества кадров, каждый из которых представляет собой растровое изображение.
- **Легкость правки трансформационной анимации.** Чтобы исправить что-то в классическом фильме, вам придется перерисовывать или переснимать целые сцены. В случае же трансформационной анимации вам во многих случаях нужно только изменить пару параметров функции, задающей анимацию.

Полюбовались мы трансформационной анимацией — и хватит! Пора и поругать ее за недостатки. Недостаток, правда, всего один, но зато какой!

Давайте еще раз посмотрим на рис. 12.1 и 12.2 и еще раз подумаем. Что мы можем делать с помощью покадровой и трансформационной анимации? С помощью покадровой — все. А с помощью трансформационной? Не так уж и много — только самые простейшие движения. Все богатство покадровой анимации, увы, нам в этом случае недоступно.

Но трансформационная анимация и не претендует на лавры покадровой. Ее задача — простейшие анимационные эффекты, применяемые в качестве элементов оформления программ и Web-страниц и для простейших, в основном, учебных фильмов. Собственно, она и появилась (вместе с пакетом Flash) как раз в то время, когда начали развиваться технологии, связанные с Интернетом и Всемирной паутиной. На большее ее возможностей не хватит.

Применение разных видов анимации

Исходя из вышесказанного, выделим области двух только что описанных видов анимации. Собственно, мы уже примерно их выделили, теперь же подытожим сказанное выше.

Покадровая анимация незаменима для создания сложных фильмов с богатой графикой. (Мы сознательно отмечаем кинематограф и ограничиваемся той областью, для которой предназначен пакет Flash, — простые фильмы учебного и развлекательного назначения, а также элементы оформления и интерфейсы пользователя.) Художественные и вообще все более-менее сложные фильмы на Flash создаются именно в виде покадровой анимации (за очень редким исключением). Поэтому, если ваши амбиции простираются за пределы самой примитивной Web-графики, наберитесь терпения, ведь рисовать фильм по кадрам, как мы уже говорили, — работа очень трудоемкая.

Трансформационная анимация, наоборот, пригодна для создания простейших анимационных эффектов для Web-страниц. В виде трансформационной анимации также создаются простейшие фильмы рекламного, развлекательного и учебного назначения, например, пресловутые баннеры. Также трансформационная анимация прекрасно подходит для создания на Flash пользовательских интерфейсов и целых программ, всех этих всплывающих меню и нажимающихся кнопок.

Ну и, конечно, никто не запрещает вам смешивать оба эти вида анимации в одном фильме, объединяя их преимущества и избегая недостатков. Таким образом, для простейших случаев создания движений вы можете использовать трансформационную анимацию, для более сложных — покадровую. Flash-художники так и поступают, благо Flash позволяет это делать.

Форматы видеофайлов

Теперь поговорим о самых распространенных на сегодняшний день форматах видеофайлов. А в описании каждого формата также опишем используемые для этого формата алгоритмы сжатия. Практически все эти форматы обеспечивают сохранение только покадровой анимации, кроме самого формата Shockwave/Flash и его предшественника FutureSplash.

QuickTime

Один из самых первых, если не самый первый, формат видеофайлов, получивший широкое распространение. Разработан фирмой Apple в конце 80-х годов, изначально предназначался для использования на компьютерах Macintosh, впоследствии был перенесен в операционную систему Microsoft Windows. Позднее подвергался неоднократным усовершенствованиям. В настоящее время последней версией является 5.0.

Позволяет хранить и видео-, и аудиоинформацию в одном файле с расширением mov. Для сжатия данных используется одноименный алгоритм. Степень сжатия довольно велика, но качество получающегося фильма не очень высоко по сравнению с качеством, обеспечиваемым алгоритмами группы MPEG и DivX. Поэтому формат и алгоритм QuickTime в последнее время теряет свои позиции.

За прошедшие десять лет формат QuickTime получил довольно большую популярность, в основном, для создания коротких музыкальных, рекламных или обзорных видеоклипов. Абсолютный лидер на платформе Macintosh. В Web-дизайне применяется крайне редко. Насколько известно автору, для распространения полноразмерных фильмов вообще не применяется, хотя в этом формате часто распространяются музыкальные видеоклипы.

Внимание!

Flash не поддерживает непосредственно этот формат. Чтобы импортировать фильм QuickTime, вам нужно будет установить на свой компьютер пакет Apple QuickTime версии 4.0 или более поздней.

AVI

Формат данных AVI (Audio and Video Interlaced — чередующиеся аудио и видео) был разработан фирмой Microsoft в начале 90-х годов для использования в мультимедийном программном пакете Microsoft Video for Windows. Для сжатия как аудио-, так и видеоинформации могли использоваться различные алгоритмы, что обеспечивает этому формату большую гибкость.

Позволяет хранить и видео-, и аудиоинформацию в одном файле с расширением avi. (Существует также разновидность этого формата для хранения только звука.) Для сжатия данных применяются следующие алгоритмы:

- ❑ Intel Indeo (старый алгоритм, обеспечивающий довольно слабое сжатие, но без проблем работающий на старых компьютерах, сейчас почти не используется);
- ❑ MPEG 1, 2 и 4, в том числе, DivX (в настоящее время распространены наиболее широко);
- ❑ различные экзотические фирменные алгоритмы.

Формат AVI — абсолютный лидер на платформе Windows. Широко используется для распространения фильмов различной длительности и назначения. Хорошо подходит для сохранения небольших анимационных роликов, применяемых в программах. (В частности, "летающие листочки" в окне копирования Windows хранятся именно в формате AVI.) Формат AVI также находит применение в Web-дизайне.

В настоящее время формат AVI признан устаревшим из-за некоторых ограничений (в частности, размер AVI-файла не может превышать 4 гигабайта). В качестве смены ему подготовлен формат WMV, описанный ниже.

Внимание!

Чтобы импортировать фильм AVI, вам нужно будет установить на свой компьютер пакет Apple QuickTime версии 4.0 или более поздней или библиотеку Microsoft DirectX версии 7.0 или более поздней.

MPEG

Этот формат был разработан в начале 90-х годов группой MPEG (Motion Picture Encoding Group — группа кодирования движущегося изображения) для сохранения фильмов, сжатых с использованием алгоритмов, созданных той же группой. Базируется на формате AVI, но лишен его ограничений.

Файл формата MPEG имеет расширение dat, mpg, mpe, mpeg, mp1, mp2 или mp4. Собственно, набор возможных расширений очень велик, но все они однотипны: как правило, используются буквы m и p.

Как уже говорилось, группой MPEG было разработано три алгоритма для сжатия видеоданных. Перечислим их и кратко опишем.

- ❑ MPEG 1. Самый первый из этого семейства алгоритмов, разработанный еще в начале 90-х. Обеспечивает среднее качество изображения и не очень высокую степень сжатия. Использовался для создания VideoCD. (Существует также разновидность этого алгоритма, предназначенная для сжатия звука, — MPEG 1 level 3 (уровень 3); аудиофайлы, сжатые с помощью этого алгоритма, имеют расширение mp3.)

- ❑ MPEG 2. Был разработан во второй половине 90-х для записи фильмов на диски DVD. Обеспечивает более высокое качество изображения и степень сжатия, чем MPEG 1. Также используется для создания SuperVideoCD.
- ❑ MPEG 4. Был разработан также во второй половине 90-х для распространения фильмов через Интернет. Обеспечивает более высокое качество изображения и степень сжатия, чем MPEG 1 и MPEG 2, также поддерживает различные дополнительные возможности, например, защиту от несанкционированного копирования и создание интерактивных элементов. На сегодня — самый перспективный алгоритм сжатия видеoinформации.

Формат MPEG используется только для распространения фильмов различной длительности и назначения на дисках CD и DVD. (Видеофайлы, записанные на диски, имеют расширение dat.) В Web-дизайне он практически не применяется.

Внимание!

Чтобы импортировать фильм MPEG, вам нужно будет установить на свой компьютер пакет Apple QuickTime версии 4 или более поздней или библиотеку Microsoft DirectX версии 7.0 или более поздней.

DivX

Фактически такого формата не существует. На самом деле, это обычный файл AVI или MPEG, сжатый с использованием алгоритма DivX. Созданный в самом конце 90-х годов на основе алгоритма MPEG 4 группой "независимых программистов" (фактически, хакеров), он обеспечивает еще большую степень сжатия при несколько более низком качестве изображения.

Видеофайлы, сжатые с использованием старых версий этого алгоритма, записывались в формате AVI. Версия 5, появившаяся совсем недавно, также поддерживает формат MPEG. Иногда файлы формата AVI, сжатые этим алгоритмом, имеют расширение divx.

Формат DivX используется, в основном, для распространения полнометражных фильмов на обычных компакт-дисках (для этого он, собственно, и создавался). Очень часто такие диски содержат пиратские копии фильмов, из-за этого формат DivX считается чуть ли не незаконным, "хакерским", хотя в самом его существовании нет ничего противозаконного. Совсем недавно формат DivX получил признание крупных разработчиков игр, распространивших в этом формате демонстрационные ролики. В Web-дизайне формат DivX не применяется.

Внимание!

Этот формат Flash не поддерживается. Чтобы импортировать видеофильм в данном формате, вам придется преобразовать его в один из поддерживаемых форматов, используя специальную программу обработки видео.

WMV

Дальнейшее развитие формата AVI. Разработан фирмой Microsoft в самом конце 90-х годов как часть инициативы Windows Media (создание набора аппаратных и программных средств, направленных на улучшение мультимедийных возможностей современных Windows-совместимых компьютеров). Само название этого формата переводится как Windows Media – Video.

Видеофайлы, сохраненные в этом формате, имеют расширение wmv или asf. (Существует особая разновидность формата — WMA (Windows Media – Audio), предназначенная для хранения звука, такие файлы имеют расширение wma.) По сравнению с AVI формат WMV имеет различные дополнительные возможности, в частности, средства защиты от несанкционированного копирования.

В настоящее время этот формат стремительно набирает популярность. Но используется он опять же для распространения полноразмерных фильмов и видеоклипов.

Внимание!

Чтобы импортировать фильм WMA, вам нужно будет установить библиотеку Microsoft DirectX версии 7.0 или более поздней.

RealMedia

Разработан фирмой RealNetwork в середине 90-х для распространения видео через Интернет. Вероятно, первый видеоформат, ориентированный на использование в Сети. Подвергался неоднократным усовершенствованиям. В настоящее время последней версией является 8.0.

Позволяет хранить и видео-, и аудиоинформацию в одном файле с расширением rm или smil. (Существует также разновидность этого формата для хранения только звука, файлы, записанные в этом формате, имеют расширение ra.) Для сжатия данных используется одноименный алгоритм. Степень сжатия весьма велика, и достигаемое при сжатии качество фильма также достаточно высоко.

В настоящее время удерживает довольно большую долю рынка "сетевого видео" и терять ее не собирается. В основном, применяется для распространения фильмов различной длительности и назначения и трансляции так называемого "интернет-телевидения". В Web-дизайне применяется крайне редко.

Внимание!

Этот формат Flash не поддерживается. Чтобы импортировать видеофильм в этом формате, вам придется преобразовать его в один из поддерживаемых форматов, используя специальную программу обработки видео.

Другие видеоформаты

Здесь мы перечислим другие видеоформаты, используемые в настоящее время. Все они собраны в табл. 12.2. И все они поддерживаются Flash.

Таблица 12.2. Другие видеоформаты

Формат	Расширение файлов	Описание
FutureSplash	spl	Предшественник Macromedia Flash
Shockwave/Flash	swf	—
Анимированный GIF	gif	—
Цифровое видео	dv	Формат записи видео, используемый цифровыми видеокамерами

Использование видеоформатов

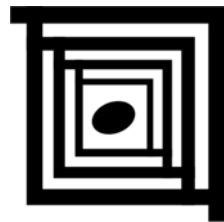
Перечислив и кратко описав каждый из наиболее распространенных на текущий момент времени видеоформатов, выясним, какой формат когда используется.

Вообще, практически всегда фильмы, созданные на Flash, экспортируют в "родной" формат Shockwave/Flash. В самом деле, зачем ломать голову, если подавляющее число интернетчиков имеют на своих компьютерах установленный проигрыватель файлов Shockwave/Flash! Поэтому они в любом случае смогут посмотреть фильм, созданный вами. А формат Shockwave/Flash достаточно компактен, чтобы не слишком загружать не очень-то быстрые на сегодняшний день каналы доступа в Интернет.

Однако в некоторых случаях лучше все же сохранить свое творение в другом формате. Тому есть причины.

Во-первых, если вы создаете очень простую анимацию, то лучше ее сохранить в формате "анимированный GIF". Результирующий файл будет очень компактным, к тому же, его смогут посмотреть даже те, кто не имеет на своем компьютере проигрывателя файлов Shockwave/Flash. Автор, в частности, делает таким способом кнопки для своего сайта, так же создаются рекламные баннеры, мелкие элементы оформления для Web-страниц и программ.

Во-вторых, если вы хотите, чтобы ваш фильм смогли посмотреть все, даже те, кто не имеет установленного проигрывателя Shockwave/Flash, то можете сохранить фильм в формате QuickTime. Учтите только, что этот формат больше популярен среди пользователей Apple Macintosh, чем среди поклонников Microsoft Windows. К тому же, в последнее время он теряет свою былую популярность.



Глава 13

Покадровая анимация

Выяснив все о современных форматах сохранения движущихся изображений, давайте перейдем к их созданию. Вы ведь ждали этого, не так ли?

Flash — пакет, предназначенный, прежде всего, для создания именно анимации (и интерактивности, но об этом поговорим потом, в *части 4*). Неподвижные изображения, конечно, с его помощью создавать тоже можно, но профессиональные художники используют его только тогда, когда под рукой нет ничего более подходящего. Для создания неподвижной графики существуют Adobe PhotoShop, Adobe Illustrator, Macromedia Freehand, Macromedia Fireworks и другие программы. Они приспособлены для этого лучше, чем Flash. Имейте это в виду.

Главное преимущество Flash в том, что он предлагает непревзойденное сочетание достаточно богатых возможностей создания неподвижных изображений, анимации и интерактивности. В этом смысле все вышеперечисленные программы просто отдыхают. При всех их немалых достоинствах никто из них не может того, что может Flash.

Если кто-то скажет, что это не так, напомните ему статистические данные. То, что проигрыватель Flash установлен на 95% всех компьютеров мира, — правда.

Но хватит пустых разговоров! Пора заняться делом.

С чего начнем?

Давайте начнем с покадровой анимации.

Почему?

Практическая реализация покадровой анимации очень проста и наглядна, хоть и весьма трудоемка. Таким образом, мы сможем больше узнать о временной шкале, с которой часто будем работать в дальнейшем. Вы ведь помните важнейший принцип познания: от простого — к сложному. Трансформационная анимация, конечно, значительно менее трудоемка, но овладеть ей сложнее, чем покадровой.

Напомним, что представляет собой покадровая анимация. Это последовательность большого количества однотипных кадров, изображающих различные фазы движения. Такая последовательность очень быстро прокручивается перед зрителем, в результате чего он видит непрерывное движение. Как видите, все достаточно просто, а если вы еще чего-то не поняли, посмотрите на рис. 12.1. Остается добавить, что на принципе покадровой анимации основаны весь кинематограф и все телевидение. Так что с покадровой анимацией вы сталкиваетесь, когда смотрите вечерние новости.

Итак, запаситесь терпением — нам придется много рисовать. Освежите свои знания инструментов рисования Flash. Создайте новый документ Flash. Начинаем!..

Создание покадровой анимации

Начнем с самых основ. А именно, с работы с временной шкалой.

А пока что давайте решим, что будем делать. Точнее, на основе какого примера мы будем изучать создание покадровой анимации. Ранее мы обходились без примера, но сейчас все-таки не тот случай, когда можно изучать основные функции Flash на пальцах.

Предположим, что нам нужно создать небольшой учебный фильм для средней школы. Этот фильм будет описывать процесс деления клетки. Вы еще не забыли школьный курс биологии? Самое время его вспомнить.

Наш фильм будет содержать двенадцать кадров — на первый раз этого хватит. Задайте частоту кадров равной 4, для этого выберите в меню **Modify** пункт **Document**, введите в поле ввода **Frame Rate** диалогового окна **Document Properties** (см. рис. 2.1) число 4 и нажмите кнопку **ОК**. Четырех кадров в секунду школьникам вполне хватит, чтобы понять великое таинство жизни.

Осталось нарисовать первый кадр нашего фильма — единственную клетку, еще не помышляющую о делении. Она будет выглядеть примерно так, как показано на рис. 13.1. Выглядит она, конечно, не очень похоже на живую клетку, слишком упрощенно — всего два эллипса без заливок. Но в данном случае нам нужно не упражняться в художествах, а научиться создавать покадровую анимацию. Этот фильм станет учебным не столько для гипотетических школьников, сколько для нас самих.

Сохраним новый документ Flash под каким-либо именем. И начнем.

Использование временной шкалы. Создание кадров

Прежде всего, обратим внимание на верхнюю часть окна документа Flash. Обычно там отображается временная шкала, с помощью которой создается

анимация, и покадровая, и трансформационная. А именно, в ней схематично показывается последовательность нарисованных нами кадров, любой из которых вы можете выбрать. Содержимое *текущего* кадра отображается внизу, на рабочем листе.

Временная шкала показана на рис. 13.2. Если вы ее почему-то не видите ни "приклеенной" в верхней или нижней части окна документа, ни в виде отдельного небольшого окна, скорее всего, вы ее скрыли. Чтобы вновь вывести ее на экран, включите пункт **Timeline** меню **View** или просто нажмите комбинацию клавиш <Ctrl>+<Alt>+<T>.

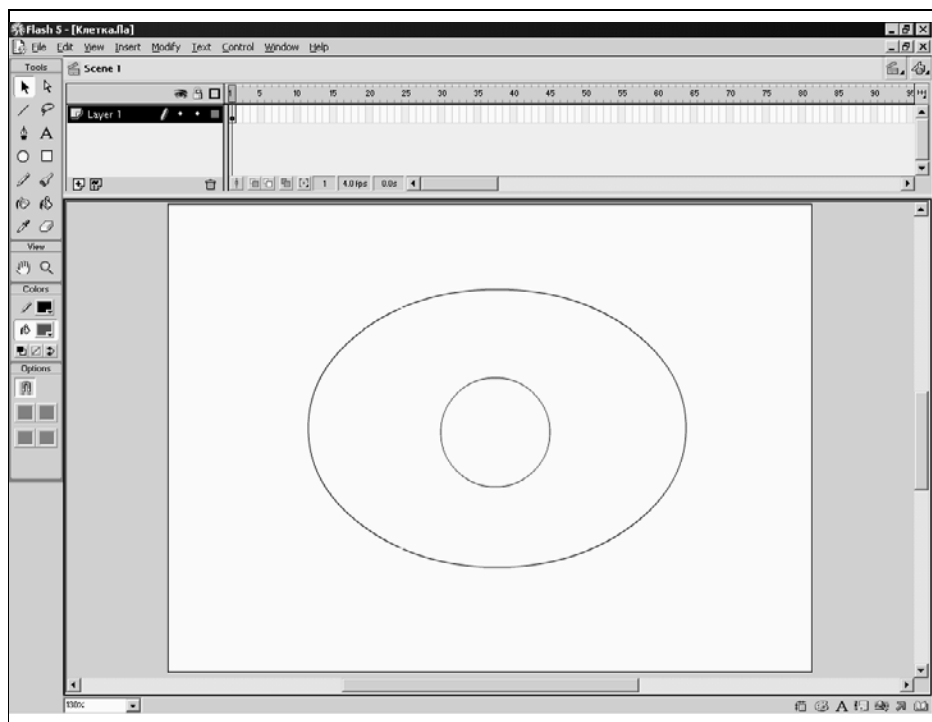


Рис. 13.1. Первый кадр нашего фильма — изначальное состояние клетки

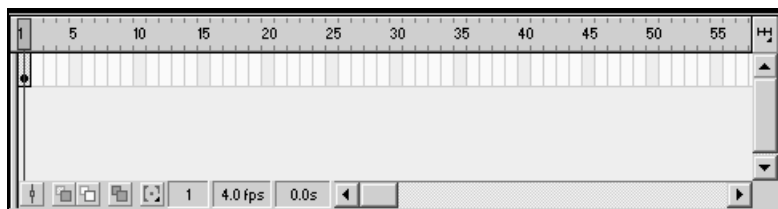


Рис. 13.2. Временная шкала

Как видите, временная шкала представляет собой более или менее длинную последовательность небольших прямоугольников, каждый из которых обозначает кадр. Непустые кадры обозначаются темными прямоугольниками, пустые — либо белыми, либо светло-серыми (каждый пятый кадр). Выше этой последовательности находится шкала, проградуированная в кадрах, таким образом, взглянув на нее, вы всегда можете выяснить номер того или иного кадра.

Ниже последовательности кадров находится строка статуса. В этой строке отображается различная справочная информация, разнесенная по трем секциям (рис. 13.3). В левой секции показывается номер текущего кадра, в средней — частота кадров, а в правой — продолжительность проигрывания всех кадров анимации вплоть до текущего. При двойном щелчке по средней секции на экране появится диалоговое окно **Document Properties** (см. рис. 2.1). Также в строке статуса находятся несколько вспомогательных кнопок (не совсем обычное явление). Эти кнопки служат для задания некоторых дополнительных параметров.

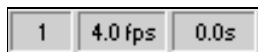


Рис. 13.3. Справочная информация, отображаемая в строке статуса, слева направо: номер текущего кадра, частота кадров, время, прошедшее с начала фильма до текущего кадра

Кроме того, временная шкала содержит две полосы прокрутки и кнопку вызова дополнительного меню, расположенную в ее правом верхнем углу. Для чего служат полосы прокрутки, понятно, а пункты дополнительного меню и их назначение мы рассмотрим далее в этой главе.

Итак, нам нужно создать последовательность кадров, т. е. покадровую анимацию, фильм. Как вы помните из *главы 12*, кадр, описывающий изображение, называется ключевым, кадр, сформированный программой на основе ключевых кадров, — промежуточным. Но, как опять же говорилось в *главе 12*, эта терминология справедлива только для трансформационной анимации, покадровая анимация же состоит из одних ключевых кадров. Поэтому, создавая кадры для нашего фильма, мы будем делать их ключевыми.

Один ключевой кадр у нас уже есть — это наше изначальное изображение готовой к делению клетки. Когда мы его нарисовали, Flash автоматически создал первый кадр анимации и сделал его ключевым. Так что теперь нам нужно просто добавлять новые кадры к уже существующим, формируя последовательность кадров. Ниже приведена последовательность создания покадровой анимации.

1. Создание нового ключевого кадра.
2. Выбор вновь созданного кадра.
3. Рисование изображения для нового кадра.
4. Если не конец фильма, то переход к первому пункту.

Вот так, получилась почти программа работы художника-аниматора.

Ключевой кадр на временной шкале обозначается темно-серым прямоугольником с черным кружком в его нижней половине (рис. 13.4). Этот кружок обозначает, что ключевой кадр содержит изображение. Ключевой кадр, не содержащий никакого изображения, пустой, обозначается белым прямоугольником с черной рамкой (рис. 13.5). Пустое же место в последовательности кадров обозначается белым или светло-серым прямоугольником вообще без рамки.



Рис. 13.4. Ключевой кадр, содержащий изображение



Рис. 13.5. Ключевой кадр, не содержащий изображения, пустой

Вы можете щелкнуть по любому кадру анимации мышью, чтобы выделить его, сделать текущим кадром. Выделенный кадр отображается в инверсном начертании: ключевой — черным прямоугольником с белым кружком, пустой ключевой — просто черным прямоугольником. Содержимое текущего кадра при этом отобразится на рабочем листе, так что вы сразу увидите, какое изображение содержит этот кадр, причем содержимое это будет выделено целиком. Также вы можете выбрать мышью пустое пространство в последовательности кадров, оно в этом случае отображается темно-синим прямоугольником без рамки.

Кроме того, если вы щелкнули по кадру, а не по пустому месту, на выбранный кадр перемещается *указатель* — тонкая красная вертикальная линия (рис. 13.6). Этот указатель помечает кадр, с которого начнется проигрывание фильма. А номер выбранного кадра и время, прошедшее с начала фильма, появляются в строке статуса временной линии (см. рис. 13.3).



Рис. 13.6. Указатель

Впрочем, вы можете перемещать указатель, перетаскивая его за красный прямоугольник, расположенный на шкале. При этом на рабочем листе будут быстро сменяться кадр за кадром.

Чтобы создать новый ключевой кадр, выделите первое пустое место правее последнего ключевого кадра и выберите пункт **Keyframe** в меню **Insert** или

пункт **Insert Keyframe** в контекстном меню выделенного "пустого места". В результате любого из этих действий будет создан и сделан текущим новый ключевой кадр.

При создании нового ключевого кадра содержимое предыдущего (расположенного правее) ключевого кадра копируется во вновь созданный. Таким образом, вам не нужно рисовать каждый кадр "с нуля" — вам нужно будет только слегка подправить предыдущее изображение. Если же вы хотите создать действительно пустой ключевой кадр, то выберите пункт **Blank Keyframe** в меню **Insert** или пункт **Insert Blank Keyframe** в контекстном меню временной шкалы. Пустой ключевой кадр может понадобиться, например, при переходе к другой сцене фильма, когда изображение резко меняется на другое: в этом случае проще нарисовать его заново, а не изменять существующее.

Иногда бывает нужно, чтобы в течение какого-то времени изображение не менялось. Достигается это двумя путями.

Во-первых, вы можете создать столько одинаковых ключевых кадров, сколько нужно. Для этого вы должны просто создать соответствующее количество ключевых кадров, ничего не меняя в их содержимом. Этот путь, однако, очень трудоемок, а множество одинаковых кадров ведет к увеличению размера файла фильма.

Во-вторых, вы можете "растянуть" один ключевой кадр на несколько делений шкалы. При "растягивании" к обычному ключевому кадру добавляется несколько промежуточных кадров, которые вместе и создают так называемый *растянутый кадр*. Такой кадр не увеличивает размера фильма, т. к. фактически хранит только одно изображение, заданное в ключевом кадре.

Чтобы создать растянутый кадр, выделите ключевой кадр, который хотите "растянуть", и либо выберите пункт **Frame** в меню **Insert**, либо выберите пункт **Insert Frame** в контекстном меню, либо нажмите клавишу <F5>. К выделенному ключевому кадру будет добавлен новый промежуточный кадр. Добавьте таким образом необходимое количество промежуточных кадров — и растянутый кадр готов.

Растянутый кадр имеет вид серого прямоугольника, занимающего несколько делений шкалы кадров, и показан на рис. 13.7. Слева находится черный или белый кружок — признак ключевого кадра, а справа — светлый прямоугольник, *конечный маркер* растянутого кадра. Вы также можете выделять мышью любой ключевой или промежуточный кадр, входящий в состав растянутого кадра, и выполнять над ним различные действия. Если вы щелкнете по промежуточному кадру дважды, то будет выделен весь растянутый кадр.



Рис. 13.7. Растянутый кадр

Внимание!

Если на вкладке **General** диалогового окна **Preferences** был включен флажок **Span Based Selection**, при щелчке по растянутому кадру будет выделен весь этот кадр.

Создайте таким образом весь фильм. Чтобы "разделить" клетку пополам, вы можете выделить поочередно сначала одну, потом вторую ее половины и "разнести" их по разным концам листа. Вспомните, ведь вы можете выделять части графического изображения перетаскиванием мыши. Не допускайте очень резких перемещений изображений на разных кадрах: это вызовет неприятное дерганье анимированного изображения при просмотре. Разумеется, вы можете использовать при рисовании любые инструменты, предлагаемые Flash, в том числе, образцы, импортированные изображения и текстовые блоки.

Не особенно заботьтесь о художественных достоинствах вашего первого фильма, ведь нам в данном случае важнее понять сам процесс, чем создать законченное произведение. И не забудьте сохранить готовый фильм на диске, чтобы не потерять при случайном отключении питания. Помните, что очень часто возвышенные мечты губятся самыми что ни на есть прозаическими вещами.

Просмотр фильма в среде Flash

Простейший способ просмотреть созданный фильм прямо в среде Flash — нажать клавишу <Enter>. Также вы можете выбрать пункт **Play** меню **Control**. Flash тотчас проиграет вам ваше творение с заданной частотой кадров, начиная с текущего кадра, и остановится на последнем кадре. Чтобы быстро переместить указатель на первый кадр фильма, выберите пункт **Rewind** в меню **Control** или нажмите комбинацию клавиш <Ctrl>+<Alt>+<R>.

Вы также можете перемещаться с кадра на кадр по фильму в обоих направлениях: вперед (к концу) и назад (к началу). Для этого нажимайте соответственно клавиши <.> и <,>. Вы также можете выбирать пункты **Step Forward** и **Step Backward** в меню **Control**. Ну и, конечно, вы можете перетаскивать указатель кадра вручную в любом направлении.

Чтобы заикнуть фильм (сделать так, чтобы он прокручивался бесконечно), включите пункт-выключатель **Loop Playback** в меню **Control**. Для прерывания этой "бесконечной истории" нажмите клавишу <Enter> или <Esc> или выберите пункт **Stop** в меню **Control**.

Flash также предоставляет для управления показом фильма один из вспомогательных инструментариев — инструментарий управления проигрыванием. Чтобы вызвать его на экран, выберите пункт **Controller** в подменю **Toolbars** меню **Window**. Этот инструментарий показан на рис. 13.8.



Рис. 13.8. Вспомогательный инструментарий управления проигрыванием

Перечислим все кнопки этого инструментария в порядке слева направо:

- ☐ остановка проигрывания;
- ☐ "перемотка" в начало, т. е. перемещение указателя на первый кадр фильма;
- ☐ перемещение на кадр назад, т. е. к началу фильма;
- ☐ запуск проигрывания;
- ☐ перемещение на кадр вперед, т. е. к концу фильма;
- ☐ "перемотка" в конец, т. е. перемещение указателя на последний кадр фильма.

На наш взгляд, ценность этого инструментария не очень велика. В подавляющем большинстве случаев для просмотра фильма хватает средств, предоставляемых пунктами меню **Control**. Инструментарий управления проигрыванием, возможно, будет полезен только тогда, когда вы создаете художественный фильм или когда вы хотите просмотреть фильм, созданный вашим коллегой, да и то вряд ли. Хотя это, конечно, дело личного вкуса пользователя.

Еще одно средство Flash будет вам полезно, но не сейчас, а в дальнейшем, когда вы начнете создавать и просматривать интерактивные фильмы. Это просмотр фильма в отдельном окне. При этом Flash автоматически сформирует файл Shockwave/Flash, загрузит в отдельное окно проигрыватель и проигрывает фильм. Чтобы просмотреть фильм в отдельном окне, выберите пункт **Test Movie** в меню **Control** или нажмите комбинацию клавиш <Ctrl>+<Enter>. Насладившись своим творением, закройте окно, в котором отображается фильм, щелкнув по кнопке закрытия окна (но не закройте случайно саму программу!) или выбрав пункт **Close** в меню **File**.

Учтите, что в отдельном окне фильм всегда показывается зацикленным, несмотря на то, включен пункт-выключатель **Loop Playback** в меню **Control** или нет. Чтобы отключить зацикливание фильма, вам нужно будет отключить пункт-выключатель **Loop** меню окна, в котором будет проигрываться фильм.

Просматривая фильм в отдельном окне, вы можете задавать качество его отображения. Для этого выберите один из пунктов подменю **Quality** меню **View**: **Low** (низкое качество), **Medium** (среднее качество) или **High** (высокое качество, выбран по умолчанию). Также вы можете задавать масштаб отображения фильма, пользуясь уже знакомыми вам пунктами меню **View**: **Zoom In** (увеличение), **Zoom Out** (уменьшение) и подменю **Magnification**.

Правка анимации

Любое, даже, на первый взгляд, совершенное, творение может быть еще немного усовершенствовано. Но, поскольку в мире нет ничего совершенного, переделывать уже сделанное нам приходится очень часто. Не зря же древние греки говорили: "чаще поворачивай стиль".

Фильмы Flash — не исключение из этого правила. Посмотрим, что предлагает нам Flash для того, чтобы исправить уже сделанное.

Работа с кадрами

Вы можете добавить в конец последовательности новый ключевой кадр. Для этого выделите первое пустое место правее последнего ключевого кадра и выберите пункт **Keyframe** в меню **Insert**, либо выберите пункт **Insert Keyframe** в контекстном меню этого пустого места. Если вы хотите создать пустой ключевой кадр, то выберите пункт **Blank Keyframe** в меню **Insert** или пункт **Insert Blank Keyframe** в контекстном меню.

Вставить новый ключевой кадр в середину последовательности несколько сложнее. Для этого вам придется сначала освободить для него место в последовательности кадров, перетаскив все кадры, расположенные правее точки вставки, вправо на нужное количество кадров. Flash "растянет" предыдущий кадр, добавив в нему несколько промежуточных кадров. Эти-то промежуточные кадры мы и превратим в ключевые. Просто выделите нужный промежуточный кадр и выберите один из уже знакомых вам пунктов **Insert Keyframe** и **Insert Blank Keyframe**.

Преобразовать промежуточный кадр в ключевой можно и другим способом. Вероятно, так будет даже проще. Для этого вам следует выделить нужный кадр и выбрать в подменю **Frames** меню **Modify** или контекстном меню выделенного кадра пункты **Convert to Key Frames** (клавиша <F6>) или **Convert to Blank Key Frames** (клавиша <F7>). Первый пункт преобразует "кадр" в ключевой кадр и копирует в него изображение из растянутого кадра. Вторым пунктом преобразует "кадр" в пустой ключевой кадр, в котором вы в дальнейшем сможете нарисовать любое изображение.

Также возможно обратное преобразование — из ключевого кадра в промежуточный, (то есть, можно удалить ключевой кадр, сделав его частью растянутого кадра). Для этого выделите нужные ключевые кадры и выберите пункт **Clear Keyframe** в меню **Insert** или контекстном меню или просто нажмите комбинацию клавиш <Shift>+<F6>.

Вы можете создать новый ключевой кадр, являющийся точной копией уже существующего в фильме ключевого кадра. Для этого щелкните кадр, который хотите скопировать, не отпуская кнопки мыши, нажмите клавишу <Alt>, перетащите кадр на нужное место и отпустите кнопку. Также вы можете копировать целые последовательности кадров.

Вы можете править изображение, содержащееся в текущем ключевом кадре. Имейте только в виду, что, если вы хотите заменить во всей анимации один экземпляр другим, созданным на основе другого образца, вы должны выполнить эту замену во всех ключевых кадрах, где он присутствует.

Вы можете переупорядочивать кадры анимации, перетаскивая их с места на место. Это выполняется мышью. Кадры могут быть перемещены как на свободное место в последовательности, так и друг на друга. Последнее, однако, мы делать не рекомендуем. Чтобы переместить сразу несколько кадров, выделите их, щелкая мышью и удерживая нажатой клавишу <Shift>. Также вы можете выделить сразу последовательность кадров. Для этого щелкните по первому из них, нажмите клавишу <Ctrl> и продолжайте щелкать по остальным нужным кадрам, не отпуская этой клавиши.

Если вы перетащите все кадры, начиная с выделенного и кончая последним кадром последовательности, вправо, Flash автоматически "растянет" предыдущий кадр. Как видите, Flash не терпит "пустоты" в середине последовательности кадров и заполняет ее всеми способами, вплоть до создания новых растянутых кадров. Поэкспериментируйте с созданной вами анимацией, чтобы понять логику Flash, и вы сможете пользоваться в дальнейшей работе этой его особенностью.

Вы также можете менять длительность растянутого кадра, перетаскивая мышью его конечный маркер и удерживая нажатой клавишу <Ctrl>. Таким же образом можно "съежить" растянутый кадр до обычного, занимающего одно-единственное деление шкалы кадров. Учтите, что при этом Flash заполнит "опустевшее" пространство в последовательности вновь созданными ключевыми кадрами.

Разумеется, вы можете удалять ненужные кадры. Для этого выделите в последовательности кадр, который хотите удалить, и выберите пункт **Remove Frames** в меню **Insert** или контекстном меню кадра или нажмите комбинацию клавиш <Shift>+<F5>. Выделенный вами кадр будет удален, а все кадры, расположенные правее его, сдвинутся влево, заполняя "вакантное" место. Вы можете удалить таким же образом сразу несколько кадров.

Внимание!

Не пользуйтесь для удаления кадров клавишей . Она удаляет содержимое выделенного кадра, но не сам кадр.

Для помещения выделенных кадров в буфер обмена Windows и вставки их оттуда Flash предлагает специальные пункты меню **Edit**. Сейчас мы их рассмотрим.

Пункты **Cut Frames** и **Copy Frames** меню **Edit** (и соответствующие им клавишные комбинации <Ctrl>+<Alt>+<X> и <Ctrl>+<Alt>+<C>) позволят вам соответственно переместить и скопировать выделенные кадры в буфер

обмена. Пункт **Paste Frames** меню **Edit** (клавишная комбинация <Ctrl>+<Alt>+<V>) вставляет кадры из буфера обмена туда, где находится выбранный в данный момент кадр (или пустое место), уже имеющиеся кадры будут отодвинуты вправо. Таким образом, вы можете копировать и перемещать кадры и последовательности кадров, не пользуясь мышью. А пункт **Select All** меню **Edit** и одноименный пункт контекстного меню временной шкалы (клавишная комбинация <Ctrl>+<A>) позволит вам выбрать сразу все кадры последовательности.

В меню **Edit** имеется также пункт **Clear Frames** (клавишная комбинация <Alt>+<Backspace>). Выбрав его, вы удалите все содержимое выделенного кадра. Так что будьте осторожны.

Все пять вышеперечисленных пунктов находятся также в контекстном меню выделенного кадра.

Вы можете инвертировать всю последовательность кадров или ее фрагмент, т. е. повернуть ее задом наперед. Для этого выделите нужные кадры (или сразу всю последовательность) и выберите пункт **Reverse** подменю **Frames** меню **Modify** или пункт **Reverse Frames** контекстного меню выделенных кадров. Таким образом, вы можете заставить две клетки сливаться в одну, что, конечно, не происходит в природе.

Но самое интересное: заставить клетку сначала делиться, как обычно, а потом — наоборот сливаться воедино. Делается это очень просто. Ниже приведена последовательность действий (предположим, что фильм "Деление клетки" полностью готов).

1. Выделяем все кадры фильма и копируем их в буфер обмена. (Пункт **Copy Frames** меню **Edit**, если вы забыли.)
2. Выделяем пустое место сразу же за последним кадром последовательности и вставляем кадры из буфера обмена. (Пункт **Paste Frames** меню **Edit**.) Это будет вторая, "фантастическая", половина фильма, описывающая процесс слияния двух клеток в одну и следующая сразу же за первой, "реалистической", частью фильма.
3. Выделяем все только что вставленные кадры ("фантастическую" часть) и инвертируем их. (Пункт **Reverse Frames** контекстного меню.)
4. Проверяем результат.

Дополнительные возможности работы с кадрами

Flash предоставляет несколько дополнительных возможностей работы с кадрами анимации. Они связаны с весьма забавной и зачастую полезной функцией, позволяющей видеть на рабочем листе содержимое сразу нескольких кадров. Это может понадобиться, например, чтобы оценить, не содержит ли ваша анимация резких рывков (которые отнюдь ее не украсят).

Чтобы включить эту функцию, нажмите кнопку-выключатель **Onion Skin**, находящуюся в строке статуса временной шкалы и показанную на рис. 13.9. На рабочем листе будет показано нечто, похожее на рис. 13.10.



Рис. 13.9. Кнопка **Onion Skin**

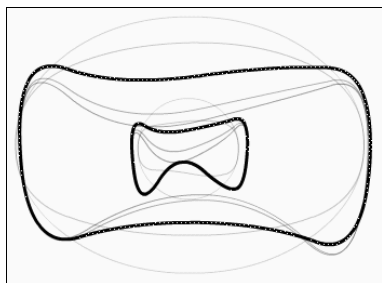


Рис. 13.10. Вид рабочего листа при включенном режиме отображения содержимого нескольких кадров

Как видите, содержимое текущего кадра отображается обычными линиями в обычных цветах. Содержимое же нескольких других кадров выводится тонкими серыми линиями в приглушенных цветах.

Существует также возможность вывести содержимое невыделенных кадров в одних линиях, без заливок. (Содержимое выделенного кадра будет выводиться полностью, со всеми заливками.) Для этого нажмите кнопку-выключатель **Onion Skin Outlines**, находящуюся в строке статуса временной шкалы и показанную на рис. 13.11. Такая функция может быть полезна на медленных компьютерах, чтобы ускорить перерисовку изображения. Также это может помочь, если ваше изображение имеет много заливок, и вы хотите временно скрыть их, чтобы изменить контуры фигур.



Рис. 13.11. Кнопка **Onion Skin Outlines**

Onion Skin и **Onion Skin Outlines** — кнопки-выключатели с зависимой фиксацией. Это значит, что одновременно может быть включена только одна из этих кнопок, либо обе этих кнопки могут быть отключены (тогда Flash показывает на рабочем листе только один — текущий — кадр).

В обоих этих случаях вы можете править только содержимое текущего кадра, это сделано, чтобы вы случайно не изменили другие кадры анимации.

Но если вы все-таки хотите иметь возможность править содержимое всех кадров, выведенных на рабочем листе, включите кнопку-выключатель **Edit Multiple Frames**, находящуюся в строке статуса временной шкалы и показанную на рис. 13.12. Только будьте внимательны: правя сразу несколько кадров, трудно уследить за всеми. Вообще, на наш взгляд, эту функцию (правку нескольких кадров) лучше использовать как можно реже.



Рис. 13.12. Кнопка **Edit Multiple Frames**

Теперь обратите внимание на временную шкалу. Вы увидите нечто, напоминающее рис. 13.13. Темно-серая полоса, охватывающая все отображаемые на рабочем листе кадры, называется *диапазоном кадров*. Этот диапазон ограничен с обоих краев *маркерами диапазона* — небольшими серыми кружками. Вы можете перемещать эти маркеры мышью, меняя размер диапазона.



Рис. 13.13. Диапазон кадров

Существует еще одна возможность задания размера диапазона кадров. Вы можете воспользоваться пунктами меню, появляющегося при нажатии кнопки **Modify Onion Markers**. Эта кнопка также находится в строке статуса временной линии и показана, вместе с раскрытым меню, на рис. 13.14.

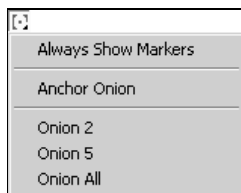


Рис. 13.14. Кнопка **Modify Onion Markers** с раскрытым меню

Для задания размеров диапазона кадров служат три пункта этого меню:

- ☐ **Onion 2** — по два кадра с обеих сторон указателя кадра;
- ☐ **Onion 5** — по пять кадров с обеих сторон указателя кадра;
- ☐ **Onion All** — все кадры последовательности.

Кроме этих трех, меню кнопки **Modify Onion Markers** содержит еще два пункта-выключателя. Пункт **Always Show Markers** заставляет Flash показывать диапазон кадров даже тогда, когда не включены ни кнопка **Onion Skin**,

ни кнопка **Onion Skin Outlines**. А пункт **Anchor Onion** "закрепляет" диапазон кадров на месте, в то время как указатель кадра может свободно перемещаться по шкале (обычно диапазон перемещается вместе с указателем). Вероятно, можно найти ситуацию, когда эти специальные функции действительно пригодятся, во всяком случае, разработчики Flash в это верят.

Дополнительные возможности временной шкалы

Уже довольно хорошо знакомая нам временная шкала Flash таит в себе несколько сюрпризов. Если их использовать с умом, эти сюрпризы станут приятными. Давайте же познакомимся с ними.

Наш первый фильм оказался довольно коротким — всего 12 кадров. Но недалек тот день, когда количество кадров ваших фильмов будет исчисляться многими десятками и сотнями. В таких случаях Flash не хватит ширины окна, чтобы отобразить все эти кадры на временной шкале. Выход из этого положения один: воспользоваться горизонтальной полосой прокрутки. Вот как раз в этом случае вам весьма поможет одна небольшая возможность, предусмотренная Flash.

Предположим, что вы поместили указатель на какой-либо кадр, а потом прокрутили временную шкалу так, что этого указателя больше не видите. В этом случае нажмите кнопку **Center Frame**, расположенную в строке статуса временной шкалы и показанную на рис. 13.15. Flash прокрутит временную шкалу так, чтобы кадр, помеченный указателем, и сам указатель оказались в ее середине.



Рис. 13.15. Кнопка **Center Frame**

Кроме того, вы можете изменять ширину прямоугольников, обозначающих кадры. Для этого вам нужно воспользоваться дополнительным меню временной шкалы. Это меню можно открыть, нажав кнопку, расположенную в правом верхнем углу временной шкалы, выше вертикальной полосы прокрутки. Эти кнопка и раскрытое меню показаны на рис. 13.16.

Для задания ширины прямоугольников-кадров служат пять пунктов-переключателей: **Tiny** (самые узкие прямоугольники), **Small**, **Normal** (значение по умолчанию), **Medium** и **Large** (самые широкие прямоугольники). Одновременно может быть включен только один из этих пунктов.

Если вы выключите пункт-выключатель **Tinted Frames**, находящийся в том же дополнительном меню, прямоугольники-кадры не будут окрашиваться в серый цвет. Они станут такими же белыми, как пустые места в последовательности кадров. Вероятно, кому-то так нравится...

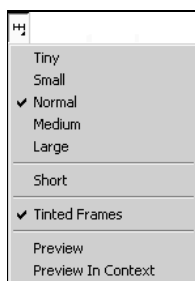


Рис. 13.16. Дополнительное меню временной шкалы

Последние два пункта-переключателя — **Preview** и **Preview in Context** — задают вид кадров во временной шкале. По умолчанию, когда оба они отключены, Flash показывает кадры в виде простых прямоугольников. Чтобы просмотреть какой-либо кадр, вам нужно будет выделить его, а чтобы просмотреть сразу несколько кадров, — воспользоваться режимом показа нескольких кадров на рабочем листе.

Если включить пункт **Preview**, кадры на временной шкале будут показаны в виде достаточно крупных прямоугольников, содержащих изображения, показывающие содержания этих кадров (рис. 13.17). Таким образом, вы всегда можете видеть, что находится в ваших кадрах. Причем, как видите, показывается только собственно изображение, а все свободное пространство листа отрезается.

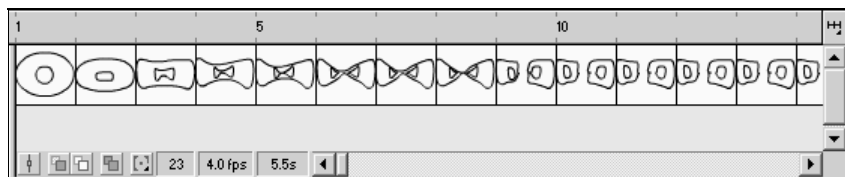


Рис. 13.17. Вид временной шкалы при включенном пункте **Preview** дополнительного меню

Пункт **Preview In Context** отличается от пункта **Preview** тем, что в каждом прямоугольнике-кадре показывается весь рабочий лист (рис. 13.18). Пустое пространство, если оно есть, не обрезается, так что изображение может получиться меньше. Однако вы всегда можете видеть, где именно на рабочем листе находится изображение и куда оно перемещается (если, конечно, вы предусмотрели такое перемещение).

Платой за наглядность служит серьезное увеличение размера прямоугольников-кадров. Поэтому оба вышеописанных режима лучше использовать только время от времени, когда без него действительно не обойтись. В остальное время лучше выбрать обычный режим отображения кадров в виде простых прямоугольников.

Пункты **Preview** и **Preview In Context** входят в ту же группу, что и **Tiny**, **Small**, **Normal**, **Medium** и **Large**. Это значит, что в этой группе может быть включен только один из пунктов. И, чтобы вернуться к обычному, схематичному режиму показа кадров, вам нужно включить один из пунктов: **Tiny**, **Small**, **Normal**, **Medium** или **Large**.



Рис. 13.18. Вид временной шкалы при включенном пункте **Preview In Context** дополнительного меню

Использование сцен

Любой достаточно длинный и сложный фильм делится на отдельные *сцены* — законченные эпизоды, в достаточной степени не зависящие друг от друга. То же самое справедливо и для фильмов, созданных во Flash. Например, в нашем случае, в качестве отдельных сцен могут быть оформлены заголовки фильма, сам фильм и титры.

В данном случае, сцены служат, скорее, для логического разбиения фильма на части. Flash проигрывает сцены одну за другой без остановок. Если же вы хотите, чтобы после проигрывания одной сцены наступала пауза или происходило что-то иное, вам нужно создать соответствующие сценарии (о сценариях и их программировании см. *часть 4*). Поэтому, если ваш фильм очень короток (как, например, наше первое творение "Делящаяся клетка"), вы можете и не использовать сцены. В самом деле, зачем выполнять лишнюю работу, если она бесполезна?

Каждая сцена должна иметь уникальное имя. Вы можете дать сценам имена, отражающие их содержание, например, Название, Фильм и Титры. По умолчанию Flash дает сценам имена вида "Scene<номер>". Изначально фильм состоит из одной-единственной сцены, имеющей название "Scene1".

Нумерация кадров фильма при использовании сцен остается сквозной, проходящей через все сцены. Например, если в фильме имеются три сцены, по десять кадров каждая, то первая сцена будет содержать с первого по десятый кадры, вторая — с 11 по 20, а третья — с 21 по 30. Однако это справедливо только для результирующего файла Shockwave/Flash. В документе Flash, с которым вы работаете, кадры для вашего удобства нумеруются отдельно в каждой сцене.

Для управления сценами — создания, изменения, удаления, упорядочивания — служит панель **Scene**. Чтобы вывести ее на экран, выберите пункт

Scene в меню **Window** или нажмите комбинацию клавиш <Shift>+<F2>. Вы также можете выбрать пункт **Scene** в меню **Modify** или пункт **Scenes** в контекстном меню пустого пространства рабочего листа. Сама панель показана на рис. 13.19.

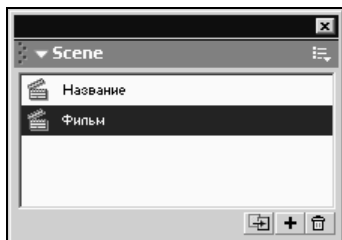


Рис. 13.19. Панель свойств **Scene**

Большую часть этого окна занимает список уже созданных сцен. Вы можете выделить, или сделать текущей, любую сцену, щелкнув по ней мышью. Выделив сцену, вы сможете просмотреть ее содержимое в окне документа: на временной шкале будут показаны все кадры, входящие в эту сцену, а на рабочем листе — содержимое текущего кадра. Учтите, что в окне документа показывается содержимое только текущей сцены. Чтобы просмотреть остальные сцены, вам нужно будет вызвать на экран панель **Scene**. Название текущей сцены показывается в инструментарии, расположенном выше рабочего листа, в его левой части, правее кнопки возврата (рис. 13.20).



Рис. 13.20. Название текущей сцены показано выше и левее рабочего листа

В правой части того же инструментария находится кнопка **Edit Scene**, при нажатии которой на экране появляется меню сцен (рис. 13.21). Чтобы просмотреть какую-либо сцену, просто выберите нужный пункт этого меню.

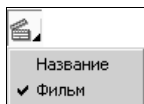


Рис. 13.21. Кнопка **Edit Scene** и раскрытое меню сцен

Есть еще один способ выбора нужной сцены. В меню **View** находится подменю **Goto**, пунктами которого вы можете воспользоваться для "путешествия" по сценам. В нижней части этого подменю находятся пункты, соответствующие

всем созданным в фильме сценам. Также вы можете воспользоваться следующими пунктами, расположенными в верхней части подменю **Goto**:

- ❑ **First** (этому пункту соответствует клавиша <Home>) — переход к самой первой сцене в списке;
- ❑ **Previous** (клавиша <PgUp>) — переход к предыдущей сцене;
- ❑ **Next** (клавиша <PgDn>) — переход к следующей сцене;
- ❑ **Last** (клавиша <End>) — переход к последней сцене в списке.

Очень странно: хоть для этих пунктов меню и заявлены "горячие" клавиши, но они почему-то не всегда работают. Что это? Очередная ошибка?..

Но вернемся к панели **Scene**.

Вы можете перетаскивать сцены в списке мышью, изменяя порядок их воспроизведения (сцены воспроизводятся в порядке сверху вниз). Также можно переименовывать сцены, чтобы дать им понятные имена. Для этого дважды щелкните мышью по имени нужной сцены. Вместо этого имени появится небольшое поле ввода, введите в него новое имя и нажмите клавишу <Enter> для его сохранения или клавишу <Esc> для отказа от него.

Чтобы добавить новую пустую сцену, нажмите кнопку **Add Scene** (рис. 13.22), расположенную в нижней части панели **Scene**. Вы также можете выбрать пункт **Scene** в меню **Insert**. Новая сцена появится в списке позицией ниже сцены, выделенной пользователем, и сама станет выделенной. Так что вы можете начать наполнять ее полезным содержимым — кадрами — сразу же после создания.



Рис. 13.22. Кнопка **Add Scene**

Есть еще один способ создания новых сцен — дублирование существующих. Для этого служит кнопка **Duplicate Scene** (рис. 13.23), расположенная также в нижней части панели **Scene**. Выделите в списке сцену, которую нужно продублировать, и нажмите эту кнопку. Новая сцена появится позицией ниже и получит имя вида "<Изначальная сцена> copy".



Рис. 13.23. Кнопка **Duplicate Scene**

Для удаления ненужной сцены вы должны выделить ее и нажать кнопку **Delete Scene** (рис. 13.24) или выбрать пункт **Remove Scene** в меню **Insert**.

Flash выдаст предупреждение, говорящее о том, что сцена удаляется безвозвратно. Нажмите кнопку **ОК**, чтобы удалить сцену, и **Cancel**, если вы еще не набрались решимости. Чтобы сразу же удалить ненужную сцену без вывода этого предупреждения, щелкните кнопку **Delete Scene**, удерживая нажатой клавишу <Ctrl>.



Рис. 13.24. Кнопка **Delete Scene**

Когда вы будете просматривать готовый фильм в среде Flash (выбрав пункт **Play** в меню **Control** или нажав клавишу <Enter>), имейте в виду, что по умолчанию показывается только содержимое текущей сцены. Чтобы просмотреть весь фильм (в смысле, все его сцены), включите пункт-выключатель **Play All Scenes** в меню **Control**. При просмотре фильма в отдельном окне (был выбран пункт **Test Movie** в меню **Control** или нажата комбинация клавиш <Ctrl>+<Enter>) всегда показываются все сцены фильма. Если же вы хотите просмотреть в отдельном окне только текущую сцену, выберите пункт **Test Scene** в меню **Control** или нажмите комбинацию клавиш <Ctrl>+<Alt>+<Enter>.

Дополнительные возможности Проводника Flash

В *главе 10* вы познакомились с весьма полезным инструментом под названием Проводник Flash. Вы узнали, что он поддерживает некоторые элементы фильма Flash, в частности, текстовые блоки и экземпляры. Но, кроме того, он может отображать также и все кадры, входящие в текущую сцену, и сами сцены, входящие в фильм.

Чтобы просмотреть сцены и кадры, включите кнопку-выключатель **Show Frames and Layers**, находящуюся в окне Проводника. (Подробнее о Проводнике и его возможностях см. *главу 10*.) Также проверьте, включен ли флажок **Frames** в окне **Movie Explorer Settings** (см. рис. 10.31). На рис. 13.25 показано окно Проводника, в котором в виде иерархического списка изображена текущая сцена фильма и все входящие в нее кадры.

Кроме того, вы можете просмотреть в окне Проводника содержимое всех входящих в фильм сцен. Для этого включите пункт-выключатель **Show All Scenes** в контекстном или дополнительном меню Проводника. После этого Проводник покажет вам все сцены фильма (рис. 13.26).

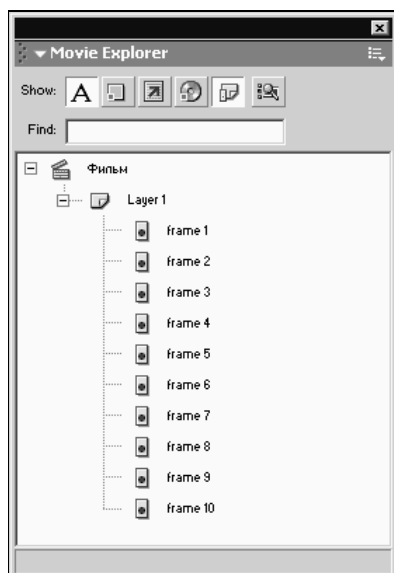


Рис. 13.25. Окно Проводника, отображающее текущую сцену фильма и список входящих в нее кадров

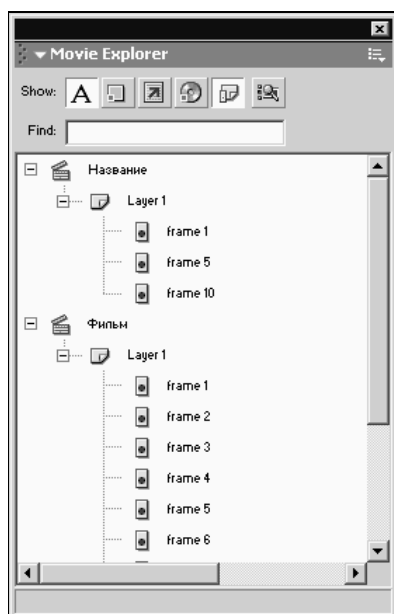
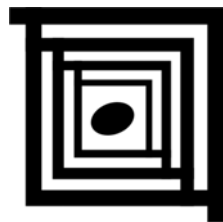


Рис. 13.26. Окно Проводника, отображающее все сцены фильма и их содержимое



Глава 14

Трансформационная анимация

Как вы узнали из *главы 13*, покадровая анимация создается очень просто. Нужно только добавить достаточное количество кадров в последовательность, а дальше — рисуй себе их содержимое, используя средства, изученные в *части 2*. Только вот беда: рисовать придется очень долго...

Но, если вы не хотите рисовать слишком долго, вам на помощь придут средства Flash по созданию трансформационной анимации. Если Flash используется для подготовки Web-графики (элементы оформления Web-страниц, анимированные рекламные баннеры, небольшие фильмы), то зачастую вся анимация ограничивается простейшими движениями какого-либо элемента. А такие движения элементарно реализуются с помощью трансформационной анимации (так называемая трансформация движения). Вам нужно будет только нарисовать два ключевых кадра последовательно — начальный и конечный — и сделать несколько щелчков мышью. Конечно, если ваша анимация сложнее, чем простое движение "из точки А в точку В", вам придется также создать дополнительные ключевые кадры, во всех точках, где движение элемента изменяется.

Более того, с помощью Flash вы можете заставить какой-либо элемент менять свой цвет или прозрачность, последнее нужно, если вы хотите сделать элемент постепенно появляющимся или исчезающим. А еще средствами Flash можно реализовать *морфинг* — плавное изменение формы любого элемента. Этот вид анимации называется трансформацией формы.

Так что средства трансформационной анимации Flash не так уж и бедны, как вы могли подумать, читая *главу 12*, где подробно описывались оба вида анимации.

Кроме меньшей трудоемкости, трансформационная анимация имеет еще одно достоинство перед покадровой: она занимает меньше места. В самом деле, вместо того, чтобы хранить все кадры фильма, программа записывает в файл только несколько чисел — параметры функции, реализующей эту анимацию. Сравните объем десятка изображений, пусть даже и векторных, и объем, занимаемый этими числами. Даем гарантию — это сравнение будет отнюдь не в пользу покадровой анимации.

Основным недостатком трансформационной анимации является ее бедность. Хотя Flash и предлагает нам достаточно мощные средства "оживить" изображения "малой кровью", красивые и сложные мультфильмы вы из одних трансформаций не сделаете. Если вы мечтаете о карьере в художественной анимации, то — увы!.. — готовьтесь опять же рисовать бесчисленные кадры, один за другим, один за другим...

Но не будем о грустном. Тем более что у нас нет времени грустить. Займемся трансформационной анимацией и посмотрим, что все-таки с ее помощью можно сделать. Flash, как всегда, нам поможет.

Трансформация движения

Трансформацией движения называется такой вид трансформации графического элемента, когда он меняет местоположение, внешний вид (то есть, искажается с помощью стандартных средств Flash: масштабирования, сдвига и вращения) или цвет. Этот вид трансформационной анимации наиболее прост в реализации, поэтому мы с него и начнем.

Прежде всего, нужно упомянуть два момента. Начинаящие пользователи Flash часто о них забывают.

Во-первых, анимировать с помощью трансформации можно не все графические элементы, а только экземпляры, группы и текстовые блоки. (Об образцах и экземплярах см. главу 10, о текстовых блоках — главу 7, о группах — главу 5.) Одним словом, то, что отображается в Проводнике Flash. При этом полноценно трансформация движения (в частности, изменение цвета) поддерживается только для экземпляров. Если же вы хотите анимировать элемент, не являющийся ни экземпляром, ни группой, ни текстовым блоком, то либо сделайте его экземпляром или группой, либо все-таки создайте покадровую анимацию. (Есть еще и третий способ — создать трансформацию формы — но о нем см. ниже.)

Во-вторых, вы можете анимировать только один-единственный элемент на текущем слое. Поэтому, если вы хотите создать несколько анимированных элементов, вам нужно разместить их в разных слоях. (О слоях см. главу 15.)

Создание простейшей трансформации движения

Давайте же создадим нашу первую трансформационную анимацию. Для примера возьмем фильм, демонстрирующий процесс деления клетки. Этот фильм мы создали, изучая покадровую анимацию, в главе 13. Надеемся, вы его сохранили.

Наш фильм будет состоять из двух сцен. Первая сцена содержит название фильма, отображаемое в обычном текстовом блоке, и носит "говорящее" на-

звание Название. Вторая сцена содержит собственно покадровую анимацию, показывающую процесс деления клетки и называется Фильм. Трансформационная анимация будет использована в первой сцене для создания "выезжающего" сверху заголовка. (Как вы помните, трансформация движения применима к текстовым блокам.) Отведем для нее пять кадров — вполне хватит, если учесть, что частота кадров нашего фильма невысока.

Итак, пора за дело. Откройте файл созданного нами учебного фильма в среде Flash и создайте в нем новую пустую сцену (Название). Переместите ее в начало списка: название фильма должно появляться в его начале, не так ли? Создайте текстовый блок, поместите его в центре рабочего листа и напишите в нем Деление клетки. Теперь все готово к созданию трансформации движения.

Собственно, существует два способа создания трансформации движения. Мы рассмотрим их оба. И начнем со способа, не самого простого по выполнению, но, на наш взгляд, позволяющего контролировать весь процесс создания анимации. Он заключается в том, что сначала создаются два ключевых кадра, а уже потом — сама анимация.

Выберите первый кадр анимации и переместите название нашего фильма вверх, пока оно не выйдет за границы листа. Вспомните: то, что находится за границами листа, не показывается при просмотре готового файла Shockwave/Flash, однако, существует в реальности. Этим можно воспользоваться, чтобы сделать какой-либо элемент невидимым до поры до времени, а потом мгновенно или постепенно вывести его на всеобщее обозрение. Так же поступим и мы: сначала название нашего фильма будет находиться за пределами рабочего листа, а потом плавно "выползет" на его середину.

Воспользуйтесь панелью **Align** (см. рис. 9.1), чтобы выровнять название точно по середине листа. Для этого выберите пункт **Align** меню **Window** или нажмите комбинацию клавиш <Ctrl>+<K>. Не забудьте только включить кнопку-выключатель **To Stage**, чтобы Flash выровнял название фильма относительно листа, а не самого себя. И в дальнейшем всегда держите эту панель под рукой — она нужна практически всегда. (Подробнее о панели **Align** и выравнивании графических фрагментов можно прочитать в *главе 9*.)

Итак, первый ключевой кадр готов. Сохраните подготовленный к созданию анимации документ (назовем его просто "подготовленным") под другим именем. Это нужно для того, чтобы мы смогли создать ту же самую анимацию вторым способом, не выполняя описанные выше подготовительные операции.

Теперь создадим второй ключевой кадр.

Выделите на временной шкале пустое место, где должно закончиться движение нашего названия. Пусть это будет кадр № 5, как мы договорились ранее. Создайте новый ключевой кадр, выбрав пункт **Keyframe** в меню **Insert** или выбрав пункт **Insert Keyframe** в контекстном меню выделенного пустого

места на временной шкале. Первый ключевой кадр тотчас "растянется", чтобы заполнить все свободное пространство от начала последовательности до второго ключевого кадра. Как вы уже знаете, это обычное поведение Flash в такого рода случаях.

Создав второй ключевой кадр, выделите его и переместите название фильма на его законное место — в середину листа. Воспользуйтесь панелью **Align**, чтобы сделать это.

Оба ключевых кадра готовы.

Настало время создать саму анимацию.

Снова выделите первый, растянутый, ключевой кадр. Его-то мы и преобразуем в анимацию. Когда вы выделите кадр, Flash автоматически выделит все, что находится на рабочем листе, в данном случае — текстовый блок названия. После этого взгляните на редактор свойств. Его вид при выделенном ключевом кадре показан на рис. 14.1.

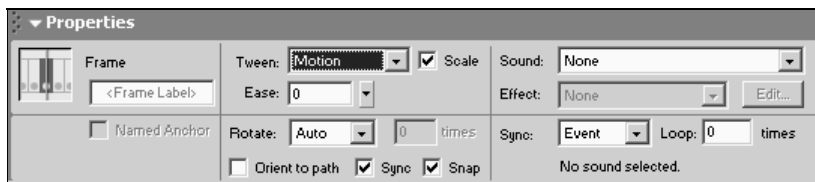


Рис. 14.1. Вид редактора свойств при выделенном ключевом кадре (задана трансформация движения)

Изначально в раскрывающемся списке **Tween** редактора свойств выбран пункт **None**. Это значит, что для выделенного на рабочем листе элемента не задана никакая трансформация. Чтобы задать трансформацию, выберите пункт **Motion**. В редакторе свойств тотчас появится множество других элементов управления, но мы пока не будем их трогать. (Все эти элементы управления мы рассмотрим далее в этой главе.)

Как вы поняли, мы задаем трансформационную анимацию сразу для всего содержимого рабочего листа. Вот поэтому на одном слое листа может находиться только один анимированный элемент.

На этом все. Мы только что создали трансформацию движения. Теперь щелкните мышью по рабочему листу или временной шкале, чтобы убрать фокус ввода с редактора свойств. И нажмите клавишу <Enter>, чтобы просмотреть полученную анимацию. Она работает!

Если вы теперь посмотрите на временную шкалу, то, вероятно, удивитесь, как Flash отображает первый, растянутый ключевой кадр (рис. 14.2). Он стал синим, более того, его подчеркивает тонкая черная стрелка, направленная слева направо. Таким образом и обозначается трансформация движения.



Рис. 14.2. Трансформация движения

Первый способ создания трансформации движения мы рассмотрели. Теперь рассмотрим второй. Он быстрее первого и отличается также и тем, что Flash сам делает за художника кое-какую работу. Поэтому второй способ проще и хорошо подходит для начинающих художников-аниматоров, либо для быстрого создания простых анимаций. Однако первый способ предоставляет пользователю кое-какие дополнительные возможности, которые могут оказаться полезными.

Откройте сохраненный ранее "подготовленный" фильм. Первый ключевой кадр уже готов, так что не будем описывать, как он создается, а сразу перейдем к созданию анимации. В дальнейшем, когда будете работать с Flash, почаще сохраняйте промежуточные варианты: вдруг к ним придется вернуться.

В прошлый раз после создания первого ключевого кадра мы сразу же создали второй. Сейчас же порядок действий будет немного другой. Выберите в меню **Insert** пункт **Create Motion Tween**. Вы также можете найти этот пункт в контекстном меню выделенного ключевого кадра. После этого Flash создаст так называемую "бесконечную" анимацию. "Бесконечная" — не в смысле заикленная, просто такая анимация не имеет конечного ключевого кадра. На временной шкале она показана так, как на рис. 14.3 (первый ключевой кадр специально "растянут"). Обратите внимание, что она выделяется штриховой линией, а не стрелкой. Вы можете увидеть такую же картину, если удалите второй ключевой кадр какой-либо анимации.

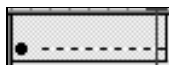


Рис. 14.3. "Бесконечная" анимация

После этого мы задаем второй ключевой кадр нашей анимации. Для этого выделяем на временной линии пустое место, где закончится наша анимация (в нашем случае это кадр № 5), и помещаем туда новый ключевой кадр. Для этого, как вы помните, нужно выбрать пункт **Keyframe** меню **Insert** или пункт **Insert Keyframe** контекстного меню выделенного пустого места на временной шкале. При этом первый ключевой кадр анимации "растягивается", чтобы заполнить все образовавшееся свободное пространство. Тогда-то вы и увидите то, что показано на рис. 14.2.

Осталось задать новое местоположение названия фильма. Как это сделать, вы уже знаете. Да там, собственно, нечего и знать — просто переместите название на новое место и, если нужно, выровняйте. Анимация уже создана самим Flash, так что сразу же можете проверить, что у вас получилось.

Вот и все! Как видите, второй способ создания анимации много проще первого. Кроме того, он примечателен тем, что Flash делает за вас еще одну работу. Так, если вы собираетесь анимировать графический фрагмент, не являющийся экземпляром (даже если это будет группа или текстовый блок), Flash сам преобразует его в образец и даст ему имя вида "tween<номер>". Далее на основе этого образца будет создан экземпляр, который и заменит выделенный вами на листе графический фрагмент. Так Flash исправляет за начинающими аниматорами одну из их характерных ошибок.

В нашем случае Flash остался верен себе. Он превратил текстовый блок названия в образец под именем `tween1` и поместил экземпляр этого образца на место самого названия. Вы, конечно, можете превратить экземпляр названия обратно в текстовый блок, выделив его и выбрав в меню **Modify** пункт **Break Apart** или нажав комбинацию клавиш `<Ctrl>+`. (Имейте только в виду, что сделать это нужно как в первом, так и во втором ключевых кадрах. И лучше сразу же удалить образец `tween1` из библиотеки.) Но мы вам не советуем это делать. Почему — узнаете позже.

Еще раз отметим, что первый способ предоставляет намного больше возможностей по управлению анимацией, чем второй. Посмотрите на рис. 14.1 — сколько элементов управления содержит редактор свойств. Хотя, создав анимацию вторым способом, вы всегда сможете потом обратиться к редактору свойств и подкорректировать все, что хотите.

Итак, простейшая анимация у нас готова. Давайте ее немного усложним.

Более сложная трансформация движения

Flash-аниматоры как начинающие, так и опытные, очень любят такой прием: какой-либо графический фрагмент постепенно исчезает или, наоборот, появляется на экране. Это похоже на то, как на фотографии, опущенной в раствор проявителя, постепенно возникает изображение. Сделаем так и мы. Пусть название фильма, закончив движение, постепенно пропадет с экрана.

Для создания такого эффекта также используется трансформационная анимация, а именно, трансформация движения. Как вы помните, при трансформации движения элемент может не только двигаться или искажаться, но и менять свой цвет. Как раз под изменение цвета и попадает наше исчезновение, в данном случае мы будем менять прозрачность названия нашего фильма.

Вот здесь-то нам и пригодится образец `tween1`, точнее, созданные на его основе экземпляры. Ведь выполнять анимированное изменение цвета можно только у экземпляров.

А теперь отвлечемся от рабочего листа и посмотрим на временную шкалу с созданной ранее анимационной последовательностью. В данном случае мы имеем всего одну последовательность, которая реализует движение названия

фильма сверху вниз. Чтобы создать другую анимацию, реализующую постепенное пропадание названия на экране, нам нужно добавить к первой последовательности вторую "встык", чтобы первая последовательность сразу же переходила во вторую. А раз так, то второй ключевой кадр первой последовательности станет первым ключевым кадром второй.

Для создания новой анимации выберем второй способ — он быстрее. А в данном случае нам нужна именно скорость и совсем не обязательно обилие настроек. Нам нужно понять, как создается сложная анимация.

Выделим второй ключевой кадр первой анимации и выберем в меню **Insert** пункт **Create Motion Tween**. Это нам уже знакомо. Теперь вставим в пустое место под № 10 новый ключевой кадр. Результат наших трудов показан на рис. 14.4. Все — анимация почти готова.



Рис. 14.4. Две анимационные последовательности

Отлично! Мы уже научились работать быстро. Благо Flash максимально идет нам навстречу.

Теперь выберем пункт **Alpha** в раскрывающемся списке, расположенном в правом верхнем углу редактора свойств. В поле ввода с регулятором, расположенном правее этого списка, вводим значение 0% — полная прозрачность. Все, название нашего фильма теперь стало невидимым.

Проверим то, что у нас получилось. Ура, работает!

Вы можете создать любую достаточно сложную анимацию, просто добавляя во временную шкалу последовательности кадров, одну за другой. Ваше название может "гулять" по всему экрану, исчезать и снова появляться, менять цвета, искажаться и восстанавливать свой первоначальный вид. Попробуйте поэкспериментировать — это не только позволит вам приобрести практику, но и просто забавно.

Но сначала все-таки стоит прочесть о трансформации движения еще кое-что.

Параметры трансформации движения

А теперь настала пора рассмотреть поближе редактор свойств. И все элементы управления, доступные на нем, если выбран ключевой кадр анимации. Посмотрите на рис. 14.1 — там показаны они все.

В верхнем левом углу редактора свойств, ниже надписи **Frame** находится поле ввода **Frame Label**. С его помощью вы можете задать уникальное имя или примечание для какого-либо кадра. *Имя кадра* будет полезно только для программирования сценариев и некоторых других возможностей, хотя вы

можете воспользоваться этим, чтобы как-то пометить этот кадр. Если вы присвоите кадру имя, он будет выглядеть так, как показано на рис. 14.5.

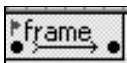


Рис. 14.5. Кадр, имеющий имя frame

Примечания отличаются от имен тем, что не экспортируются в окончательный файл Shockwave/Flash. Поэтому примечание, в отличие от имени, может быть достаточно развернутым. Примечание также вводится в поле ввода **Frame Label**, но его должны предварять два символа /. Кадр, содержащий примечания, выглядит так, как показано на рис. 14.6.



Рис. 14.6. Кадр, содержащий примечание frame

Флажок **Named Anchor** позволяет превратить кадр, имеющий имя, в *именованный "якорь"* (рис. 14.7). Так называется кадр, являющийся своего рода "зацепкой" для проигрывателя Flash. Если фильм Flash проигрывается в окне Web-обозревателя, вы можете пользоваться кнопками **Forward (Вперед)** и **Back (Назад)**, чтобы "прыгать" от одного именованного "якоря" к другому. Если кадр не имеет имени (в поле ввода **Frame Label** ничего не введено), флажок **Named Anchor** недоступен.



Рис. 14.7. Именованный "якорь", имеющий имя frame

Назначение раскрывающегося списка **Tween** вы уже знаете. Поэтому перейдем к рассмотрению остальных элементов управления, расположенных в редакторе свойств.

Флажок **Scale** нужно включить, если вы применяли масштабирование к анимированному элементу. Впрочем, по умолчанию он всегда включен. Если же его отключить, элемент в любом случае не будет масштабироваться.

С помощью поля ввода с регулятором **Easing** вы можете задать степень замедления движения анимированного элемента в конце анимации. Если вы введете в это поле значение от 1 до 100, движение элемента начнется быстро, а потом замедлится. Если же, наоборот, вы введете отрицательное значение от -1 до -100, элемент начнет двигаться медленно, а потом его движение ускорится. Если вы хотите, чтобы элемент всегда двигался равномерно, оставьте в этом поле ввода значение по умолчанию — 0.

Раскрывающийся список **Rotate** задает направление вращения анимированного элемента, если таковое было к нему применено. Всего в этом списке четыре пункта:

- ☐ **None** — элемент не будет вращаться вообще, даже если для него было задано вращение;
- ☐ **Auto** — элемент будет вращаться в том направлении, в котором был задан поворот (значение по умолчанию);
- ☐ **CW** — элемент всегда будет вращаться в направлении по часовой стрелке;
- ☐ **CCW** — элемент всегда будет вращаться в направлении против часовой стрелки.

Если в списке **Rotate** были выбраны пункты **CW** или **CCW**, то становится доступным поле ввода, расположенное правее этого списка. В нем задается количество дополнительных поворотов, которые сделает анимированный элемент. Значение по умолчанию — 0, т. е. элемент сразу повернется на заданный угол, не делая никаких дополнительных поворотов.

Если в редакторе свойств вы видите кнопку, показанную на рис. 14.8, это значит, что с вашей анимацией что-то не в порядке. При нажатии этой кнопки на экране появится окно предупреждения, описывающее проблему, с которой столкнулся Flash при создании анимации.



Рис. 14.8. Кнопка выдачи предупреждения о проблеме, связанной с созданием анимации

Остальные элементы управления мы опишем в последующих главах этой книги.

Трансформация формы

Трансформация формы — это плавное изменение формы графического фрагмента, иначе говоря, его морфинг. Под изменением формы при этом следует понимать не только изменение самой формы фрагмента, но и его местоположения, цвета и т. п. Также анимируемый фрагмент может разделяться на части, и, наоборот, несколько фрагментов могут сливаться, образуя единое целое. В этом смысле, трансформация формы дает аниматору большие возможности, чем трансформация движения.

Как вы помните, трансформация движения может применяться только для экземпляров, групп и текстовых блоков (для последних двух — с некоторыми ограничениями). Трансформация формы, наоборот, может быть применена только к графическим фрагментам, не являющимся ни тем, ни другим,

ни третьим. Поэтому, если вы хотите применить такой вид трансформации к какому-либо фрагменту, то сначала удостоверьтесь, что он является "обычной" графикой. Чтобы преобразовать графический фрагмент в "обычную" графику, выделите его и выберите пункт **Break Apart** в меню **Modify**, в контекстном меню выделенного графического фрагмента или нажмите комбинацию клавиш <Ctrl>+.

В отличие от трансформации движения, применимой только к одному элементу в слое, трансформация может применяться сразу к нескольким графическим фрагментам. Следите только, чтобы все они располагались в одном слое. (О слоях см. главу 15.)

Как создается трансформация формы? Так же, как трансформация движения. Вы задаете ключевые кадры, где анимация начинается и кончается, "пристыковываете" разные анимационные последовательности друг к другу, а остальное, как говорится, дело техники.

Пожалуй, трансформация формы — неплохая альтернатива покадровой анимации. Конечно, если эта анимация достаточно проста.

Создание простейшей трансформации формы

Создание трансформации формы мы будем изучать на другом примере. Давайте создадим новый документ Flash и, воспользовавшись инструментом "карандаш", нарисуем на рабочем листе какую-нибудь кляксу посложнее. Например, такую, похожую на собаку, как на рис. 14.9. Не забываем также создать заливку, для чего воспользуемся "ведром с краской". (Об инструментах рисования см. главу 5.)

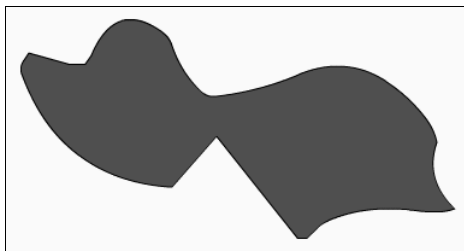


Рис. 14.9. Первый ключевой кадр трансформации формы

Итак, первый ключевой кадр у нас готов. Теперь нужно создать второй ключевой кадр и — самое главное — дать понять Flash, что мы от него хотим.

Как создать второй ключевой кадр, вы уже знаете. Просто создайте новый ключевой кадр на том делении временной шкалы, где создаваемая вами анимация должна закончиться. Flash автоматически скопирует во второй ключевой кадр содержимое первого. А теперь делайте с ним все, что душе

угодно. Например, можете отделить контур от заливки и вдоволь поворачивать и искажать их. Вы даже можете удалить какие-то фрагменты графики и добавить новые. В результате всех этих действий у вас получится то, что показано на рис. 14.10.

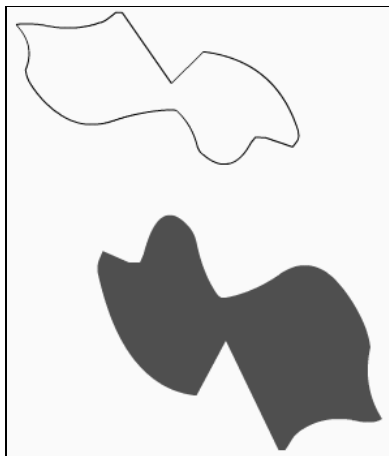


Рис. 14.10. Второй ключевой кадр трансформации формы

Чтобы создать трансформацию формы, нам опять же понадобится редактор свойств. Выделите первый ключевой кадр будущей анимации и выберите в раскрывающемся списке **Tween** редактора свойств пункт **Shape**. Созданная таким образом анимация появится на временной шкале (рис. 14.11). Как и трансформация движения, она отображается в виде тонкой черной стрелки, но не на синем, а на зеленом фоне.

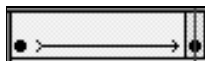


Рис. 14.11. Трансформация формы

Остается проверить, что мы сделали. Нажмите клавишу <Enter>, предварительно убрав фокус ввода с редактора свойств. Хм, в самом деле, интересно! Нарисованная нами фигура разделяется на части, которые, совершив несколько кульбитов, "успокаиваются" на экране отдельно друг от друга. Собака, чтобы расти, сбрасывает кожу...

Вы также можете заставить нашу "собаку" менять цвет. Для этого выберите последний (или первый) ключевой кадр, выделите на рабочем листе заливку и измените ее цвет. Вы также можете изменить цвет контура. Теперь "перемотайте" анимацию в начало и посмотрите, что получится.

Выше мы сказали, что трансформация формы, возможно, лучшая альтернатива покадровой анимации. Проверим это на практике. Выясним, получится

ли у нас процесс деления клетки, который мы худо-бедно воспроизвели ранее, в *главе 13*. Создадим новый документ Flash и попробуем создать очередную версию фильма о делении клетки исключительно средствами трансформационной анимации. Точнее, не просто новую версию, а как это называется у киношников? римейк.

Итак, рисуем первый ключевой кадр. Как он выглядит, мы знаем. Чтобы нарисовать второй ключевой кадр, воспользуемся одним из трюков Flash, позволяющим разделять изображение на части. (Собственно, мы уже им пользовались.) Сначала выделяем левую часть нашей "клетки" и относим ее влево, потом выделяем оставшуюся, правую, часть и относим вправо. Осталось только придать обоим половинам "клетки" вид их "родителя" — и второй ключевой кадр готов.

Теперь создаем саму анимацию и проверяем результат. Вот так штука! Наша "клетка" просто разваливается пополам, а вовсе не делится, как это делают настоящие, живые клетки. Да, Flash не так уж и интеллектуален, как мы о нем думали...

Но что же делать? Есть два пути решения проблемы.

Путь первый: разбить наш фильм на две анимационные последовательности — это обычная практика при создании сложной анимации. Первая последовательность покажет, как "клетка" делится пополам, и отдельные ее половинки расходятся в разные стороны. Вторая же последовательность покажет, как уже разошедшиеся в стороны половинки нашей "клетки" принимают форму новых клеток. Однако это все равно будет не то: сначала Flash добросовестно покажет, как "клетка" "разрывается" пополам, а потом — как эти половинки скругляются. Но ведь реальные клетки делятся не так! Реально оба этих процесса — и "разрыв", и скругление — происходят одновременно. А этого мы и пытаемся добиться от Flash!

Поэтому ничего не остается, кроме как пойти по второму пути. А именно, вернуться к покадровой анимации. Как ни крути, но человеческих рук не заменишь никакой, даже самой умной, программой. Перефразируя известное высказывание, можно сказать: компьютеру — компьютерово, человеку — человеческое.

Параметры трансформации формы

Когда вы выбираете в раскрывающемся списке **Tween** редактора свойств пункт **Shape**, ниже появляется группа элементов управления (рис. 14.12). Давайте выясним, зачем они нужны.

В поле ввода **Frame Label** задается уникальное имя или примечание для какого-либо кадра. Флажок **Named Anchor** превращает именованный кадр в "якорь". Эти элементы управления вам уже знакомы.

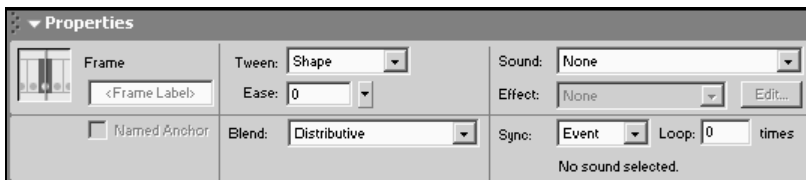


Рис. 14.12. Вид редактора свойств при выделенном ключевом кадре (задана трансформация формы)

С помощью поля ввода с регулятором **Easing** вы можете задать степень замедления изменения формы графического фрагмента в конце анимации. Если вы введете в это поле значение от 1 до 100, фрагмент начнет менять форму быстро, а потом это изменение замедлится. Если же, наоборот, вы введете значение от -1 до -100 , изменение формы фрагмента сначала будет медленным, а потом ускорится. Если вы хотите, чтобы форма фрагмента всегда изменялась с одинаковой скоростью, оставьте в этом поле ввода значение по умолчанию, равное нулю.

Раскрывающийся список **Blend** позволяет сделать формы, расположенные на промежуточных, т. е. сформированных самим Flash, кадрах анимации, более или менее угловатыми. Имейте, однако, в виду, что заданные с его помощью настройки справедливы только тогда, когда графический фрагмент, чья форма изменяется, имеет углы и прямые линии. Раскрывающийся список **Blend** имеет два пункта:

- ☐ **Distributive** — промежуточные формы будут по возможности сглаженными (значение по умолчанию);
- ☐ **Angular** — промежуточные формы могут быть угловатыми, если исходный фрагмент содержал углы и прямые линии.

Маркеры трансформации и их использование

Иногда бывает нужно, чтобы какая-либо точка фрагмента при изменении его формы переместилась в строго определенное место, т. е. вы хотите сами определять, какая точка куда переместится, не отдавая это на откуп Flash, который зачастую обходится с вашей графикой весьма вольно. В этом случае вам очень помогут так называемые *маркеры трансформации*. С их помощью вы можете выделить некоторые нужные вам точки на контуре анимруемого фрагмента и задать место, куда каждая точка переместится в конце анимации. Такие маркеры помечаются маленькими латинскими буквами от *a* до *z*. Одновременно может быть задано до 26 маркеров включительно.

Если вы хотите использовать маркеры трансформации, создайте оба ключевых кадра и саму анимацию. Проверьте, все ли работает. Определите точки, которыми вы хотите управлять. Минимизируйте количество этих точек, помните, что их может быть только 26.

Чтобы создать маркер трансформации, выделите первый ключевой кадр анимационной последовательности и выберите пункт **Add Shape Hint** в подменю **Transform** меню **Modify** или нажмите комбинацию клавиш **<Ctrl>+<Shift>+<H>**. На рабочем листе появится небольшой красный кружок с латинской буквой в центре, если этот маркер — первый, который вы создали, то в его центре будет буква **a**. Если вы почему-то не видите никаких красных кружков с буквами в центре, включите пункт-выключатель **Show Shape Hints** в меню **View** или нажмите комбинацию клавиш **<Ctrl>+<Alt>+<H>**.

Итак, мы создали первый маркер трансформации. Но он не привязан ни к одной точке на контуре нашей фигуры. Давайте исправим это. "Захватите" его мышью и "приклейте" к нужной точке контура графического фрагмента. Полученный результат показан на рис. 14.13. Все, таким образом вы задали первую контролируемую вами точку контура.

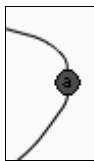


Рис. 14.13. Маркер трансформации

Затем выделите последний ключевой кадр анимационной последовательности. Вы увидите, что точно такой же красный кружок с буквой **a** находится где-либо на рабочем листе. Это парный маркер (маркеры трансформации всегда "ходят парами") соответственно на первом и последнем ключевых кадрах анимации. "Приклейте" его к контуру там, где в конце анимации должна оказаться заданная вами на первом ключевом кадре контрольная точка. Кружок (маркер) станет зеленым. Если же теперь вы вернетесь к первому ключевому кадру, то увидите, что его маркер стал желтым. Это значит, что контролируемая вами точка определена и на первом, и на последнем ключевых кадрах анимации.

На рис. 14.14 показан первый, а на рис. 14.15 — последний ключевые кадры нашей анимации. На первом ключевом кадре маркер трансформации должен быть желтым, а на последнем — зеленым. Проверьте, так ли это, если же это не так, поместите маркер на обоих кадрах точно на линию контура.

Запустите теперь созданную нами анимацию. Как видите, маркер помог — помеченная им точка оказалась как раз на нужном нам месте. Посмотрите на рис. 14.16 — на нем показана наша анимация в режиме отображения нескольких кадров. Для сравнения: на рис. 14.17 показана та же анимация, но созданная без маркера.

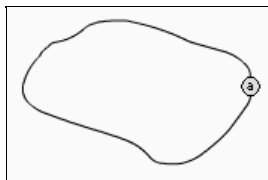


Рис. 14.14. Первый ключевой кадр анимации, сделанной с использованием маркера трансформации

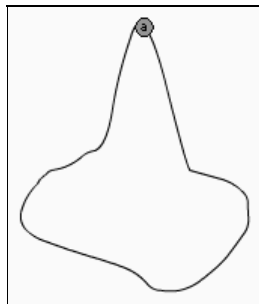


Рис. 14.15. Последний ключевой кадр анимации, сделанной с использованием маркера трансформации

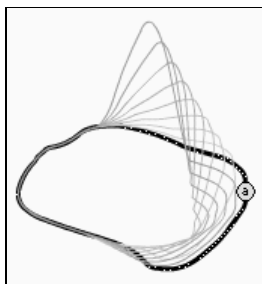


Рис. 14.16. Анимация, созданная с использованием маркера трансформации

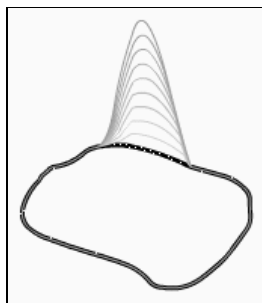


Рис. 14.17. Анимация, созданная без использования маркера трансформации

Вероятно, используя маркеры трансформации, можно попытаться создать римейк фильма "Деление клетки", используя только трансформационную анимацию, а именно, трансформацию формы. Не будем описывать в подробностях этот процесс — пусть это будет вашим домашним заданием. Если вы хотите действительно это сделать, то, скорее всего, вам придется задать несколько маркеров и поэкспериментировать с ними, помещая в разные точки контуров и внимательно следя за результатами. Получится, конечно, коряво, но хотя бы попробовать стоит.

Вы можете добавлять новые маркеры трансформации, выбрав пункт **Add Shape Hint** в подменю **Transform** меню **Modify** или нажав комбинацию клавиш <Ctrl>+<Shift>+<H>. Также вы можете щелкнуть по уже существующему маркеру правой кнопкой мыши и выбрать в контекстном меню пункт **Add Hint**. Очередные маркеры получают следующие буквы латинского алфавита.

Здесь стоит дать один совет. При добавлении новых маркеров старайтесь, чтобы в первом ключевом кадре они располагались в порядке по часовой стрелке. Впоследствии вам будет проще отследить, откуда и куда они перемещались.

Вы можете перемещать маркеры мышью, "приклеивая" их к разным местам контура. Это нужно хотя бы для того, чтобы пометить нужную точку на контуре фрагмента.

Вы можете удалять ненужные маркеры. Это может выполняться двумя различными способами, решайте сами, какой для вас удобнее. Во-первых, вы можете щелкнуть по нужному маркеру правой кнопкой мыши и выбрать в контекстном меню пункт **Remove Hint**. Во-вторых, вы можете просто "утащить" ненужный маркер прочь с рабочего листа.

Вы также можете быстро удалить все созданные вами маркеры трансформации. Для этого либо выберите пункт **Remove All Hints** в контекстном меню любого маркера, либо выберите одноименный пункт в подменю **Transform** меню **Modify**.

Вы можете временно скрыть с рабочего листа все маркеры трансформации. Для этого выключите пункт-выключатель **Show Shape Hints** в меню **View** или нажмете комбинацию клавиш <Ctrl>+<Alt>+<H>. Вы также можете отключить пункт-выключатель **Show Hints** в контекстном меню любого маркера. (Непонятно только, как его там снова включить. Еще одна загадка Flash...)

Вот и все о трансформационной анимации и обоих ее видах. Поговорим теперь, когда какой вид использовать.

Использование обоих видов трансформаций

Итак, подведем итоги. Снова кратко перечислим то, что мы знаем о различных видах трансформационной анимации. Это поможет нам очертить круг задач, решаемых каждым из них.

Трансформация движения:

- ☐ применяется к экземплярам, группам и текстовым блокам, причем к группам и текстовым блокам применимы не все ее возможности;
- ☐ позволяет изменять местоположение и размеры элемента, вращать и сдвигать, а также менять его цвет (это применимо только к экземплярам);
- ☐ позволяет анимировать только один элемент на слое.

Трансформация формы:

- ☐ применяется только к "обычной" графике, не являющейся ни экземпляром, ни группой, ни текстовым блоком;
- ☐ позволяет в достаточно широких пределах менять форму фрагмента графики (а также местоположение и цвет);
- ☐ позволяет анимировать сколько угодно элементов на слое.

Все ясно. Теперь давайте выясним, когда какую трансформацию лучше применять.

Очевидно, что трансформация движения замечательно подходит в тех случаях, когда вы используете экземпляры, группы и текстовые блоки. А экземпляры необходимы тогда, когда ваша графика содержит множество одинаковых фрагментов, в этом случае вместо того, чтобы рисовать их по отдельности, проще создать единственный библиотечный образец. Тогда размер результирующего файла Shockwave/Flash значительно сократится. (Почему это так, вы узнали в *главе 10*, где об образцах и библиотеках говорилось более подробно.) А если в графике активно используются экземпляры, то сам Бог велел применить трансформацию движения. И уж, конечно, нет иного, кроме трансформации движения, способа заставить "жить" текстовый блок.

Если же ваша графика не содержит одинаковых фрагментов, вам нет нужды забивать документ Flash еще и библиотечными образцами. Это заметно увеличит размер результирующего файла Shockwave/Flash, но не даст вашей графике ничего полезного. Поэтому в случае "обычной" графики лучше всего применять трансформацию формы. Ну и, конечно, без нее не обойтись, если вам нужно выполнить морфинг какого-либо графического фрагмента.

Вложенная анимация

Всем хороша трансформационная анимация: требует немного места для хранения, легка в создании, очень просто корректируется. Но, к сожалению, возможности ее сильно ограничены. Особенно в этом плане показательна трансформация движения — с ее помощью вы можете создавать только самые простые движения.

Представим себе такую ситуацию. Вы хотите создать анимированный элемент — прямоугольник, который бы двигался по рабочему листу справа налево. Вы скажете, что это реализуется элементарно, и будете правы. Но что делать, если вы захотите, чтобы этот прямоугольник не только двигался по листу, но еще и вращался при этом?

Возможны три способа решения этой проблемы:

1. Обратиться к покадровой анимации. Там эта проблема стоять не будет: вы сможете нарисовать все, что угодно, и Flash корректно это проиграет. Но приготовьтесь к тому, что вам придется рисовать каждый кадр анимации.
2. Прибегнуть к трансформации формы. Правда, для особо сложных случаев это может не помочь.
3. Создать вложенную анимацию. Вот об этом мы сейчас и поговорим.

Flash предоставляет интересную возможность: вы можете анимировать образец библиотеки, превратив его в небольшой фильм. Впоследствии можно

создать экземпляр на основе этого анимированного образца и анимировать его. В этом случае не только сам экземпляр будет двигаться или иначе менять свой вид, но и параллельно будет проигрываться его собственная анимация. Благодаря этому вы можете создавать очень сложные анимации.

Все вышесказанное проиллюстрировано на рис. 14.18. Как видите, одна анимация как бы вкладывается внутрь другой. Такая анимация и называется *вложенной*. Также часто такую анимацию называют *многоуровневой*.

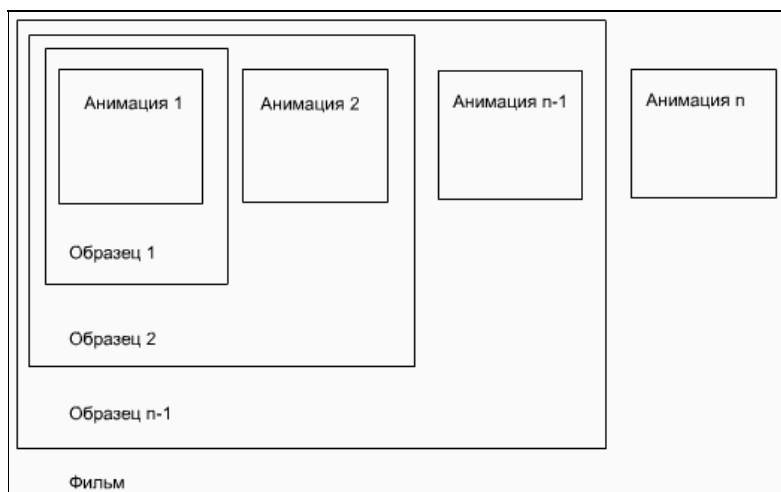


Рис. 14.18. Схема, иллюстрирующая принцип создания вложенной анимации

Давайте для примера создадим наш анимированный прямоугольник. Или, иначе говоря, создадим анимацию *первого уровня*.

Итак, сначала нам нужно создать сам образец. Для "вложения" анимации Flash предлагает нам создать либо обычный графический образец, содержащий анимацию, либо образец-клип. Если вы хотите просто "вложить" анимацию внутрь образца, не имея никаких далеко идущих целей, то создайте графический образец. Если же вы хотите управлять созданным экземпляром из сценария на ActionScript, вам придется использовать образец-клип. А поскольку мы еще не знакомы с программированием в среде Flash, то создадим хорошо нам знакомый графический образец.

Как создать новый образец Flash, вы уже знаете. Нарисуйте на рабочем листе прямоугольник и преобразуйте его в графический образец. Назовите его, скажем, *Square*.

Теперь нам нужно заставить его вращаться. Откройте полученный образец в режиме редактирования (неважно, каком). Выделите нарисованный нами прямоугольник целиком, т. е. и контур, и заливку. Сгруппируйте его, выбрав пункт **Group** в меню **Modify** или нажав комбинацию клавиш **<Ctrl>+<G>**. И давайте заставим его вращаться.

Создайте анимационную последовательность из десяти кадров (конечно, десять — это только пример, вы можете задать любое количество кадров). Как это делается, вы уже знаете, единственное исключение: раньше вы проделывали это для собственно документа Flash, а теперь — для образца. Выделите на временной шкале последний ключевой кадр, переключитесь на инструмент "трансформатор" (кнопка его включения была показана на рис. 9.8) и включите модификатор "вращение" (кнопка была показана на рис. 9.12). Поверните наш прямоугольник на 180°. Теперь можете проверить, что у вас получилось. А получится у вас должно нечто, показанное на рис. 14.19.

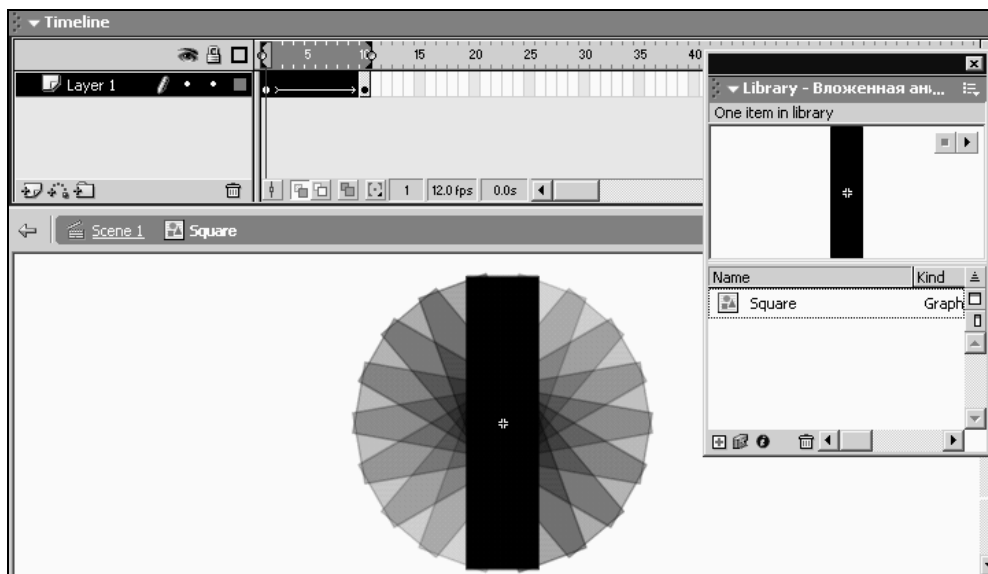


Рис. 14.19. Готовый анимированный образец

Вы можете создать анимированный образец и другим способом. Давайте последовательно опишем все шаги, которые вам следует для этого предпринять.

1. Создайте нужный графический элемент на рабочем листе и либо сгруппируйте его, либо преобразуйте в образец.
2. Создайте необходимую анимацию.
3. Скопируйте в буфер обмена нужные кадры этой анимации. Для этого выделите их на временной шкале и выберите пункт **Copy Frames** меню **Edit**.
4. Создайте новый пустой образец и откройте его в режиме правки, неважно, в каком именно.

5. Вставьте на временную шкалу скопированные вами ранее кадры. Для этого выделите пустое место под номером 1 и выберите пункт **Paste Frames** меню **Edit**.
6. Проверьте созданный образец.

Неважно, каким способом вы создали анимированный образец. Важно то, что вы его создали. Поэтому перейдем к следующему этапу нашей работы. Создадим анимацию *второго уровня*.

Как видите, в окне библиотеки виден новый созданный нами образец *Square*. То, что он содержит анимацию, видно из двух небольших кнопок, отображаемых Flash в верхнем правом углу панели предварительного просмотра этого окна. Правая кнопка запускает проигрывание анимации, а левая — останавливает его.

Теперь переключитесь в режим правки документа. Поместите в верхнем правом углу рабочего листа экземпляр образца *Square*. Создайте анимацию, перемещающую этот экземпляр из верхнего правого в нижний левый угол рабочего листа, продолжительностью как минимум вдвое большей, чем продолжительность анимации самого образца (в нашем случае — 20 кадров). И запустите готовую анимацию. У вас получится то, что показано на рис. 14.20.

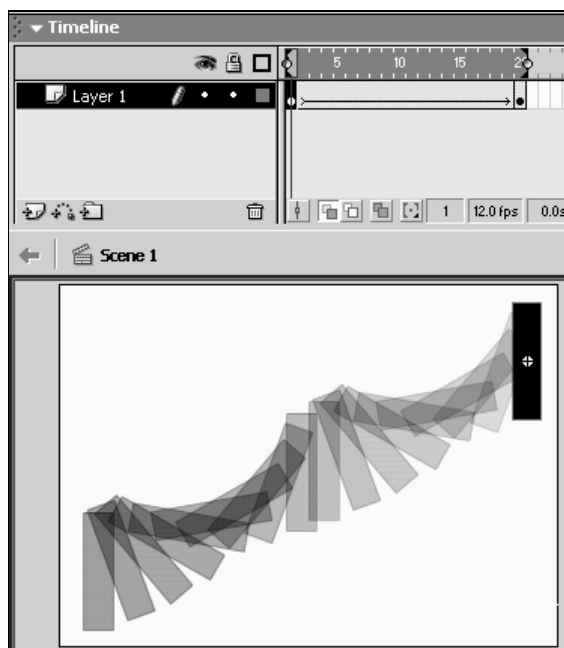


Рис. 14.20. Готовая двухуровневая анимация

А теперь давайте рассмотрим некоторые возможности, предлагаемые Flash для управления экземплярами, созданными на основе анимированных образцов. Выделите на рабочем листе наш анимированный экземпляр. И обратите внимание на редактор свойств (рис. 14.21). Нас интересует раскрывающийся список, расположенный правее кнопки **Swap**, и поле ввода **First**. Зачем они нужны?



Рис. 14.21. Вид редактора свойств при выбранном анимированном экземпляре

Раскрывающийся список позволяет задать, какая часть анимации образца, на основе которого был создан выделенный экземпляр, будет проиграна как часть общей вложенной анимации. Этот список содержит три пункта:

- ❑ **Loop** — анимация образца будет проигрываться бесконечно ("зацикленная" анимация). Причем проиграна она будет, начиная с кадра, номер которого введен в поле ввода **First**. То есть, общая анимация будет выглядеть, например, так, как на рис. 14.20. Этот пункт списка выбран по умолчанию;
- ❑ **Play Once** — анимация образца будет проиграна только один раз, после чего остановится. Причем проиграна она будет, начиная с кадра, номер которого введен в поле ввода **First**. Общая анимация в этом случае будет выглядеть, как показано на рис. 14.22;
- ❑ **Single Frame** — будет показан только один кадр анимации образца под номером, введенным в поле ввода **First**. Пример общей анимации вы можете увидеть на рис. 14.23.

Теперь рассмотрим такой случай. Предположим, что нам нужно изменить продолжительность нашей анимации, показанной на рис. 14.20, с двадцати, скажем, до пятнадцати кадров. Если мы просто изменим ее продолжительность, то получим результат, показанный на рис. 14.24.

В конце этой анимации, между четырнадцатым и пятнадцатым кадрами, наблюдается хорошо заметный скачок. Это происходит из-за того, что Flash сохраняет местоположение анимированного элемента в первом и последнем ключевых кадрах анимации. Такой подход оправдан в случае простого движения элемента, но в случае вложенной анимации он не срабатывает. В самом деле, Flash сохранил в последнем ключевом кадре одно положение анимированного элемента, а он на самом деле должен находиться совсем в другом. Что делать?

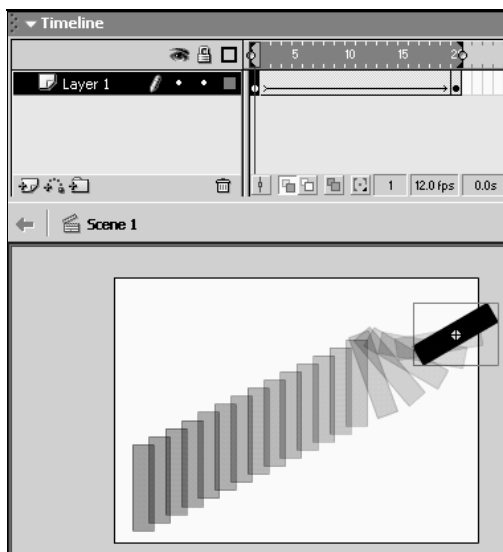


Рис. 14.22. Пример двухуровневой анимации (для анимированного экземпляра был выбран пункт **Play Once** раскрывающегося списка режима проигрывания, а в поле ввода **First** было введено значение 4)

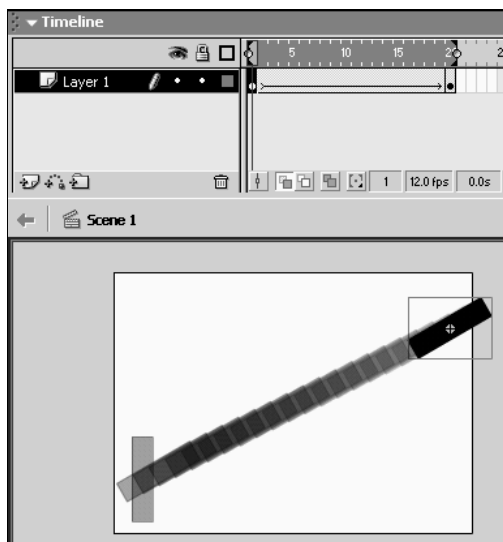


Рис. 14.23. Пример двухуровневой анимации (для анимированного экземпляра был выбран пункт **Single Frame** раскрывающегося списка режима проигрывания, а в поле ввода **First** было введено значение 4)

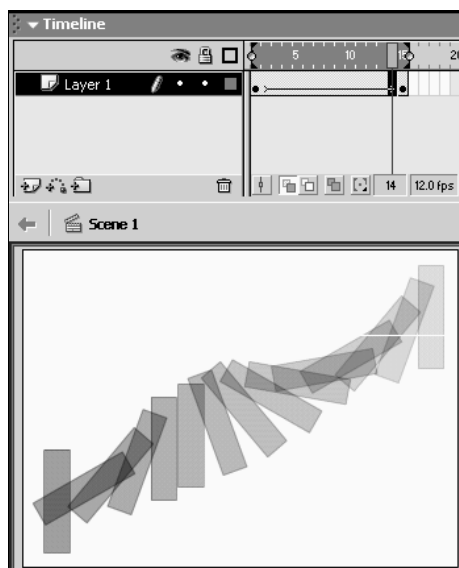


Рис. 14.24. Результат изменения продолжительности анимации, показанной на рис. 14.20

Выделите на временной шкале всю анимацию, которую нужно "исправить", — и первый, и второй ключевые кадры. И выберите пункт **Synchronize Symbols** в подменю **Frames** меню **Modify**. Вы также можете включить флажок **Sync** в редакторе свойств, возможно, для этого вам придется выбрать в раскрывающемся списке **Tween** пункт **Motion** или **Shape**, в зависимости от типа вашей анимации. После этого анимация станет плавной, без рывков, что и иллюстрирует рис. 14.25.

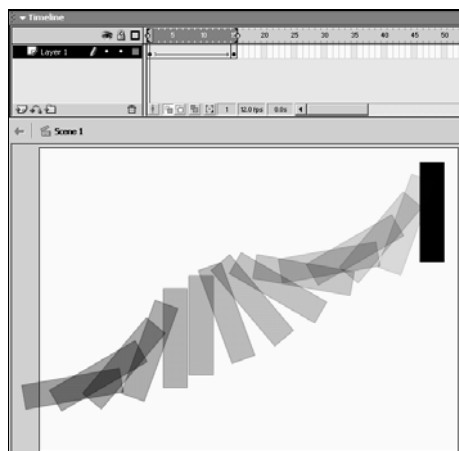
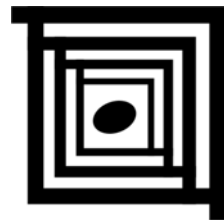


Рис. 14.25. Сглаженный вариант анимации, показанной на рис. 14.24



Глава 15

Слои

А сейчас мы ненадолго прервем свой рассказ об анимации. И поговорим о слоях. Поскольку без слоев более-менее сложной трансформационной анимации все равно не создашь. Да и при создании покадровой анимации слои могут стать серьезным подспорьем. Если, конечно, научиться ими пользоваться...

Слой (в терминологии Flash — *layer*) можно рассматривать как лист прозрачной пленки, лежащий на рабочем листе Flash. Вы можете рисовать на слое, используя изученные в *главе 5* инструменты рисования. Кроме того, вы можете класть поверх этого слоя или под ним другие слои, на которых тоже что-то нарисовано. И, наконец, можно легко переключаться между слоями, чтобы работать с нарисованной на них графикой. Каждый слой имеет уникальное имя, с помощью которого он однозначно идентифицируется.

Выясним теперь преимущества, даваемые слоями.

- ❑ В *главе 5* вы познакомились с фрагментацией и слиянием графических фрагментов. В качестве борьбы с этим явлением предлагались группировка, преобразование фрагмента графики в образец и "разнесение" фрагментов по разным слоям. В самом деле, если какие-либо графические фрагменты находятся в разных слоях, то они не будут ни фрагментироваться, ни сливаться. А все потому, что они не соприкасаются друг с другом.
- ❑ Вы уже знаете, что методом трансформации движения можно анимировать только один графический элемент в слое. Но если слоев может быть сколько угодно, то и количество анимированных элементов тоже не ограничено. Каждый анимированный элемент находится в своем слое, не мешая таким образом другим анимированным элементам. Единственный недостаток такого подхода: в сложных фильмах слоев может быть очень много.
- ❑ Создание некоторых специальных эффектов, таких, как слоинаправляющие и маскирующие слои. Они будут описаны далее в этой главе.

Еще нужно сказать, что слои сами по себе не увеличивают размер файла изображения Flash. Увеличивает его размер только графика, расположенная в этих слоях.

Применение слоев

Здесь мы опишем работу со слоями и их применение для создания сложной графики и анимации.

Создание и использование слоев

Взгляните в верхнюю левую часть окна документа, левее уже знакомой вам временной шкалы. Вы увидите нечто, напоминающее список, представляющий собой таблицу из четырех колонок (рис. 15.1). В самой левой колонке стоит что-то вроде *Layer 1*. Это список слоев, уже созданных в документе, а *Layer 1* — имя пока что единственного слоя, полученного при создании нового документа самим Flash.

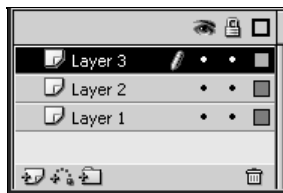


Рис. 15.1. Список слоев документа

Слои в этом списке располагаются в том порядке, в каком они "сложены" на рабочем листе. В нашем случае, слой *Layer 1* будет самым нижним, слой *Layer 3* — самым верхним, а слой *Layer 2* будет находиться между ними. Вы можете щелкнуть по нужной строке списка, чтобы выделить соответствующий слой, выделенный слой помечается черным цветом и изображением ручки (на рис. 15.1 это слой *Layer 3*).

На рис. 15.1 вы видите три слоя с именами *Layer 1*, *Layer 2* и *Layer 3*. На самом же деле, изначально там будет присутствовать только один слой — *Layer 1*, который создаст для нас Flash. Чтобы добавить новый слой, нажмите кнопку **Insert Layer** (рис. 15.2), расположенную в нижней части списка слоев. Вы также можете выбрать пункт **Layer** в меню **Insert** или пункт **Insert Layer** в контекстном меню выделенного слоя. Новый слой будет вставлен в список сразу же над выделенным слоем и автоматически будет выделен.



Рис. 15.2. Кнопка **Insert Layer**

Создайте три слоя, как показано на рис. 15.1. Сейчас мы сделаем небольшой учебный фильм, показывающий, как можно анимировать сразу несколько графических элементов. А именно, мы сделаем неподвижный графический фон и два движущихся "над ним" элемента — квадрат и круг.

Но, прежде всего, дайте слоям понятные имена. Для этого дважды щелкните по имени нужного слоя. После этого вместо его имени появится небольшое поле ввода, в котором будет поставлено старое имя слоя. Введите новое имя и нажмите клавишу <Enter>. Если же вы передумали менять имя слоя, нажмите клавишу <Esc>.

В отличие от имен образцов, имена слоев могут содержать буквы кириллицы и пробелы, так что вы сможете дать своим слоям действительно понятные имена. Дайте слоям имена *Круг*, *Квадрат* и *Фон* в порядке сверху вниз. И приступим к рисованию.

Сначала рассмотрим, как делается неподвижный фон нашей анимации. Как вы знаете, стандартными средствами Flash можно сделать только однотонную заливку фона рабочего листа. Цвет фона задается с помощью селектора цвета **Background Color** диалогового окна **Document Properties** (см. рис. 2.1). Если вы хотите сделать для фона рабочего листа градиентную или графическую заливку, вам придется прибегнуть к небольшой хитрости. А именно, разместить в самом "нижнем" слое прямоугольник таких же, как рабочий лист, размеров и задать для него нужную заливку. Этот прямоугольник и станет фоном вашей анимации.

Итак, приступим к делу. Выделите в списке слой *Фон* и нарисуйте на рабочем листе большой прямоугольник. Создайте для него градиентную заливку, причем сделайте ее посветлее, чтобы на ее фоне были хорошо различимы круг и квадрат. Можете удалить у прямоугольника контур — он нам все равно не понадобится. Оставшаяся заливка и будет нашим неподвижным фоном.

Далее выделите слой *Квадрат* и нарисуйте на рабочем листе квадрат или прямоугольник. Преобразуйте его в графический образец под именем *Квадрат*. Для этого выделите его (и контур, и заливку) и выберите в меню **Insert** пункт **Convert to Symbol**. В диалоговом окне **Convert to Symbol** (см. рис. 10.2) в поле ввода **Name** введите *Квадрат*, включите переключатель **Graphic** и нажмите кнопку **ОК**. После этого Flash создаст новый образец и автоматически заменит выделенный квадрат экземпляром этого образца.

Осталось только выделить слой *Круг*, нарисовать в нем круг или эллипс и преобразовать его в образец *Круг*. Как это сделать, вы уже знаете.

Все, изображения готовы. Теперь приступим к созданию анимации.

Здесь нужно дать кое-какие пояснения. Взгляните еще раз на рис. 15.1. Вы видите, что последовательность кадров у каждого созданного нами слоя своя. Это значит, что можно создавать разные анимации (покадровые и трансформационные) для каждого слоя и его содержимого отдельно, не затрагивая другие слои. Так мы и поступим.

Выделим слой **Квадрат**, и Flash сам выделит все его содержимое — единственный экземпляр образца **Квадрат**. Пусть он движется вертикально вверх-вниз (это только пример, поэтому вы можете задать и другое, более сложное движение). Переместим **Квадрат** в самую верхнюю точку его траектории, выберем пункт **Create Motion Tween** в меню **Insert**, создадим новый ключевой кадр на позиции, скажем, № 10 шкалы кадров, выделим новый кадр и переместим **Квадрат** в нижнюю точку траектории. Вот и все, анимация готова.

Далее выделим слой **Круг**. Для **Круга** зададим горизонтальное движение влево-вправо. Делается это аналогично.

Кажется, все готово. Давайте проверим, что у нас получилось. Нажмем клавишу <Enter> и Да-а-а!.. Анимации-то работают, но фон присутствует только на первом кадре, а потом куда-то пропадает. Что делать?

Чтобы неподвижный фон присутствовал во всех кадрах анимации, нам нужно "растянуть" кадр, где он присутствует, на все эти кадры. Вы можете сделать это, выделив на временной шкале пустое место там, где должен закончиться ваш растянутый кадр, и выбрать пункт **Insert Frame** в контекстном меню. Если вам нужно изменить размер растянутого кадра, перетащите его концевой маркер, удерживая нажатой клавишу <Ctrl>. Результат показан на рис. 15.3.



Рис. 15.3. Растянутый кадр, содержащий неподвижный фон анимации, должен "охватывать" ее полностью

Вот теперь все работает нормально! На всякий случай, если у вас что-то не получилось, приводим рис. 15.4. Посмотрите на него и исправьте ошибки.

С помощью слоев вы можете создавать значительно более сложную анимацию. Например, сделать переливающуюся надпись.

Создайте новый документ Flash. В середине рабочего листа создайте текстовый блок и поместите в него надпись **Flash**. Давайте сделаем так, чтобы буквы этой надписи волнообразно появлялись и пропадали. Получится очень интересный эффект, который часто используется на страницах-заставках Web-сайтов.

Для того чтобы реализовать такой эффект, нам нужно над каждой буквой этой надписи выполнить трансформацию движения. А именно, изменить ее прозрачность с полной (элемент полностью невидим) до нулевой (элемент полностью видим) и обратно до полной. В результате этого буква сначала плавно появится, а потом снова исчезнет.

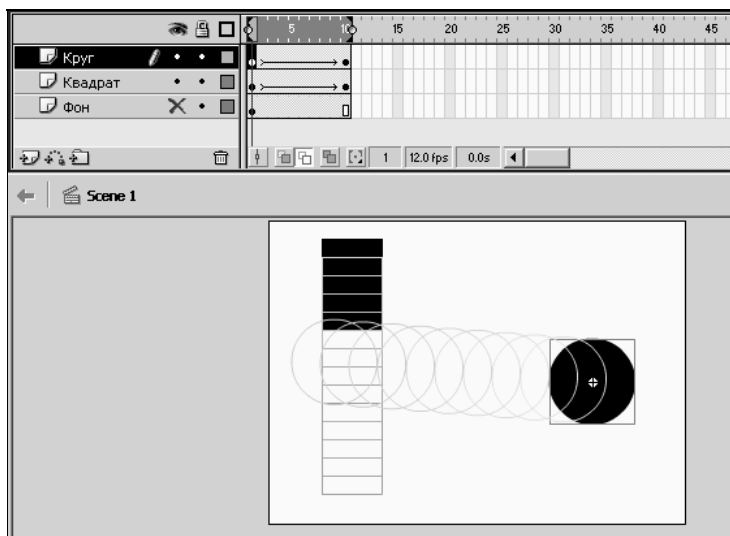


Рис. 15.4. Созданная нами готовая анимация (неподвижный фон временно удален)

Но мы также знаем, что трансформацию движения можно применить над единственным элементом в слое. Выходит, что нам для достижения своей цели нужно "разбросать" все буквы надписи Flash по разным слоям! Но как это сделать? Неужели придется создавать все пять слоев, рисовать на них буквы и следить, чтобы они находились на своих местах? Нет ли другого способа сделать это?

Есть. Flash содержит встроенные средства для *распределения* выделенных графических элементов по разным слоям. Таким образом, если вы выберете несколько графических элементов, Flash создаст нужное количество новых слоев и поместит в них эти элементы. Распределенные по слоям элементы пропадут с изначального слоя, где они находились до распределения, остальное же содержимое этого слоя останется нетронутым.

Имена вновь созданным слоям даются по таким правилам:

- ☐ слою, содержащему экземпляр, дается имя, совпадающее с именем образца, на основе которого был создан этот экземпляр;
- ☐ слою, содержащему именованный графический элемент (например, текстовое поле или образец-клип), дается имя, совпадающее с именем этого элемента;
- ☐ слою, содержащему символ из разбитого на части текстового блока, дается имя, совпадающее с этим символом;
- ☐ слою, содержащему любые другие графические элементы, дается имя вида "Layer <цифра>".

Сначала разобьем нашу надпись на отдельные символы, т. е. создадим несколько (в нашем случае, пять) текстовых блоков, каждый из которых будет содержать только одну букву. Для этого выделим надпись и выберем пункт **Break Apart** в меню **Modify** или контекстном меню выделенной надписи или просто нажмем комбинацию клавиш <Ctrl>+. Созданные Flash текстовые блоки станут выделенными.

Теперь нам нужно "разбросать" полученные текстовые блоки по слоям. Сначала проверим, выделены ли они. Затем выберем пункт **Distribute to Layers** в меню **Modify** или контекстном меню выделенных элементов или нажмем комбинацию клавиш <Ctrl>+<Shift>+<D>. Результат показан на рис. 15.5. Как видите, Flash создал пять новых слоев, назвав их соответствующими буквами надписи, изначальный слой (Flash) остался, но в нем теперь ничего нет.



Рис. 15.5. Результат распределения надписи по слоям

Удалим изначальный слой *Flash*, чтобы он нам не мешал. И создадим анимацию для всех букв надписи по очереди.

Для того чтобы задать преобразование цвета для текстового блока, его нужно преобразовать в образец. Выполните преобразование в образец, которому дайте имя *F* (как и слою). Создайте две анимационные последовательности: первая последовательность будет выполнять постепенное появление буквы (изменение прозрачности от 0 до 100%), а вторая — наоборот, постепенное исчезновение буквы. У вас должна получиться анимация из двух последовательностей, показанная на рис. 15.6.

Создадим теперь анимацию для следующей буквы — *l*. Преобразуем ее также в образец с именем *l*. И посмотрим на временную шкалу.



Рис. 15.6. Анимация из двух последовательностей, реализующая постепенное появление первой буквы и затем ее постепенное исчезновение

Сделаем так, чтобы анимация второй буквы начиналась чуть позже, чем анимация первой буквы, скажем, на один кадр. Поэтому сдвинем первый ключевой кадр будущей анимационной последовательности на одно деление шкалы вправо. (Flash, чтобы заполнить образовавшийся "пробел", создаст пустой ключевой кадр в первой позиции шкалы, но это не очень важно для нас сейчас.) И создадим точно такую же анимацию из двух последовательностей, как изображено на рис. 15.6, только сдвинутую вправо на один кадр. Результат показан на рис. 15.7.



Рис. 15.7. Анимация из двух последовательностей, реализующих постепенное появление соответственно первой и второй букв и затем постепенное их исчезновение

Создайте аналогичные анимации для трех остальных букв. Не забывайте сдвигать каждую из них вправо на кадр. Готовая анимация показана на рис. 15.8.



Рис. 15.8. Готовая анимация, реализующая постепенное появление всех букв надписи и затем постепенное их исчезновение

Как видите, распределяя анимированные элементы по разным слоям, мы можем создавать сколь угодно сложную анимацию. А также делать некоторые элементы, например, графический фон, неподвижными.

Управление слоями

Вы можете добавить новый слой, нажав кнопку **Insert Layer** (см. рис. 15.2), расположенную в нижней части списка слоев. Вы также можете выбрать пункт **Layer** в меню **Insert** или пункт **Insert Layer** в контекстном меню выделенного слоя. Новый слой будет вставлен в список сразу же над выделенным слоем.

Вы можете перемещать слои в списке, меняя порядок их перекрытия друг другом. Для этого просто перетащите нужный слой мышью на новое место.

Есть три способа выделить в списке нужный слой. Во-первых, вы можете щелкнуть по нему мышью в списке слоев, после этого все содержимое этого слоя на рабочем листе будет выделено. Во-вторых, вы можете щелкнуть по любому кадру анимации, созданной в этом слое. В-третьих, вы можете просто выделить на рабочем столе один из графических фрагментов, находящихся в этом слое. (Помните: все слои прозрачны, поэтому сквозь пустое пространство слоев вы можете видеть все, что находится под ними.)

Вы можете также выделять сразу несколько слоев в списке. Если вам нужно выделить непрерывную группу слоев, щелкните по первому слою в группе, нажмите клавишу <Shift> и, удерживая ее, щелкните по последнему слою в группе. Если же вам нужно выделить несколько несвязанных слоев, сначала щелкните по первому, а потом продолжайте щелкать по остальным, удерживая нажатой клавишу <Ctrl>. Как видите, здесь работает та же методика, что и в Проводнике Windows.

Слои можно переименовывать. Для этого дважды щелкните по имени нужного слоя. После этого вместо его имени появится небольшое поле ввода, в котором будет поставлено старое имя слоя. Введите новое имя и нажмите клавишу <Enter>. Если же вы передумали менять имя слоя, нажмите клавишу <Esc>.

Удалить ненужный слой вы можете разными способами. Проще всего выделить слой, который вы хотите удалить, и нажать кнопку **Delete Layer** (рис. 15.9), расположенную в нижней части списка слоев. Также вы можете перетащить ненужный слой прямо на эту кнопку. Ну и, наконец, вы можете выбрать пункт **Delete Layer** в контекстном меню выделенного слоя. Учтите только, что при удалении слоя удаляется также вся расположенная в нем графика.



Рис. 15.9. Кнопка **Delete Layer**

Вы можете переносить последовательности кадров из одного слоя в другой, просто выделяя их и перетаскивая мышью. Если вы хотите скопировать последовательность кадров в другой слой, то при ее перетаскивании удерживайте нажатой клавишу <Alt>.

Для копирования последовательностей кадров из одного слоя в другой можно пользоваться пунктом **Copy Frames**, а для переноса — пунктом **Cut Frames**. Эти пункты находятся в меню **Edit** и контекстном меню выделенного кадра или последовательности кадров. Вы также можете воспользоваться комбинациями клавиш <Ctrl>+<Alt>+<C> и <Ctrl>+<Alt>+<X> соответственно. Просто выделите нужную последовательность кадров и выберите один из этих пунктов. Далее выделите пустое место во временной шкале другого слоя и выберите пункт **Paste Frames** в меню **Edit** и контекстном меню выделенного кадра или последовательности кадров или нажмите комбинацию клавиш <Ctrl>+<Alt>+<P>.

Теперь давайте еще раз посмотрим на список слоев. Как вы уже знаете, он состоит из четырех колонок (порядок перечисления слева направо):

- ☐ имя слоя;
- ☐ видимость слоя (помечена изображением глаза);
- ☐ блокировка слоя (помечена изображением замка);
- ☐ отображение содержимого слоя разными цветами (помечено черным прямоугольником).

Колонка имени слоя нам уже знакома. Давайте рассмотрим возможности, предоставляемые остальными тремя колонками.

Вторая колонка (видимость слоя) позволяет убрать с рабочего листа (но не из списка) на время целый слой вместе с его содержимым. Таким образом, вы можете временно убрать с глаз целые фрагменты графики. Чтобы скрыть слой, щелкните мышью по точке, находящейся в этой колонке, — и вместо точки появится красный крестик, означающий, что данный слой скрыт. Чтобы вновь "открыть" скрытый слой, щелкните по красному крестику, и он исчезнет.

Также Flash предоставляет возможность *заблокировать* какие-либо слои. Графика, находящаяся в заблокированном слое, недоступна для изменения. Это может понадобиться, например, если вы хотите изменить часть какой-либо сложной графики, расположенную в одном слое, не затрагивая остальные слои. Чтобы заблокировать слой, щелкните мышью по точке, находящейся в колонке блокировки слоя (третья по счету). Опять же, после щелчка вместо точки появится красный крестик, который пропадет при следующем щелчке.

Последняя — четвертая — колонка позволяет отобразить содержимое слоя схематично и раскрасить его в разные цвета. При этом графические фрагменты будут отображаться в виде контуров, без заливок. Это часто помогает пра-

вильно определить, в каком слое находится тот или иной графический фрагмент. Чтобы отобразить содержимое слоя схематично, щелкните мышью по цветному квадрату, находящемуся в четвертой колонке. При этом сплошной квадрат превратится в контур, и содержимое данного слоя отобразится схематично. Чтобы вернуть графике нормальный вид, снова щелкните по цветному квадрату, точнее, контуру, и он снова примет вид квадрата.

Если вы щелкните, скажем, по точке во второй колонке, удерживая клавишу <Ctrl>, то скроете сразу все слои. Такого же результата вы добьетесь, если щелкнете по значку глаза — заголовку колонки. Если же вы щелкнете по точке, удерживая клавишу <Alt>, то скроете все слои, кроме того, кому принадлежит эта точка. Все эти приемы также действуют и для других колонок списка строк.

Если вы дважды щелкнете по значку, находящемуся левее имени слоя, на экране появится диалоговое окно **Layer Properties** (рис. 15.10). Вы также можете выбрать пункт **Layer** в меню **Modify** или контекстном меню выделенного слоя. В диалоговом окне можно задать некоторые параметры слоя.

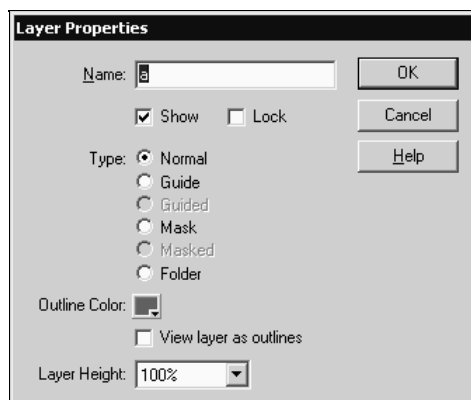


Рис. 15.10. Диалоговое окно **Layer Properties**

В поле ввода **Name** задается имя слоя. Флажок **Show** включает или отключает показ слоя. Флажок **Lock** позволяет заблокировать слой. А флажок **View layer as outlines** включает или отключает схематичный показ содержимого слоя. Все это вам уже знакомо.

Набор переключателей **Type** задает *тип* слоя. Его мы рассмотрим далее в этой главе.

С помощью селектора цвета **Outline Color** вы можете задать цвет, которым будет отображаться графика, находящаяся в этом слое, если включен режим схематичного отображения. Вряд ли вы будете часто пользоваться этим селектором цвета: Flash и сам довольно удачно распределяет цвета соответствующим слоям.

С помощью раскрывающегося списка **Layer Height** вы можете задать высоту строк списка слоев. Этот список содержит три пункта: **100%** (значение по умолчанию), **200%** и **300%**. На рис. 15.11 показан слой **a**, представленный строкой утроенной высоты (пункт **300%** раскрывающегося списка **Layer Height**). Однако вряд ли сейчас вам будет полезна эта возможность, она пригодится, когда вы будете работать со звуком. (О работе со звуком см. главу 17.)



Рис. 15.11. Слой, представленный строкой утроенной высоты

Задав нужные параметры, нажмите кнопку **ОК**. Если же вы передумали изменять их, нажмите кнопку **Cancel**.

Кстати, о высоте строк списка слоев. Вы можете уменьшить их высоту примерно в полтора раза, выбрав пункт **Short** в дополнительном меню временной шкалы. Это показано на рис. 15.12. Таким образом, можно просматривать большее количество строк в списке слоев без его прокрутки. Обратите внимание, что заданная вами высота строки слоя **a** осталась утроенной.

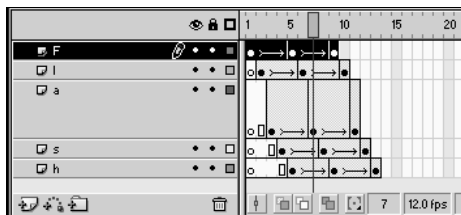


Рис. 15.12. Компактное представление строк списка слоев

И, разумеется, вы можете менять размеры списка слоев, перетаскивая мышью его горизонтальную и вертикальную границы.

Использование папок в списке слоев

Для организации слоев в иерархическую структуру Flash предлагает *папки*. Используя папки, вы можете "разложить" по отдельным "полочкам" слои, представляющие разные графические элементы. Например, так:

кнопки

кнопка 1

кнопка 2

кнопка 3

. . .

клипы

клип 1

клип 2

. . .

графика

элемент 1

элемент 2

. . .

неподвижные элементы

фон 1

фон 2

. . .

Или так:

сцена 1

кнопка 1

элемент 1

клип 1

. . .

сцена 2

кнопка 2

кнопка 3

элемент 2

фон 1

. . .

сцена 3

элемент 3

фон 2

звук 1

. . .

Выбор, какой схемы придерживаться, — за вами. Подробнее это было описано в *главе 10*, посвященной библиотекам и образцам. Все, сказанное там, справедливо и в случае слоев.

Чтобы создать папку, нажмите кнопку **Insert Layer Folder** (рис. 15.13), расположенную в нижней части списка слоев. Вы также можете выбрать пункт **Layer Folder** в меню **Insert** или пункт **Insert Folder** контекстного меню слоя. Вновь созданная папка появится выше выделенного слоя (рис. 15.14).

Рис. 15.13. Кнопка **Insert Layer Folder**

Рис. 15.14. Папка списка слоев

Новая папка получит имя вида "Folder <номер>". Советуем вам сразу же переименовать ее. Переименование папок выполняется так же, как переименование слоев.

Чтобы поместить слой или несколько слоев в папку, просто выделите их и перетащите мышью на строку списка, обозначающую эту папку, или на один из слоев, уже находящихся в папке. Папка с помещенными в нее слоями показана на рис. 15.15.

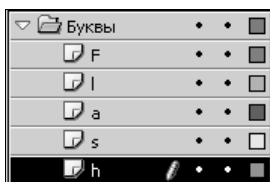


Рис. 15.15. Папка с помещенными в нее слоями

Flash также поддерживает создание папок, вложенных в другие папки (рис. 15.16). Просто выделите любой слой, находящийся в папке, и нажмите кнопку **Insert Layer Folder**. Также вы можете перетащить одну папку в другую. Это позволяет создавать сложные иерархические структуры. Следите только, чтобы такая структура не стала в один прекрасный день слишком сложной.

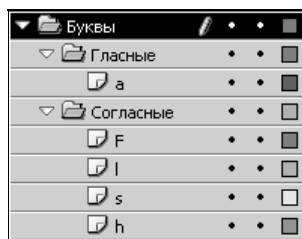


Рис. 15.16. Папки, вложенные в другие папки

Левее имени папки находится значок папки, а еще левее — небольшая треугольная стрелка. Если она повернута вниз, то папка раскрыта, и вы можете

наблюдать все ее содержимое. Если она повернута вправо, то папка закрыта. Чтобы переключить папку из свернутого в развернутое состояние, щелкните мышью значок треугольной стрелки. Также вы можете выбрать в контекстном меню папки или находящегося в ней слоя пункт **Expand Folder** (развернуть папку) или **Collapse Folder** (свернуть папку). А пункты **Expand All Folders** и **Collapse All Folders** позволят вам соответственно развернуть или свернуть все папки списка слоев.

Чтобы извлечь слой из папки, просто "вытащите" его мышью наружу. Таким же образом вы можете перемещать слои между папками. То же самое справедливо и для вложенных папок.

Вы можете перемещать папки в списке, изменяя порядок их расположения. При этом все слои, находящиеся в этих папках, перемещаются вместе с папками. И, конечно, вы можете переупорядочивать слои внутри папок.

Вы можете выделять несколько папок, переименовывать их, блокировать, скрывать и переключать в схематичный режим все находящиеся в них слои. Эти операции выполняются так же, как и для слоев.

Чтобы удалить ненужную папку, выделите ее и нажмите кнопку **Delete Layer** (см. рис. 15.9), расположенную в нижней части списка слоев. Не удивляйтесь — эта кнопка действует и для папок. Вы также можете выбрать пункт **Delete Folder** в контекстном меню выделенной папки. Имейте в виду, что папка будет удалена вместе со всеми находящимися в ней слоями.

Вы можете переносить последовательности кадров из всех слоев папки в другую папку. Для этого сначала сверните нужную папку. После этого Flash автоматически выделит последовательность кадров слоев, находящихся в этой папке. Вам останется только перетащить их мышью в нужный слой. Вы не можете перетащить кадры прямо в папку: вам нужно будет хотя бы создать в ней новый пустой слой. Если вы хотите скопировать последовательность кадров из папки в другой слой, то при ее перетаскивании удерживайте нажатой клавишу <Alt>. Можно также воспользоваться уже знакомыми вам пунктами **Cut Frames**, **Copy Frames** и **Paste Frames** меню **Edit** и контекстного меню выделенной последовательности кадров.

И, наконец, вы можете превратить слой в папку (и наоборот). Для этого дважды щелкните по значку, находящемуся левее имени слоя, или выберите пункт **Layer** в меню **Modify** или контекстном меню выделенного слоя. На экране появится диалоговое окно **Layer Properties** (см. рис. 15.10). Включите переключатель **Folder** в группе **Type** и нажмите кнопку **OK**. Учтите, что содержимое этого слоя безвозвратно пропадет.

Чтобы превратить папку в слой, вам нужно будет выбрать переключатель **Normal**. Слои, вложенные в эту папку, останутся нетронутыми.

Специальные слои

А теперь пора рассказать об особых разновидностях слоев. Это слои-направляющие и маскирующие слои.

Слои-направляющие

При создании анимации очень часто бывает нужно, чтобы какой-либо элемент двигался по некоему *пути*. Таким путем может быть прямая, кривая или ломаная линия, окружность или сложный контур. Обычными средствами, которые мы изучили в *главе 14*, направить анимированный элемент по пути невозможно. Для этого нужно использовать специальный слой, называемый *слоем-направляющей*.

Рассмотрим, как создаются и используются слои-направляющие.

Откройте новый документ Flash, поместите на рабочий лист прямоугольник и создайте простую анимацию, перемещающую его по листу слева направо. Как это сделать, вы уже знаете. После этого сохраните получившийся фильм.

Чтобы получить слой-направляющую и привязать к нему анимированный элемент, выделите слой, содержащий этот анимированный элемент, и выберите пункт **Motion Guide** в меню **Insert**. Вы также можете выбрать пункт **Add Motion Guide** в контекстном меню выделенного слоя. Созданный вами слой-направляющая появится над выделенным слоем (рис. 15.17) и будет иметь имя вида "Guide: <имя выделенного слоя>".

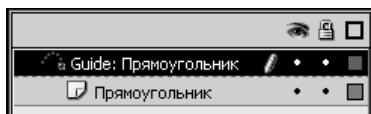


Рис. 15.17. Слой-направляющая

Теперь переключитесь на слой-направляющую и нарисуйте нужный вам путь. Для этого вы можете пользоваться инструментами "карандаш", "перо", "линия", "прямоугольник", "эллипс" и "кисть". Проследите, чтобы в слое-направляющей не было никакой графики, кроме пути.

Нарисовав путь, привяжите к нему ваш анимированный элемент. Для этого выделите анимированный элемент и переместите его так, чтобы его точка фиксации "приклеилась" к линии пути. Если вы не уверены в верности своей руки или исправности мыши, то можете включить флажок **Snap** в редакторе свойств (см. рис. 14.1). После этого Flash автоматически "приклеит" элемент к пути за его точку фиксации.

Собственно, это все. Вам останется только скрыть "с глаз долой" слой-направляющую, чтобы он не портил общую картину. Для этого щелкните мы-

шью по точке, находящейся во второй колонке списка слоев. И можете проверить вашу анимацию. У вас должно получиться нечто, похожее на рис. 15.18.

В редакторе свойств вы можете найти еще один флажок — **Orient to Path**. Если его включить, то Flash будет автоматически ориентировать анимированный элемент по линии пути. Мы не будем подробно описывать, что это значит, — просто взгляните на рис. 15.19.

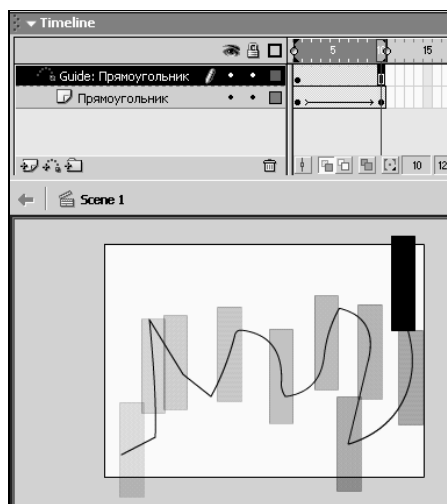


Рис. 15.18. Готовая анимация, использующая слой-направляющую

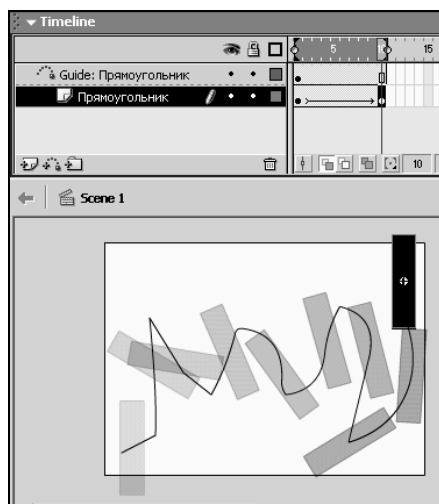


Рис. 15.19. Готовая анимация, использующая слой-направляющую и автоматическое ориентирование по пути

Соответственно, если вы хотите "отвязать" анимированный элемент от пути, вам нужно не только "отклеить" его от самой линии пути, но и отключить флажки **Snap** и **Orient to Path**. Более того, отключить эти флажки нужно в первую очередь.

Создать слой-направляющую можно и другим способом. Создайте новый слой, обычный, какие вы создавали ранее в этой главе, выделите его и выберите пункт-выключатель **Guide** в контекстном меню выделенного слоя. Вы также можете выбрать пункт **Properties** в контекстном меню, включить в появившемся на экране диалоговом окне **Layer Properties** переключатель **Guide** и нажать кнопку **ОК**. Обратите внимание, что слой-направляющая, к которому не привязан ни один слой, имеет другой значок, чем слой-направляющая, к которому привязаны слои с анимированными элементами.

Чтобы привязать теперь к вновь созданному слою-направляющей другие слои с анимацией, вы можете использовать три разных способа. Во-первых, можно просто перетащить слой с анимацией и "бросить" под слоем-направляющей. Во-вторых, можно выделить какой-либо слой, уже привязанный к слою-направляющей, и создать новый слой. (Как вы знаете, новый слой помещает-

ся прямо над выделенным, значит, в нашем случае он также будет привязан к слою-направляющей.) В-третьих, можно выбрать пункт **Properties** в контекстном меню, включить в появившемся на экране диалоговом окне **Layer Properties** переключатель **Guided** и нажать кнопку **OK**.

"Отвязать" же слой с анимацией от слоя-направляющей можно двумя способами. Вероятно, вы уже догадались, какими. Первый способ: перетащить слой с анимацией так, чтобы он оказался выше слоя-направляющей. Второй способ: выделить слой с анимацией или слой-направляющую, выбрать пункт **Properties** в контекстном меню, включить в появившемся на экране диалоговом окне **Layer Properties** переключатель **Normal** и нажать кнопку **OK**. В этом случае либо слой с анимацией будет "отвязан" от слоя-направляющей, либо слой-направляющая будет преобразован в обычный слой, что вызовет такой же результат. Чтобы превратить слой-направляющую в обычный слой, вы также можете отключить пункт-выключатель **Guide** в контекстном меню выделенного слоя.

Маскирующие слои

Вероятно, вам встречались изображения Flash, по которым "ползает" своего рода "прожектор" — круглое светлое пятно, "высвечивающее" изображение частями. Такие изображения встречаются довольно часто: иногда это карты звездного неба, иногда — карты земной поверхности, а иногда — какие-то "картины реальной жизни". Такое светлое пятно, "высвечивающее" в один момент времени только часть какого-либо лежащего под ним изображения, совпадающую с его размерами, называется *маской*, слой, на котором оно находится, — *маскирующим слоем*, а "высвечиваемое" маской изображение — *маскируемым* изображением. В случае Flash маскируемое изображение находится в маскируемом слое.

Вы можете думать, что маска — это отверстие в непрозрачном маскирующем слое. (Обычно слои, как вы знаете, прозрачны.) Сквозь это отверстие можно видеть все, что лежит под этим слоем. Отверстие может быть любым графическим фрагментом: геометрической фигурой, экземпляром (только не кнопкой) или текстовым блоком. Наконец, это отверстие может быть анимировано методами трансформационной анимации.

Как же создать и использовать маскирующий слой? Очень просто, даже проще, чем слой-направляющую.

Прежде всего, создайте слой с маскируемым изображением. Как это сделать, вы, конечно, знаете, ведь вы уже не раз создавали во Flash различные изображения, в том числе, и анимированные. Можете также анимировать маскируемое изображение: получится очень интересный эффект. Назовите как-нибудь единственный пока что слой этого изображения. И сохраните файл Flash.

Теперь поместите над единственным слоем этого изображения новый пустой слой. Не будем описывать, как это делается. Далее нарисуйте изображение, которое станет вашей маской. При этом имейте в виду следующее:

- ☐ любая заливка станет прозрачной частью маски;
- ☐ пустое пространство на рабочем листе станет непрозрачной частью маски;
- ☐ различные цвета и стили линий, градиентные и графические заливки будут проигнорированы.

Преобразуйте нарисованное вами изображение в образец. Анимлируйте его. "Растяните" единственный ключевой кадр слоя, содержащего маскируемое изображение, так, чтобы он "покрыл" всю анимацию. Теперь все готово к созданию маскирующего слоя.

Выделите в списке слой, содержащий изображение-маску. После этого выберите пункт **Mask** в контекстном меню выделенного слоя. Вы также можете выбрать пункт **Properties** в контекстном меню, включить в появившемся на экране диалоговом окне **Layer Properties** переключатель **Mask** и нажать кнопку **ОК**. Два созданных вами слоя показаны на рис. 15.20.



Рис. 15.20. Маскирующий и маскируемый слой

Как видите, кроме всего прочего, и маскирующий, и маскируемый слои заблокированы. Блокировка обоих слоев — важное условие их нормальной работы во Flash. Если вы не заблокируете хотя бы один слой, маскирование работать не будет. Если же вам нужно что-то изменить, то сначала разблокируйте нужный слой, измените, что хотите, а потом не забудьте заблокировать его снова.

Вы можете маскировать одним маскирующим слоем несколько слоев с графикой. Для этого создайте нужные слои, наполните их графикой, перетащите их и "бросьте" под маскирующим слоем. Вы также можете выбрать пункт **Properties** в контекстном меню, включить в появившемся на экране диалоговом окне **Layer Properties** переключатель **Masked** и нажать кнопку **ОК**.

"Вынуть" же слой с графикой из-под маскирующего слоя можно двумя способами. Первый способ: перетащить слой с маскируемой графикой так, чтобы он оказался выше маскирующего слоя. Второй способ: выделить слой с графикой или маскирующий слой, выбрать пункт **Properties** в контекстном меню, включить в появившемся на экране диалоговом окне **Layer Properties** переключатель **Normal** и нажать кнопку **ОК**. В результате этого маскирующий или маскируемый слой станет обычным слоем.

К сожалению, вы не можете маскировать уже маскированный слой, т. е. накладывать маску на маску. Flash не предоставляет стандартных средств, что-

бы сделать это. Вам придется использовать вложенную анимацию, описанную в *главе 14*. Последовательность действий в этом случае будет такая:

1. Создайте анимированный графический образец или образец-клип с двумя слоями: маскирующим и маскируемым. Маскирующий слой этого образца станет маской первого уровня, а маскируемый слой будет содержать собственно маскируемую графику.
2. Вернитесь в основной документ Flash. Поместите полученный образец-клип в слой. Создайте еще один слой и сделайте его маскирующим. Созданный вами слой станет маской второго уровня.
3. Задайте необходимую анимацию и просмотрите готовый фильм.

Чтобы использовать многоуровневое маскирование графики, вам нужно будет создать соответствующее количество образцов-клипов, каждый из которых будет включать в себя предыдущий образец и маску очередного уровня. На рис. 15.21 показано, как это будет выглядеть.

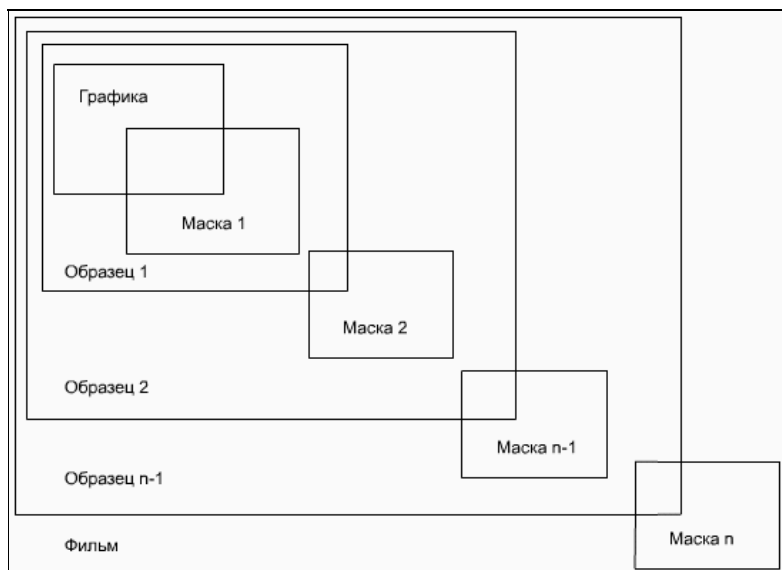


Рис. 15.21. Схема, иллюстрирующая принцип создания многоуровневых масок

Поддержка слоев Проводником Flash

В *главе 10* и *главе 13* этой книги говорилось об инструменте под названием Проводник Flash, который может помочь вам отыскать нужный фрагмент изображения. Как вы знаете, Проводник поддерживает образцы, экземпляры, текстовые блоки, кадры и сцены. Но, кроме того, он поддерживает еще и слои.

Чтобы просмотреть все слои и кадры, составляющие изображение Flash, включите кнопку-выключатель **Show Frames and Layers**, находящуюся в окне Проводника. Также проверьте, включен ли флажок **Layers** в окне **Movie Explorer Settings** (см. рис. 10.31). На рис. 15.22 показано окно Проводника, в котором в виде иерархического списка показана текущая сцена фильма и все входящие в нее слои и кадры.

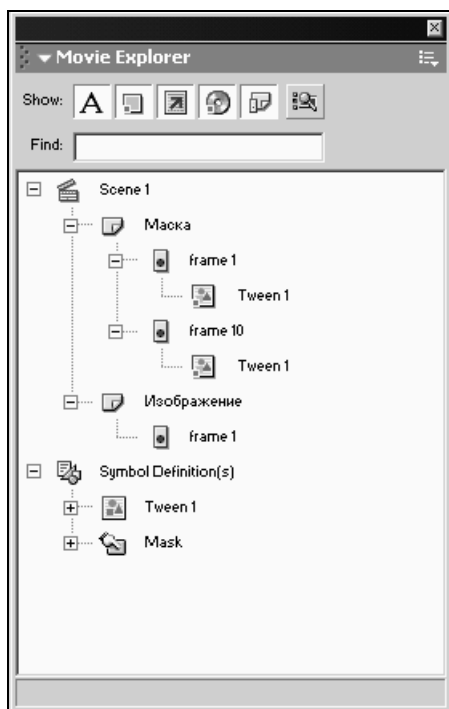
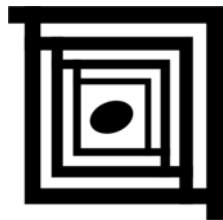


Рис. 15.22. Окно Проводника, отображающее текущую сцену фильма, и список входящих в нее слоев и кадров

Глава 16



Импорт анимации и видео

Об импорте графики мы уже говорили. Также упоминали и о том, что для каждой задачи существует свой идеальный инструмент. Ну и, конечно, мы достаточно подробно разъяснили, какие задачи решает Macromedia Flash. Все это было описано в *главе 8*, где рассказывалось об импорте статичной графики, и, пожалуй, повторять все это здесь — значит зря расходовать бумагу.

И все же мы кое-что скажем. Как-никак, статичная графика и анимация — вещи очень разные, а во многом — диаметрально противоположные.

Прежде всего нужно сказать, что Flash позволяет как внедрять импортированное видео в файл документа Flash, так и создавать ссылку на внешний видеофайл. В первом случае говорят о *внедренном* в документ Flash клипе, а во втором — о клипе, *связанном* с документом Flash. Маленькие видеофайлы, как правило, внедряются в документ Flash, а большие — связываются с ним. К несчастью, поддерживается только связывание с файлами формата QuickTime.

Теперь поговорим о собственно импорте внешнего видео. И укажем, какие файлы лучше внедрять, а какие — связывать.

Чаще всего приходится импортировать во Flash фильмы, созданные в самом Flash, но другими художниками. Например, вы можете "позаимствовать" удачные анимированные элементы для своего изображения или фигурные кнопки для своей Flash-программы. (О программировании во Flash см. *часть 4* этой книги.) Конечно, эти элементы могут быть помещены в обычные или разделяемые библиотеки Flash, но зачастую они просто "лежат" в Сети "россыпью", в виде отдельных SWF-файлов. Поскольку такие файлы имеют небольшой размер, они внедряются в документ Flash, это позволяет сократить до минимума количество файлов, необходимых для показа изображения.

Также Flash поддерживает импорт "анимации", отдельные кадры которой хранятся в пронумерованных файлах. Такие наборы изображений могут быть созданы разными графическими программами, в частности, самим Flash. Подобные файлы также внедряются в документ Flash.

Видеоклипы, сохраненные в других форматах: QuickTime, MPEG и др. (все поддерживаемые Flash видеоформаты описаны в *главе 12*), внедряют во Flash значительно реже. Причины этому чисто технические, и заключаются они в том, что, как правило, в таких форматах распространяются полнометражные игровые, документальные и музыкальные фильмы. А файлы, содержащие такие фильмы, занимают очень большой объем, что не может не сказаться на размерах результирующего файла Flash, а значит, на времени его загрузки. Поэтому, такие файлы практически всегда связывают с документом Flash. В результате, внешняя анимация загружается только тогда, когда в ней возникнет надобность.

При импортировании во Flash внешних файлов возникает еще один вопрос — о защите авторских прав их создателей (если, конечно, вы не создали эти файлы сами). В самом деле, воровать нехорошо. Поэтому прежде, чем позаимствовать что-либо чужое для своего блага, обязательно выясните, как к этому отнесется владелец. На Web-сайтах, занимающихся распространением графики, обязательно указывается, что с этой графикой разрешено делать. Поэтому будьте внимательны: всегда читайте то, что написано мелким шрифтом.

Поддержка форматов анимации и видео

Сначала выясним, какие форматы хранения анимации и видео мы можем импортировать во Flash. Таким образом, мы сразу сможем сказать, что удастся импортировать, а что придется преобразовывать в совместимый формат, используя другие программы.

Список поддерживаемых форматов

В табл. 16.1 приведен список форматов анимации и видео, которые можно импортировать во Flash.

Таблица 16.1. Список форматов анимации и видео, которые можно импортировать во Flash

Название формата	Расширение файлов	Требуется Apple QuickTime 4*	Требуется Microsoft DirectX 7*
Apple QuickTime	mov	+	
AVI	avi	++	+
Microsoft WMA	wma, wmv, asf		+
MPEG	mpeg, mpe, mpg, dat	+	+

Таблица 16.1 (окончание)

Название формата	Расширение файлов	Требуется Apple QuickTime 4*	Требуется Microsoft DirectX 7*
Цифровое видео	dv	+	

* Данная версия или более новая.

** Требуется либо Apple QuickTime 4, либо Microsoft DirectX 7.

Как вы уже знаете, Flash поддерживает импорт "анимации", отдельные кадры которой хранятся в пронумерованных файлах. Если вы импортируете файл растрового изображения (формата BMP, GIF или любого другого), чье имя кончается числом, Flash ищет аналогичные ему файлы. (Подробнее об импорте статичной графики см. главу 8.) Если он находит последовательность таких пронумерованных файлов, например, Рис1.bmp, Рис2.bmp, Рис3.bmp и т. д., то предлагает импортировать всю эту последовательность как видеоклип. На экране появляется соответствующее предупреждение, нажмите кнопку **Yes (Да)** или **No (Нет)**. Если вы нажмете **Yes**, Flash импортирует последовательность файлов и превратит их в клип, если же вы нажмете **No**, будет импортирован только выбранный вами файл.

Если вы хотите связать изображение Flash с внешним файлом, учтите что в данной версии Flash (MX) поддерживается только связывание с файлами QuickTime. Если же вы хотите связать изображение Flash с видеофайлом другого формата, то нужно преобразовать последний в файл QuickTime, воспользовавшись программным пакетом Apple QuickTime.

Внимание!

В некоторых случаях для видеофайлов, содержащих звуковое сопровождение, может быть импортировано только видео, но не звук. Это может произойти из-за того, что Flash не поддерживает данный формат сжатия и записи звуковых данных. Например, такое часто происходит при импорте файлов QuickTime. Чтобы избежать этой проблемы, следует преобразовать звуковую дорожку видеоклипа к одному из совместимых форматов, воспользовавшись специальными программами.

Как Flash обрабатывает внедренные видеофайлы

Как вы знаете, Flash может как внедрять видео в изображение, так и связывать его с внешними видеофайлами. При просмотре изображения Flash связанный файл воспроизводится с помощью поддерживающей данный формат

программой, в нашем случае, Apple QuickTime. Внедренные же файлы Flash воспроизводит "своими силами".

Для сжатия внедренного видео Flash использует особый алгоритм — Sorenson Spark. Можно сказать, что это фирменный алгоритм Macromedia, разработанный специально для Flash MX. Sorenson Spark позволяет сжать видеоданные достаточно сильно для того, чтобы получившийся фильм Flash можно было без особых проблем передать по медленным каналам связи. И, вместе с тем, этот алгоритм обеспечивает достаточно высокое качество изображения, чтобы не разочаровать пользователей.

Все внедренные видеофайлы принудительно перекодируются с использованием алгоритма Sorenson Spark. Сначала внедренный видеофайл декодируется с помощью встроенных средств либо пакета Apple QuickTime, либо расширения операционной системы Windows Microsoft DirectX. При декодировании видеоданные переводятся в некоторое внутреннее представление, которое, в частности, используется и для вывода их на экран. После этого Flash снова кодирует эти данные, но уже с использованием алгоритма Sorenson Spark.

Для обработки видео- и аудиоданных, а именно, сжатия и распаковки, любое программное обеспечение (не только Flash) использует специальный модуль, называемый *кодеком*. Кодеком фактически состоит из двух подмодулей: *кодера*, обеспечивающего сжатие данных, и *декодера*, выполняющего их распаковку. Среда создания изображений и фильмов Flash содержит в себе и кодер, и декодер — полную версию кодека Sorenson Spark. Что касается проигрывателя Flash, то он содержит урезанную версию, состоящую только из декодера.

Полезные советы по подготовке видео для импорта

Здесь мы приведем несколько полезных советов, которые помогут вам правильно подготовить видеоклипы для импорта их во Flash. Они особенно справедливы в тех случаях, если вы подготавливаете материалы, предназначенные для распространения через Интернет, когда размер результирующего файла играет важную роль.

Особенности современных алгоритмов сжатия видео (в том числе, и Sorenson Spark) таковы, что неподвижные или малоподвижные сцены сжимаются лучше, чем быстро меняющиеся. Поэтому не следует выбирать для своего изображения Flash видеоклипы, содержащие какие-либо динамичные, быстро меняющиеся сюжеты. Ну, а если это нужно вам для каких-то специальных целей, имейте в виду, что расплатой будет увеличение размера файла и соответственно времени его загрузки.

Удаляйте шумы из видеоклипов. Мало того, что "зашумленные" клипы плохо сжимаются, так они еще и выглядят не очень презентабельно. И пользователь, даже если он и дождется окончания загрузки вашего изображения, все равно будет разочарован. Также позаботьтесь о том, чтобы видеоклип имел приемлемые яркость, контрастность, насыщенность цветов и пр.

Для уменьшения размеров видеоклипа часто стоит поступиться его качеством. Ниже приведены три характеристики видео, напрямую влияющие на его качество.

- ❑ *Ширина потока данных* или просто поток данных (распространен также термин "битрейт" — калька с английского bitrate). Измеряется в килобитах в секунду и характеризует объем данных, отводимый на хранение видеoinформации. Фактически ширина потока данных показывает, сколько информации выбрасывается при сжатии (как вы помните, для видео используются алгоритмы сжатия с потерями). Чем меньше ширина потока данных, тем хуже качество клипа и тем меньше его размеры.
- ❑ *Частота кадров*. Чем она ниже, тем хуже качество клипа и тем меньше его размеры. Для выбора частоты кадров руководствуйтесь табл. 12.1.
- ❑ *Размеры изображения клипа*. Опять же, чем они меньше, тем хуже качество клипа и тем меньше его размеры. Для выбора размеров клипа руководствуйтесь табл. 16.2.

Таблица 16.2. Стандартные значения размеров клипа

Размеры клипа, мм	По каким каналам будут передаваться клипы
160×120	Модемы
192×144	ISDN
320×240	Быстродействующие каналы (кабельные, оптоволоконно, Ethernet и др.)

Но не переусердствуйте с ухудшением качества видео. Иначе его просто никто не захочет смотреть.

Импорт видео и работа с ним

А теперь давайте рассмотрим, как же в изображение Flash импортируется видео, и какие средства предоставляются для работы с ним.

Как вы знаете, все импортированные во Flash видеоклипы преобразуются в образцы-импортированные видео. Так что вы можете создавать на рабочем листе столько экземпляров импортированного клипа, сколько хотите, и размер файла Flash при этом увеличиваться не будет. (Хотя, конечно, при самом импорте клипа он увеличится, и зачастую очень сильно.)

Внедрение видео

Для того чтобы внедрить в документ Flash внешний видеофайл, поместив его и в библиотеку, и на рабочий лист, выберите пункт **Import** в меню **File** или нажмите комбинацию клавиш <Ctrl>+<R>. На экране появится стандартное диалоговое окно открытия файла **Windows**. Найдите нужный файл и нажмите кнопку открытия файла этого диалогового окна.

Когда вы хотите импортировать видеоклип только в библиотеку, выберите пункт **Import to Library** в меню **File**. На экране появится стандартное диалоговое окно открытия файла **Windows**. Найдите нужный файл и нажмите кнопку открытия файла.

Если вы собираетесь импортировать файл QuickTime, на экране появится диалоговое окно **Import Video** (рис. 16.1). В нем находится всего два переключателя. Переключатель **Embed video in Macromedia Flash document** позволяет вам внедрить видеофайл в документ Flash. А переключатель **Link to external video File** обеспечивает связывание документа Flash с этим видеофайлом. Поскольку нам нужно внедрить видеофайл в документ Flash, следует включить переключатель **Embed video in Macromedia Flash document**. И, разумеется, нажать кнопку **OK**. Если вы нажмете кнопку **Cancel**, файл не будет импортирован.

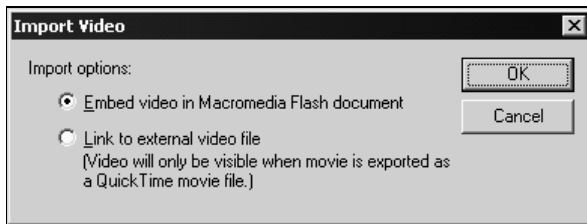


Рис. 16.1. Диалоговое окно **Import Video**

Если вы импортируете видеофайл в любом другом формате, поддерживаемом Flash, диалоговое окно **Import Video** не появится. Дело в том, что только файлы QuickTime могут быть связаны с документом Flash.

Далее на экране появится еще одно диалоговое окно под названием **Import Video Settings** (рис. 16.2). С его помощью вы можете задать различные параметры импортируемого видеоклипа.

С помощью регулятора **Quality** задается степень сжатия видеоданных. Чем она выше, тем ниже качество клипа и тем меньше размер массива видеоданных. Рекомендуется задавать высокую степень сжатия для больших клипов и низкую — для небольших.

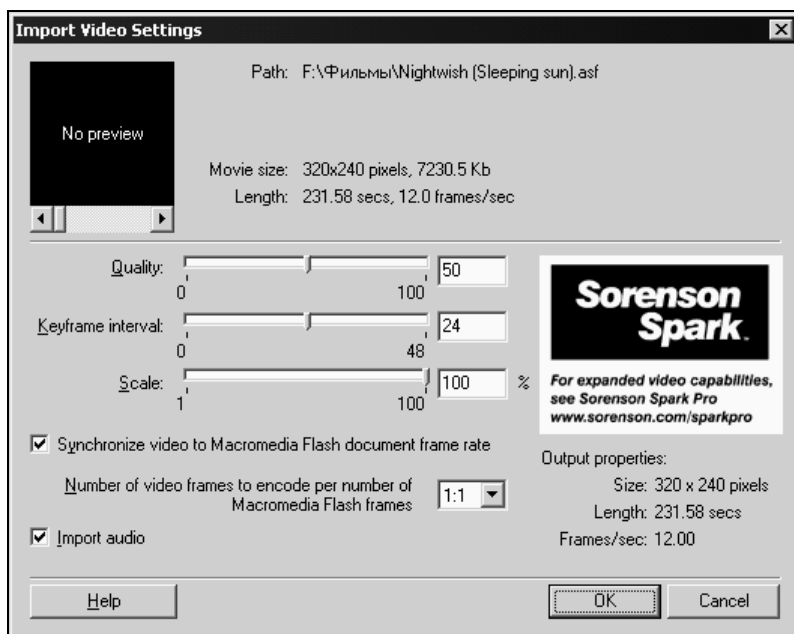


Рис. 16.2. Диалоговое окно **Import Video Settings**

Регулятор **Keyframe interval** задает количество кадров между ключевыми кадрами видео. Дело в том, что для уменьшения размеров видеофайла кодер записывает целиком только некоторые кадры из всей последовательности, такие кадры называются *ключевыми*. *Промежуточные* кадры фактически не записываются, вместо этого кодер записывает только изменения, произошедшие в них по сравнению с ключевыми кадрами. Чем больше интервал между ключевыми кадрами, тем ниже качество видео, но зато тем меньше размер массива видеоданных. Поэтому чем больше видеофайл, тем большее значение регулятора **Keyframe interval** следует установить. Если же видеофайл совсем маленький, можно задать значение, равное единице (то есть, каждый кадр будет ключевым).

С помощью регулятора **Scale** задается масштаб импортированного клипа. Например, чтобы уменьшить его размер наполовину, задайте значение 50%. Чем оно меньше, тем меньше видимые размеры клипа и тем меньше размер массива видеоданных.

Флажок **Synchronize video to Macromedia Flash document frame rate** включает или отключает синхронизацию частоты кадров внедренного клипа с частотой кадров самого документа Flash. Изначально этот флажок всегда включен. Отключите его, когда синхронизация не нужна, например, если вы импортируете обычный фильм или музыкальный клип.

С помощью раскрывающегося списка **Number of Video frames to encode per number of Macromedia Flash frames** задается отношение между количеством кадров импортированного клипа и количеством кадров документа Flash. Иными словами, если вы выберете значение 1:1, то на один кадр документа Flash будет проигран один кадр импортированного клипа. Если же вы выберете значение 2:3, то на два кадра импортированного клипа будут проиграны три кадра документа Flash. В последнем случае и импортированный клип, и основной фильм Flash, так или иначе, будут проиграны синхронно.

Флажок **Import audio** включает или отключает импорт звукового сопровождения. По умолчанию он включен. Если Flash не может импортировать звуковое сопровождение, вместо этого флажка будет отображено предупреждение.

В верхней части диалогового окна находятся сведения об импортируемом видеофайле и небольшая панель предварительного просмотра клипа. К сожалению, предварительный просмотр работает далеко не всегда. А в нижнем правом углу отображаются параметры, которые будет иметь видеоклип после импорта. Таким образом, вы всегда можете контролировать результаты заданных вами в этом диалоговом окне параметров.

Задав нужные параметры, нажмите кнопку **ОК**. Если вы нажмете кнопку **Cancel**, файл импортирован не будет.

Процесс импорта клипа может занять много времени. Так что наберитесь терпения, особенно если у вас старый маломощный компьютер. Во время импорта Flash будет показывать небольшое окно с индикатором прогресса.

Если вы импортировали видеоклип не только в библиотеку, но и на рабочий стол, то после окончания импорта на экране может появиться сообщение, показанное на рис. 16.3. Таким образом Flash говорит вам, что для проигрывания клипа нужно создать анимацию с количеством кадров, равным количеству кадров этого клипа. А поскольку вы эту анимацию еще не создали, Flash собирается сделать это за вас и спрашивает вашего разрешения. Если вы позволяете это ему сделать, нажмите кнопку **Yes**, если нет — кнопку **No**.



Рис. 16.3. Сообщение о добавлении новых кадров

Если вы не хотите, чтобы Flash в дальнейшем выводил это сообщение, перед тем, как нажать кнопку **Yes** или **No**, включите флажок **Don't show me this**

message again. После этого Flash будет автоматически выполнять ваше последнее указание.

Результат всех этих долгих мучений показан на рис. 16.4. Обратите внимание на анимационную последовательность во временной шкале и новый образец в библиотеке.

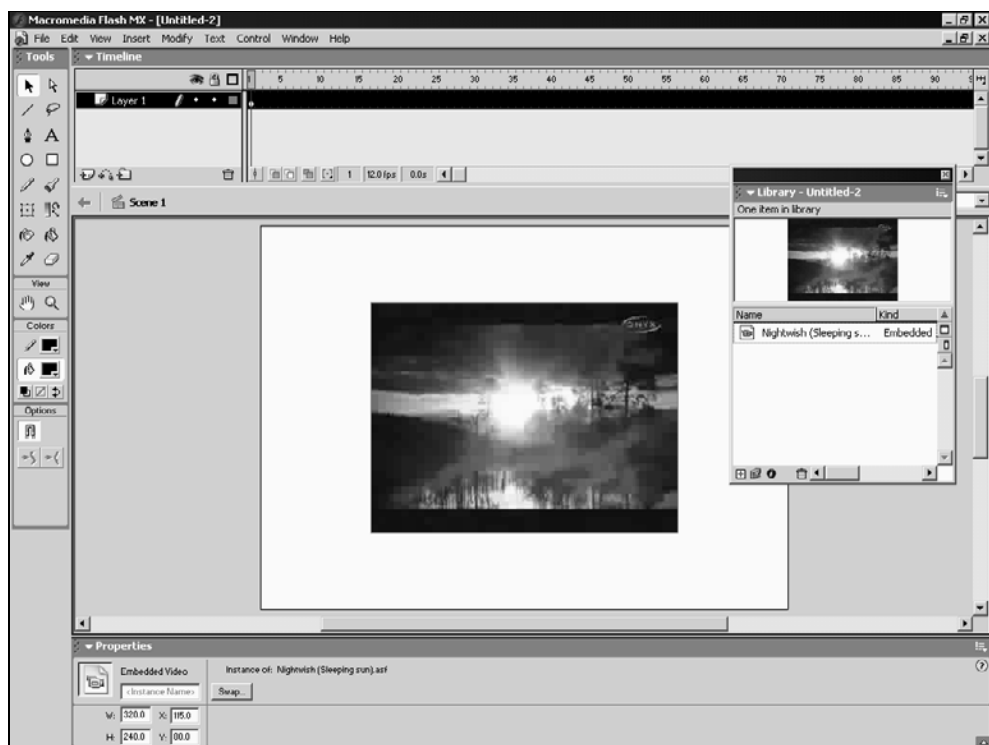


Рис. 16.4. Импортированный клип

Есть еще один способ импортирования видео в библиотеку (но не на рабочий лист). Откройте окно библиотеки, нажав клавишу <F11> или выбрав пункт-выключатель **Library** в меню **Window**. Выберите пункт **New Video** в дополнительном меню окна библиотеки. В списке образцов появится новый с именем вида "Embedded Video <номер>". Выделите его и выберите пункт **Properties** в контекстном меню выделенного образца или дополнительном меню окна. На экране появится диалоговое окно **Embedded Video Properties** (рис. 16.5).

В поле ввода, расположенном в верхней части окна, укажите имя создаваемого образца. После этого нажмите кнопку **Import**. На экране появится стандартное диалоговое окно открытия файла **Windows**. Найдите нужный файл и нажмите кнопку открытия файла этого диалогового окна. После

этого на экране появится уже знакомое вам диалоговое окно **Import Video Settings** (см. рис. 16.2), с помощью которого вы сможете задать различные параметры импортируемого видеоклипа. Заметьте, что таким образом вы не можете связать видеоклип с документом Flash. Задайте нужные параметры и нажмите кнопку **ОК**.

После завершения процесса импорта диалоговое окно примет вид, показанный на рис. 16.6. В частности, в нем будут отображены параметры импортированного файла и появятся еще несколько кнопок. Нажмите кнопку **ОК**, чтобы завершить импорт файла.



Рис. 16.5. Диалоговое окно **Embedded Video Properties**, показывающее параметры еще не импортированного клипа

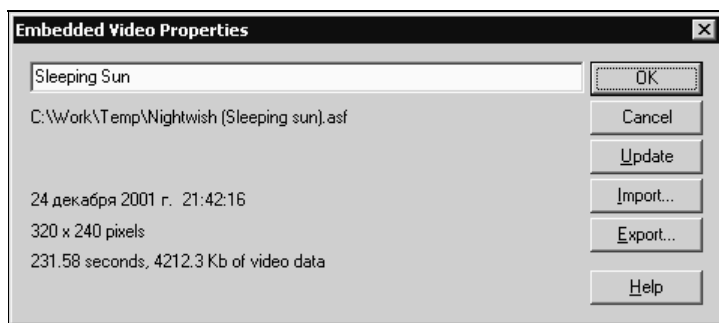


Рис. 16.6. Диалоговое окно **Embedded Video Properties**, показывающее параметры внедренного клипа

Связывание видео

Как вы помните, Flash позволяет связать видеофайл формата QuickTime с документом Flash. При этом сам видеофайл не будет включен в состав документа, а останется внешним файлом. При выводе готового изображения Flash этот файл будет загружен только тогда, когда в нем возникнет необходимость.

Для того чтобы связать с документом Flash внешний видеофайл QuickTime, поместив этот файл и в библиотеку, и на рабочий лист, выберите пункт **Import** в меню **File** или нажмите комбинацию клавиш <Ctrl>+<R>. На экране появится стандартное диалоговое окно открытия файла Windows. Найдите нужный файл и нажмите кнопку открытия файла этого диалогового окна.

Если вы хотите поместить связанный видеоклип только в библиотеку, но не на рабочий лист, выберите пункт **Import to Library** в меню **File**. На экране появится стандартное диалоговое окно открытия файла Windows. Найдите нужный файл и нажмите кнопку открытия файла.

Когда на экране появится диалоговое окно **Import Video** (см. рис. 16.1), включите переключатель **Link to external video file** и нажмите кнопку **OK**. После этого выбранный вами видеофайл будет связан с документом Flash.

Использование импортированного видео

Чтобы импортированный вами видеоклип — внедренный или связанный — нормально воспроизводился, вам нужно создать для него соответствующей длины последовательность кадров, точнее, один растянутый кадр. Обычно это делается автоматически самим Flash при импорте клипа в библиотеку и на рабочий лист. Но если вы отменили автоматическое создание последовательности кадров или импортировали клип только в библиотеку, вам придется позаботиться об этом самим.

Причины отказаться от автоматического формирования последовательности кадров могут быть любыми. Например, вы хотите проиграть только часть клипа или вообще любите все делать своими руками, не полагаясь на технику. В последнем случае вы все же будете не правы: создавать последовательность из нескольких тысяч кадров — занятие, мягко говоря, трудоемкое.

К сожалению, узнать продолжительность клипа в кадрах можно только при его импорте. Если же вы забыли, сколько в клипе кадров, вы можете импортировать его снова, прочитать этот параметр в диалоговом окне **Import Video Settings** и нажать кнопку **Cancel**.

Прежде всего, проверьте, создали ли вы новый экземпляр клипа на рабочем столе. Как создавать экземпляры образцов, вы уже знаете, — об этом рассказывалось в *главе 10*.

Итак, выделите единственный кадр, созданный Flash, и "растяните" его. Для этого добавьте несколько промежуточных кадров, нажимая клавишу <F5>, а когда кадр "растянется" достаточно, "захватите" его мышью, удерживая нажатой клавишу <Ctrl>, и "вытяните" вправо. Переместите курсор мыши за правую границу временной линии — в этом случае Flash выполнит ее автоматическую прокрутку до крайнего правого предела. После этого отпустите кнопку мыши — и вы увидите, что справа появится еще очень много места для вашего растянутого кадра. Повторите описанную выше манипуляцию,

пока последовательность кадров (точнее, растянутый кадр) не примет нужную длину.

После этого вы можете просмотреть импортированный клип. Нажмите клавишу <Enter> — и Flash его проиграет. Если же вы хотите еще и прослушать звуковое сопровождение (например, насладиться потрясающими Nightwish в полной мере), вам придется выбрать пункты **Test Movie** или **Test Scene** в меню **Control** и просмотреть получившийся фильм в отдельном окне Flash. В самой среде Flash звуковое сопровождение импортированного клипа не проигрывается.

Вы можете как угодно искажать находящийся на рабочем листе клип, применяя все инструменты, описанные в *главе 9*. Вы также можете анимировать этот клип. Просто создайте второй ключевой кадр и задайте нужную анимацию. Это будет забавно: на рабочем столе проигрывается и одновременно движется туда-сюда видеоклип.

Вы можете проиграть не весь клип, а некоторую его часть. Вернее, какую-то его часть, начиная с начала. Для этого просто создайте анимационную последовательность с длиной, меньшей числа кадров клипа. К сожалению, проиграть средствами Flash произвольный фрагмент клипа, начинающийся с любого его кадра, а не только с первого, судя по всему, невозможно.

Можно также преобразовать образец-импортированное видео в другой образец, просто выбрав пункт **Convert to Symbol** в меню **Insert** и задав соответствующие параметры в диалоговом окне параметров образца. Это может понадобиться хотя бы для того, чтобы сделать импортированный клип разделяемым.

Работа с импортированными клипами

Вы можете переименовывать и удалять импортированные в документ Flash видеоклипы так же, как и любые другие образцы. Как это делается, описывалось в *главе 10*.

Также вы можете вызвать на экран диалоговое окно **Embedded Video Properties** (см. рис. 16.6), выбрав пункт **Properties** в контекстном или дополнительном меню окна библиотеки. Ниже мы подробно рассмотрим все не описанные еще нами элементы управления, находящиеся в этом окне. Имейте в виду, что все эти элементы доступны только для внедренных клипов.

Иногда бывает нужно отредактировать импортированный клип. Сам Flash такой возможности не предоставляет — следует пользоваться другими программами, например, мощным пакетом Adobe Premiere или весьма хитроумным и очень капризным, но зато бесплатным редактором VirtualDub. Правка видеоклипа осуществляется очень просто: исходный файл, хранящий этот клип, загружается в программу, правится и сохраняется. После этого, если клип был внедрен, нужно обновить его средствами Flash. Для этого служит

кнопка **Update**. При ее нажатии на экране появится стандартное диалоговое окно открытия файла **Windows**. Найдите требуемый файл и нажмите кнопку открытия файла этого диалогового окна.

Обновить импортированный клип можно и другим способом. Не открывая окна **Embedded Video Properties**, просто выделите в списке образцов нужный и в контекстном или дополнительном меню выберите пункт **Update**. На экране появится диалоговое окно **Update Library Items** (см. рис. 10.16). Далее включите флажки против требуемых файлов в списке импортированных клипов и нажмите кнопку **Update**. После обновления образцов нажмите кнопку **Close**.

С помощью кнопки **Import** вы можете импортировать на место существующего клипа другой. При нажатии этой кнопки на экране появится стандартное диалоговое окно открытия файла **Windows**. Найдите нужный файл и нажмите кнопку открытия.

Нажав кнопку **Export**, вы можете сохранить видеоклип в формате *Macromedia Flash Video*. Это особый формат видеоданных, сжатых с помощью алгоритма Sorenson Spark. Впоследствии можно импортировать сохраненный в этом формате клип в другой документ Flash, причем такой импорт будет выполнен быстрее, т. к. Flash уже не нужно будет перекодировать клип.

Для связанных клипов вы можете изменить имя файла, где хранится этот клип. Для этого точно так же вызовите диалоговое окно **Embedded Video Properties**. Для связанных клипов оно имеет несколько другой вид (рис. 16.7). Нажмите кнопку **Set Path**. На экране появится стандартное диалоговое окно открытия файла **Windows**, в котором вам остается выбрать подходящий файл.

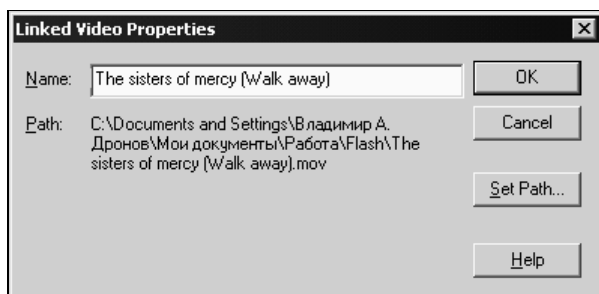


Рис. 16.7. Диалоговое окно **Embedded Video Properties**, показывающее параметры связанного клипа

Изменение экземпляра клипа

Иногда бывает необходимо заменить один экземпляр клипа, находящийся на рабочем листе, на другой, основанный на другом образце, сохранив при

этом все преобразования, выполненные над этим экземпляром. Для такого случая в редакторе свойств имеется кнопка **Swap** (см. рис. 10.6). Выделите на рабочем листе нужный клип и нажмите эту кнопку. На экране появится диалоговое окно **Swap Embedded Video** (рис. 16.8).



Рис. 16.8. Диалоговое окно **Swap Embedded Video**

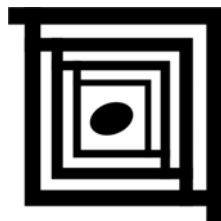
В списке, занимающем большую часть этого окна, перечислены все импортированные к этому времени в текущий документ Flash клипы. Выберите требуемый клип, после чего в небольшом поле предварительного просмотра, расположенном левее и выше списка, будет показано его содержимое. После этого вам останется только нажать кнопку **OK**, чтобы выполнить смену экземпляра клипа, или кнопку **Cancel** для отказа от этого.

Имейте в виду, что новый экземпляр унаследует все преобразования старого экземпляра. Также помните, что если вы заменили один экземпляр другим, основанном на другом образце, в ключевом кадре анимации, то вам потребуется сделать то же самое и во всех остальных ключевых кадрах этой анимации, в которых присутствует этот экземпляр. (Об анимации и ключевых кадрах см. главу 12.)

Внимание!

К сожалению, текущая версия Flash не позволяет менять внедренный клип на связанный.

Глава 17



Работа со звуком

История повторяется.

Когда-то кино было "великим немым". Люди приходили в кинотеатры, чтобы смотреть на крошечном экране черно-белые дергающиеся картинки под музыку разболтанного пианино. Все это было в новинку, как и первые фанерные самолеты, и первые телефоны, в которые нужно было кричать во все горло, как и четыре параллельные линии, показанные на экране первого телевизора. Все это было чудом, недоступным пониманию широких масс, а оттого — вдвойне притягательным.

Сейчас кино перестало быть чудом, также как воздушные сообщения, телефония и телевидение. Чудо превратилось в бизнес, а что может быть прозаичнее. Люди разговаривают по мобильникам, летают на самолетах, смотрят дома телевизоры, а кино да что о нем говорить — кинотеатры практически умирают. Никто не хочет ходить в кино: его заменило домашнее видео, VideoCD, DVD и DivX. (Говорят, в столицах ситуация лучше, чем в провинции, но кто его знает.) Настанет день, когда чудо умрет окончательно, уступив место чему?

Единственное, что никогда не умрет, — это магия киноплёнки. Магия черно-белых прыгающих лиц тех, кого давно с нами нет...

Когда-то компьютеры были большими и медленными, а перерабатывать умели только одни перфокарты или перфоленты в другие такие же перфокарты или перфоленты. Никто и в мыслях не мог допустить, что через несколько десятков лет их далекие потомки станут игрушками для пресыщенных обывателей и их детишек. Как же — такая большая и сложная техника финансировалась военными, очередь была расписана на недели вперед, какие еще игрушки?!!

Но прошли годы, и компьютеры действительно стали игрушками. В самом деле, если в доме есть компьютер, то обязательно найдутся и CD с игрушками и фильмами (если, конечно, компьютер достаточно мощный, а владелец — не законченный трудоголик). Более того, зачастую этот компьютер используется только для игр и просмотра фильмов. Конечно, владелец

пытается заняться чем-то полезным на своем "железном друге", но эти желания так и не претворяются в действительность.

Когда-то Интернет был сплошь текстовым, разбавленным редкими вкраплениями корявых рисунков. В те времена Всемирной Сетью пользовались, в основном, только ученые, военные и государственные чиновники, а что разве нужны этим сухарям дизайнерские изыски? Первым интернетчикам вполне хватало текстовой информации и файлов, никто из них и в страшном сне не мог предвидеть, для чего будет использоваться Сеть через десять лет. Как же можно во время работы качать музыку или смотреть интерактивное видео, когда надо же, елки-палки, работать — за работу казна деньги платит, а не за музыку и видео!

Теперь да что рассказывать об этом — сами зайдите в Сеть или хотя бы вспомните, зачем вы заходили туда последний раз! Даже если у вас и было в мыслях найти что-то действительно для дела, вы точно не забывали и о развлечениях. А какие развлечения сейчас наиболее доступны рядовому пользователю Сети? Конечно, музыка. (К сожалению, видео могут смотреть только обладатели достаточно быстрых каналов, которых пока немного. Но со временем предпочтения большинства могут измениться.)

История повторяется. И кино, и компьютеры, и Интернет из "великих немых" превратились в "великих болтунов".

Звук. Аудио. Музыка. Вот об этом и пойдет речь в этой главе.

Представление звуковой информации

Сначала поговорим о том, каким образом в компьютерах хранится звуковая информация. И также опишем популярные форматы представления этой информации и их поддержку со стороны Flash.

Кодирование и хранение звуковых данных

Для кодирования звука и записи его в файл применяются три различных способа. Все эти способы мы сейчас кратко рассмотрим.

Первый способ — кодирование и запись в файл собственно звука. В этом случае к компьютеру подключается микрофон или иной источник звука, и сигнал с него с помощью специальных программ записывается в файл. Такой способ записи звука применяется в подавляющем большинстве случаев. Например, популярные форматы WAV, MP3, VQF, WMA — все это форматы хранения звуковых данных.

Преимуществом такого подхода к хранению звука является точное его воспроизведение на любом устройстве. Например, вы можете переписать в формат MP3 любимые кассеты, и ваша запись будет нормально проиграна

на любом компьютере или носимом МРЗ-проигрывателе, хотя, конечно, с разным качеством. Также достоинствами являются возможность записи любых звуков (голоса, природных и искусственных шумов, любых музыкальных инструментов) и легкость обработки (подавления шумов, нормализации и др.). Недостаток, пожалуй, один: большой объем получающегося в результате записи звука массива данных.

Чтобы уменьшить массив звуковых данных, применяют различные алгоритмы сжатия. Это может быть сжатие как без потерь, так и с потерями. Как правило, без потерь сжимают звук, предназначенный для дальнейшей обработки, а с потерями — предназначенный для распространения (например, известный формат МРЗ). Иногда звук не сжимают вообще.

Кодирование и запись звука — самый распространенный способ хранения звуковых данных. В частности, он используется для распространения музыки на компакт-дисках и в файлах. На компакт-дисках записывается несжатый звук, для распространения в файлах применяются различные алгоритмы сжатия. Также этот способ записи звука служит для создания звукового сопровождения к видеоклипам.

Второй способ — хранение в файлах только команд для воспроизведения звука. При проигрывании таких файлов эти команды читаются и выполняются либо специальной программой-проигрывателем, либо соответствующим образом оснащенной звуковой картой компьютера. Для создания таких файлов используются специальные программы или музыкальные синтезаторы, опять же в совокупности со специальными программами.

Для воспроизведения собственно звука проигрыватели файлов и звуковые карты имеют примеры звучания реальных инструментов или различных реальных или синтезированных звуков, называемые *инструментами* или *сэмплами*. Для удобства загрузки и управления инструменты объединяются в *банки инструментов*. Некоторые очень старые программы и звуковые карты применяют для вывода звука метод *частотного синтеза*, но качество такого звука очень плохое.

Достоинством второго способа является компактность получающихся файлов. В самом деле, чтобы записать пару байт, составляющих команду, много места не нужно. Недостатки: невозможность записи любых звуков (только тех, для которых в используемом банке инструментов есть соответствующие сэмплы) и зависимость воспроизводимого звука от используемой программы, звуковой карты и банка инструментов.

Второй способ используется практически для тех же целей, что и первый. С его помощью можно записывать и распространять музыку, хранить репетиционные записи (они занимают немного места), помещать музыкальное сопровождение в различные программы и на Web-сайты. Самые распространенные форматы записи звука вторым способом: MIDI и KAR.

Третий способ — небольшая модификация второго. А именно: в файле сохраняются не только команды для воспроизведения звука, но и используемые для этого сэмплы. Фактически каждый такой файл содержит в себе свой собственный банк инструментов. Такой формат записи данных часто называют *гибридным*.

Третий способ объединяет достоинства и недостатки первого и второго способов. Файлы, созданные третьим способом, адекватно воспроизводятся на любом оборудовании и любой программой, они имеют размер, меньший чем файлы, созданные первым способом, но больший, чем файлы звуковых команд.

Как ни странно, но третий способ записи звука не снискал особой популярности. В настоящий момент его используют только особые "малобюджетные" программы создания музыки, называемые *трекерами*. Файлы, создаваемые с помощью трекеров, называются *трекерными модулями*. Существует целая подпольная трекерная субкультура, живущая своей особой жизнью, но почти не известная "в миру". Часто, говоря о форматах записи звука и программах для создания музыки, трекеры даже не упоминают. И, разумеется, не много на свете есть программ, поддерживающих трекерные модули.

Форматы звука, поддерживаемые Flash

Итак, с основными принципами записи звука мы познакомились. Теперь настала пора узнать о форматах звуковых данных, поддерживаемых Flash.

WAV

Формат WAV (от английского wave — волна) был создан фирмой Microsoft в качестве стандартного формата хранения звуковых данных в операционной системе Windows. К настоящему времени получил широчайшее распространение для записи звука, предназначенного для последующей обработки, для хранения сэмплов, помещения звуковых данных в различные программы, на Web-сайты и т. п. Поддерживается абсолютно всеми программами записи и обработки звука, за исключением самых специализированных.

Файл формата WAV имеет "говорящее" расширение wav. Звуковые данные могут быть сжаты с использованием любого алгоритма, для которого в системе установлен кодек, в том числе, популярным алгоритмом MPEG 1 уровень 3, или вообще не сжаты.

Судя по всему, формат WAV будет применяться и в дальнейшем. Пока что никаких причин отказываться от него нет. И все же, для распространения музыки он применяется очень редко.

MP3

Формат MP3 был создан для распространения музыкальных файлов, сжатых по алгоритму MPEG 1 уровень 3. В настоящее время, похоже, стал самым популярным форматом звуковых файлов в Интернете. Поддерживается абсолютно всеми программами записи и обработки звука, за исключением некоторых узкоспециализированных.

Файл этого формата имеет "говорящее" расширение mp3. Звуковые данные всегда сжимаются с помощью алгоритма MPEG 1 уровень 3, другие алгоритмы не допускаются. Файл MP3 также может хранить дополнительные данные, называемые *тегами*. В частности, с помощью тегов сохраняются имя исполнителя, название композиции, название альбома, год записи альбома и жанр музыки.

Этому формату, похоже, уготована долгая жизнь. Несмотря на то, что многие фирмы-распространители музыкальной продукции настроены против него, MP3 полностью завоевал Интернет и сдавать завоеванные позиции не собирается. Более того, этот формат не собирается уступать и более новым форматам записи звука (VQF, OGG Vorbis, MS WMA и др.), хотя более новые разработки обеспечивают лучшее качество звука при значительно меньшем размере файла.

Другие форматы звука, поддерживаемые Flash

В табл. 17.1 перечислены другие форматы звуковых файлов, поддерживаемые Flash.

Таблица 17.1. Список форматов звуковых файлов, которые можно импортировать во Flash

Название формата	Расширение файлов
AIFF	aif
Apple QuickTime*	mov
Sun AU	au

* Имеются в виду файлы QuickTime, содержащие только звук.

Внимание!

Flash не поддерживает непосредственно эти форматы. Чтобы импортировать звуковой файл одного из этих форматов, вам нужно будет установить на свой компьютер пакет Apple QuickTime версии 4 или более поздней.

Форматы звука, не поддерживаемые Flash

Кроме звуковых форматов, поддерживаемых Flash, мы также рассмотрим форматы, которые им не поддерживаются. Это пригодится вам в дальнейшей работе: встретив файл, записанный в одном из таких форматов, вы будете знать, что с ним делать. А именно, перекодировать его в один из поддерживаемых форматов, используя специальное программное обеспечение.

WMA

Разработан фирмой Microsoft в самом конце 90-х годов как часть инициативы Windows Media (создание набора аппаратных и программных средств, направленных на улучшение мультимедийных возможностей современных Windows-совместимых компьютеров). Само название этого формата переводится как Windows Media — Audio. Изначально формат WMA предназначен для постепенной смены форматов WAV и, в идеале, MP3, по сравнению с обоими этими форматами WMA обеспечивает лучшее качество звука и меньший размер файла.

Аудиофайлы, сохраненные в этом формате, имеют расширение wma. Существует также особая разновидность этого формата — WMV (Windows Media — Video), предназначенная для хранения видео, такие файлы имеют расширение wmv или asf. По сравнению с WAV и MP3 формат WMA имеет различные дополнительные возможности, в частности, средства защиты от несанкционированного копирования.

В настоящее время этот формат стремительно набирает популярность. Но серьезно поколебать позиции формата MP3 ему пока не удастся. Хотя в будущем, благодаря поддержке крупнейших фирм-распространителей музыки, он вполне может вытеснить MP3.

RealMedia

Разработан фирмой RealNetwork в середине 90-х для распространения звуковой информации через Интернет. Также часто встречается название RealAudio. Подвергался неоднократным усовершенствованиям. В настоящее время последней версией является 8.0.

Файл формата RealMedia имеет расширение ra. (Существует также разновидность этого формата для хранения видео, файлы, записанные в этом формате, имеют расширение rm или smil.) Для сжатия данных используется одноименный алгоритм. Степень сжатия весьма велика, и достигаемое при сжатии качество звука также достаточно высоко.

В настоящее время удерживает довольно большую долю рынка "сетевого аудио" и терять ее не собирается. В основном, применяется для распространения музыки и трансляции так называемого "интернет-радио".

MIDI

Собственно, это не только и не столько формат музыкальных файлов, сколько стандарт обмена информацией между различным музыкальным оборудованием и программным обеспечением. Был разработан в начале 80-х годов, когда получили распространение цифровые музыкальные синтезаторы, в виде стандарта *General MIDI*. Используется до сих пор, наряду с различными "расширенными" форматами (MIDI XG и др.).

Файл этого формата имеет расширение `mid`, `midi` или `gmi`. Он содержит набор команд воспроизведения звука для программного или аппаратного проигрывателя MIDI (то есть, имеет место второй способ записи звуковых файлов). Из-за этого MIDI-файлы имеют очень маленький размер.

В настоящее время удерживает относительно небольшую нишу — распространение музыки по Сети и создание фонового музыкального сопровождения для Web-сайтов.

Трекерные модули

Первые трекеры появились в начале 80-х годов, тогда же появились и первые форматы сохранения трекерной музыки. В настоящее время популярны, в основном четыре формата модулей, названные так же, как программы, в которых они были созданы:

- ❑ SoundTracker (расширение файла — `mod` или `mdz`);
- ❑ FastTracker (расширение файла — `xm` или `xmz`);
- ❑ ImpulseTracker (расширение файла — `it` или `itz`);
- ❑ ScreamTracker 3 (расширение файла — `st3` или `stz`).

Каждый такой файл содержит как набор команд воспроизведения звука, так и набор сэмплов для их проигрывания (то есть, файл создан третьим способом).

Трекеры (и, соответственно, трекерные модули) сегодня — это андерграунд, "подполье" современной музыки. За пределами этого "подполья" они практически не известны, что, надо сказать, вполне устраивает самих музыкантов-трекерщиков. Есть, правда, единичные примеры использования трекерной музыки в играх (например, популярная компьютерная игра "Unreal") и, по слухам, исполнения по радио.

Импорт звука и работа с ним

На этом теория закончилась. Наступило время практических занятий.

Импорт звука

Для того чтобы импортировать в документ Flash звуковой файл, выберите пункт **Import** в меню **File** или нажмите комбинацию клавиш `<Ctrl>+<R>`.

Вы также можете выбрать пункт **Import to Library** в меню **File**. В любом случае, на экране появится стандартное диалоговое окно открытия файла **Windows**. Найдите нужный файл и нажмите кнопку открытия файла этого диалогового окна. После довольно продолжительного процесса импорта, в течение которого Flash выведет на экран индикатор прогресса, файл будет импортирован.

Импортированные звуковые файлы помещаются в библиотеку в виде образцов-звуков. Таким образом, вы можете создавать сколько угодно экземпляров этого звука, и размер файла Flash при этом не увеличится.

Вместо того чтобы искать нужные звуковые файлы или создавать их самим, вы можете воспользоваться библиотекой общего использования **Sounds**, предоставляемой с Flash. Просто выберите пункт **Sounds** в подменю **Common Libraries** меню **Window** — и в появившемся на экране окне библиотеки вы найдете достаточное количество звуковых образцов для своих первых фильмов.

Использование звука в фильме

Импортированный звук используется в фильме точно так же, как и любые другие образцы, содержащиеся в библиотеке. Вы создаете для звука отдельный слой, помещаете в него экземпляр нужного образца-звука, создаете нужной длины последовательность кадров — и все! Возможно также, вы захотите задать для отдельных кадров этой последовательности какие-то эффекты: запустить проигрывание звука с этого кадра или, наоборот, остановить его.

Создание нового слоя мы описывать не будем — об этом достаточно подробно было сказано в *главе 15*. Желательно, чтобы в рабочем слое не было никакой графики. Вы, конечно, можете поместить несколько экземпляров звука в один слой, но это не рекомендуется. Будет лучше, если вы поместите каждый звук в своем слое: тогда у вас будет больше возможностей по управлению используемыми в фильме звуками.

Поместите в слой экземпляр образца-звука. Создайте последовательность кадров нужной длины. Для этого выделите пустое место на временной шкале, где должна заканчиваться ваша последовательность кадров, и выберите пункт **Frame** в меню **Insert**, пункт **Insert Frame** в контекстном меню выделенного пустого места или просто нажмите клавишу <F5>. Ваша последовательность кадров примет вид такой, как на рис. 17.1. Назовем ее *звуковой дорожкой*.

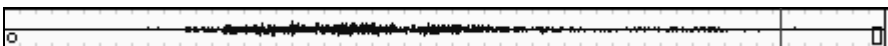


Рис. 17.1. Звуковая дорожка

Взгляните теперь на редактор свойств. В правой его части находится набор элементов управления, с помощью которых вы можете задать дополнительные параметры звукового сопровождения (рис. 17.2). (Если вы его почему-то не видите, проверьте, выделен ли кадр, содержащий звук.) Давайте рассмотрим все эти элементы управления.

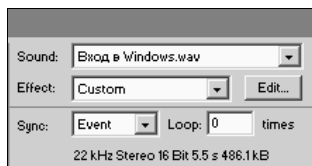


Рис. 17.2. Элементы управления для задания параметров звукового сопровождения фильма

С помощью раскрывающегося списка **Sound** вы можете выбрать звуковой образец, который будет проигрываться в данной последовательности кадров. Выбор звука в этом списке — альтернатива создания экземпляра звукового образца перетаскиванием из окна библиотеки. Пользуйтесь им, если окно библиотеки у вас скрыто. Если же вы не хотите отменить проигрывание звука, выберите в этом списке пункт **None**.

С помощью раскрывающегося списка **Effect** вы можете задать несложный спецэффект, который будет применен Flash к проигрываемому звуку. Этот список доступен только тогда, когда вы выберете в списке **Sound** какой-либо образец. Он содержит восемь пунктов, которые мы сейчас подробно опишем.

Если выбран пункт **None**, то к проигрываемому звуку не будет применено никакого спецэффекта, т. е. звук будет проигран "как есть".

При выборе пунктов **Left Channel** или **Right Channel** звук будет проигран соответственно только в левом или правом канале. Это значит, что звучать он будет только в левой или правой колонке компьютера.

Если выбран пункт **Fade Left to Right**, то звук при проигрывании будет плавно "перетекать" из левого канала в правый. Пункт **Fade Right to Left** вызывает обратное "перетекание" — из правого канала в левый. Это может быть забавным, если вы создали элемент, который перемещается по рабочему листу слева направо или справа налево — звук будет "перемещаться" за ним следом.

Когда выбран пункт **Fade In**, громкость звука будет плавно увеличиваться во время проигрывания. Это может быть полезно, например, при проигрывании заставки вашего фильма. Пункт **Fade Out** выполняет обратное действие — плавное уменьшение громкости звука, что может пригодиться при завершении фильма.

Пункт **Custom** позволит вам самим задать, как будет изменяться громкость и местоположение источника звука в пространстве (*панорамирование*) во время его проигрывания. Как это делается, мы рассмотрим чуть позже.

С помощью раскрывающегося списка **Sync** вы можете задать, как будет проигрываться звук. В этом списке содержится четыре пункта, которые мы сейчас рассмотрим.

Если выбран пункт **Event**, то звук начнет проигрываться, как только Flash проиграет первый кадр содержащей его последовательности. Звук будет проигран до конца, несмотря на то, кончилась ли последовательность кадров, и проигрывается ли еще сам фильм. Такие звуки в терминологии Flash называются *сигналами* (event). Если в одно и то же время проигрывается несколько сигналов, то они будут автоматически смикшированы и все равно проиграются до конца.

При использовании сигналов учтите один момент. Дело в том, что сигнал будет проигран только тогда, когда массив его звуковых данных будет полностью загружен. Поэтому, если вы собираетесь выкладывать свои фильмы в Интернет, не делайте сигналы слишком длинными, иначе пользователю, решившему просмотреть ваше творчество, придется очень долго ждать.

Если выбран пункт **Start**, то звук будет проигран так же, как и при выбранном пункте **Event**. За одним исключением: если такой же звук уже проигрывается в другой последовательности кадров, этот проигран не будет. Фактически, это такой же сигнал, но проигрываемый в данный момент времени только один раз.

Выбор пункта **Stop** останавливает проигрывание любого звука. Вы можете использовать это, если хотите заставить замолчать какой-нибудь сигнал. Для этого просто создайте новый пустой ключевой кадр сразу же за последовательностью, где начал проигрываться этот сигнал, выберите в списке **Sound** нужный звук, а в списке **Sync** — пункт **Stop**.

Если вы выберете пункт **Stream**, то звук будет проигрываться синхронно с проигрыванием всей анимации. Такой звук называется *потокowym* (stream). Когда анимация закончится (все последовательности кадров будут проиграны до своего последнего кадра), закончится и потоковый звук. Примером потокового звука может служить голос какого-либо персонажа вашего фильма.

Потоковые звуки проигрываются прямо во время загрузки. Проигрыватель Flash начинает проигрывание потокового звука, как только загрузит достаточное для этого количество данных. А во время проигрывания он подгружает остальные данные. Иногда, чтобы синхронизировать потоковый звук и анимацию, проигрыватель Flash идет даже на пропуск кадров.

Важное отличие потокового звука от сигнала в том, что все потоковые звуки микшируются при экспорте фильма. Сигналы же микшируются во время проигрывания фильма, и это правильно — в самом деле, мало ли какой сигнал может быть проигран при воспроизведении. С потоковыми звуками все ясно заранее при создании фильма, поэтому Flash подготавливает их к проигрыванию уже при экспорте, чтобы снять с проигрывателя Flash часть работы.

Поле ввода **Loop** служит для задания количества повторений звука. Изначально там стоит 0, что означает проигрывание один раз или, если хотите, ноль повторений.

Если вы хотите, чтобы какой-то звук проигрывался непрерывно, поступите следующим образом. Выясните продолжительность всего фильма, для чего выделите во временной шкале самый последний кадр вашей анимации и посмотрите на строку статуса, где показывается прошедшее с начала анимации время. После этого поделите полученное время на продолжительность нужного звука, а частное от этого деления введите в поле ввода **Loop**. Например, если вы хотите зациклить пятнадцатисекундный звук, а длина вашего фильма равна 15 минутам, вам нужно будет ввести в это поле число 60.

Очень хорошая идея — для создания фонового музыкального сопровождения взять короткий фрагмент и зациклить его так, чтобы он "покрыл" весь фильм. Таким образом вы можете сильно уменьшить размер результирующего SWF-файла.

И последнее. Не зацикливайте потоковые звуки. Иначе Flash автоматически добавит к вашей последовательности столько кадров, сколько длится каждое повторение этого потокового звука. И ваш фильм сильно вырастет в размерах, что вряд ли пойдет ему на пользу.

Правка звука средствами Flash

Выше мы упомянули, что, выбрав в списке **Effect** пункт **Custom**, вы можете задать некоторые параметры звука, в частности то, как будут изменяться его громкость и панорамирование во время проигрывания. Вы также можете сделать это, нажав кнопку **Edit**, расположенную правее списка **Effect** (см. рис. 17.2). А собственно правка звука выполняется в диалоговом окне **Edit Envelope** (рис. 17.3), которое появится на экране после любого из этих действий.

Правка выполняется с помощью двух больших *графиков*, расположенных соответственно в верхней и нижней частях этого диалогового окна. Верхний график позволяет задать изменение громкости, а нижний "отвечает" за панорамирование. Кроме того, в левом и правом нижнем углах этого окна находятся наборы небольших кнопок, которые весьма помогут вам в работе.

Но рассмотрим все по порядку.

Давайте посмотрим на верхний график, график громкости. Вы видите, что на нем схематично отображен ваш звук (который вы выбрали в списке **Sound** редактора свойств). Если вы пользовались программами записи звука, хотя бы стандартно поставляемым с Windows Фонографом, вы знаете, что это такое. Если нет, то запомните, что таким образом Flash схематично представляет ваш звук, большие "всплески" на этом графике представляют большую громкость, маленькие "всплески" — малую громкость.

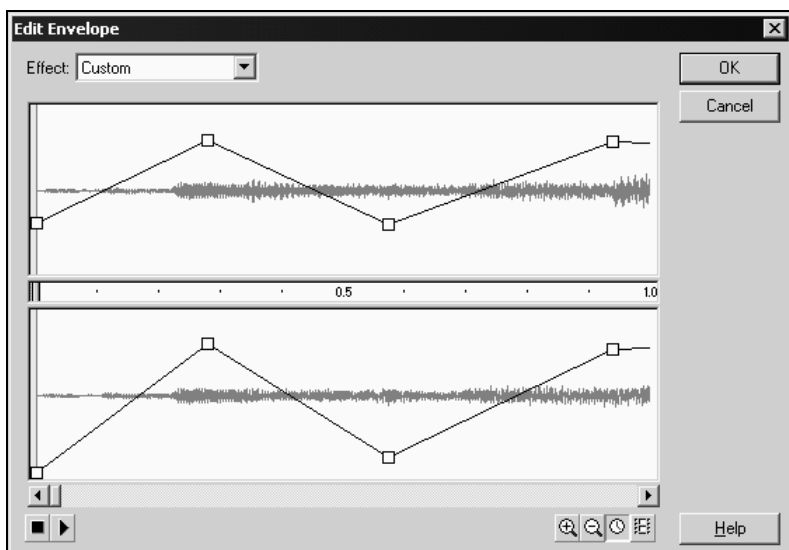


Рис. 17.3. Диалоговое окно **Edit Envelope**

Аналогичное изображение представлено и в нижнем графике. Только оно является схемой позиционирования звука в двумерном пространстве: верхняя половина этого графика представляет левый канал стереозвука, а нижняя — правый канал. Все почти так же, как и в графике громкости, только "всплески" вверх обозначают наличие звука в левом канале, "всплески" вниз — в правом.

Если вы знакомы с черчением, то можете представлять график громкости как вид вашего звука сбоку, а график панорамирования — как вид сверху.

Между этими графиками находится временная шкала, проградуированная в секундах. Вы можете изменить градуировку с секунд на кадры, щелкнув четвертую слева кнопку-переключатель в группе, расположенную в нижнем правом углу окна (см. рис. 17.3), эта кнопка имеет изображение киноплёнки. Чтобы вернуть секунды, щелкните третью слева кнопку-переключатель (с изображением часов). Эти две кнопки имеют зависимую фиксацию, т. е. одновременно может быть включена только одна из них.

Первая и вторая слева кнопки в этой группе позволяют вам соответственно увеличить или уменьшить масштаб графика. Для этого вам нужно просто щелкнуть одну из них.

В самом низу, ниже графика панорамирования, находится полоса прокрутки. С ее помощью вы можете прокручивать оба графика сразу. Серые области на графике слева и справа обозначают соответственно начало и конец вашего звука.

Кроме того, в верхнем левом углу этого диалогового окна находится раскрывающийся список **Effect**. Он дублирует своего "коллегу" в редакторе свойств.

Теперь рассмотрим, как выполняется правка звука.

Задание изменения и громкости, и панорамирования выполняется с помощью так называемой *огibaющей*. Это тонкая черная линия, хорошо заметная на графике. (Каждый график имеет свою *огibaющую*.) С помощью ее вы графически задаете, как будет "взлетать" или "падать" громкость вашего звука и в какую сторону он будет "влиять" при проигрывании (то есть, изменение панорамирования). Это делается прямо на соответствующем графике, просто и наглядно.

Чтобы переместить какую-либо точку *огibaющей* на графике, щелкните по ней мышью. В этом месте появится довольно крупный белый квадрат — *маркер огibaющей*. Этот маркер задает точку, за которую вы можете "тащить" мышью. Просто перетащите его на нужное место графика, и *огibaющая* примет нужную форму. *Огibaющая* верхнего графика задает изменение громкости звука, а *огibaющая* нижнего графика — его панорамирование.

Теперь еще раз посмотрите на временную шкалу. В крайних левой и правой ее точках вы увидите небольшие серые *маркеры начала* и *конца* вашего звука. С их помощью вы можете задать точки, где будет начато и закончено воспроизведение вашего звука. И эти точки совсем не обязаны совпадать с изначальными началом и концом импортированного звукового файла. Таким образом, вы можете воспроизвести этот звук не с начала, а, скажем, с середины, и не до конца, не производя в нем никаких изменений и не пользуясь внешними программами. В частности, с помощью этих маркеров, вы можете убрать ненужную тишину в начале и конце звукового фрагмента.

В нижнем левом углу находятся две кнопки, позволяющие вам проверить полученный в результате правки звук. Правая кнопка (с направленной вправо треугольной стрелкой) запускает проигрывание, а левая (с черным квадратом) — останавливает его.

Задайте требуемую *огibaющую* для громкости и панорамирования. Прокрутите графики, если нужно. При необходимости задайте начало и конец воспроизведения звука. Прослушайте полученный результат, исправьте его и нажмите кнопку **ОК**. Если же вы не хотите сохранять результаты вашей правки, нажмите кнопку **Cancel**.

Работа с импортированными звуками

Вы можете переименовывать и удалять импортированные в документ Flash звуковые образцы так же, как и любые другие образцы. Можно также разделить любой образец-звук для совместного доступа к нему. Как это делается, описывалось в *главе 10*.

Также вы можете вызвать на экран диалоговое окно **Sound Properties** (рис. 17.4), выбрав пункт **Properties** в контекстном или дополнительном меню окна библиотеки. Основную часть элементов управления этого окна мы рассмотрим ниже, а сейчас коснемся только некоторых из них.

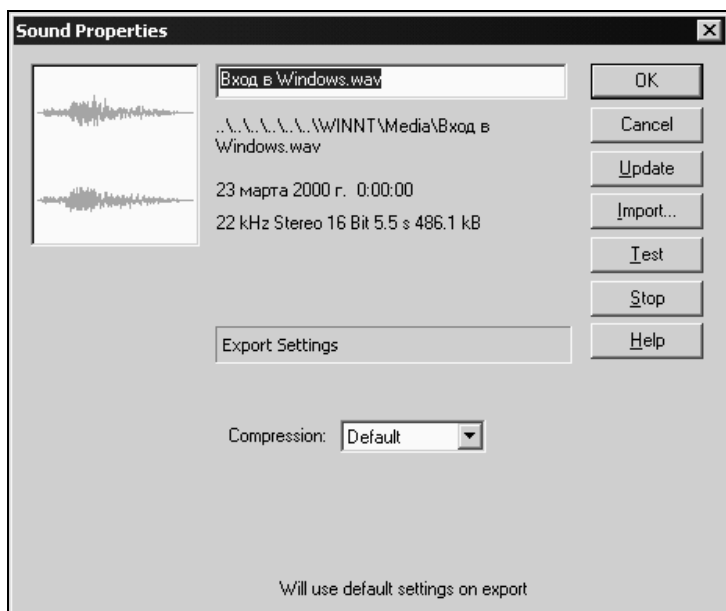


Рис. 17.4. Диалоговое окно **Sound Properties**

Вы можете изменить имя образца-звука, введя его в поле ввода, расположенное в верхней части диалогового окна. Изначально оно совпадает с именем импортированного звукового файла.

С помощью кнопки **Import** можно импортировать на место существующего звука другой. При нажатии этой кнопки на экране появится стандартное диалоговое окно открытия файла Windows. Найдите нужный файл и нажмите кнопку открытия.

Нажав кнопку **Test**, вы можете прослушать импортированный клип. Заметьте, что здесь он будет проигран в своем изначальном виде. Чтобы остановить проигрывание, нажмите кнопку **Stop**.

В верхней части диалогового окна **Sound Properties** вы можете увидеть сведения о звуке. А в нижней части этого диалогового окна находятся элементы управления, предназначенные для задания степени сжатия звука при экспорте. Мы рассмотрим их ниже.

Чтобы обновить импортированный звук после его правки во внешней программе, выделите в списке образцов нужный и в контекстном или дополнительном меню выберите пункт **Update**. На экране появится диалоговое окно **Update Library Items** (см. рис. 10.16). После этого включите флажки против требуемых файлов в списке импортированных звуков и нажмите кнопку **Update**. После обновления образцов нажмите кнопку **Close**.

Задание параметров звука

При экспорте фильма, содержащего импортированный звук, Flash преобразует его в свой внутренний формат, после чего, если нужно, выполняет микширование нескольких звуков в один. Разумеется, Flash также выполняет сжатие звука, а иначе размер фильма получится очень большим. Вы можете управлять параметрами сжатия каждого конкретного звука, а именно, степенью и используемым алгоритмом. Рассмотрим, как это делается.

А делается это либо в уже знакомом вам диалоговом окне **Sound Properties** (см. рис. 17.4), либо в особом диалоговом окне **Sound Settings** (рис. 17.5). Вероятно, в окне **Sound Settings** работать удобнее, поэтому его мы рассмотрим подробнее. Впрочем, в окне **Sound Properties** содержатся те же самые элементы управления, так что вы можете пользоваться ими аналогичным образом.

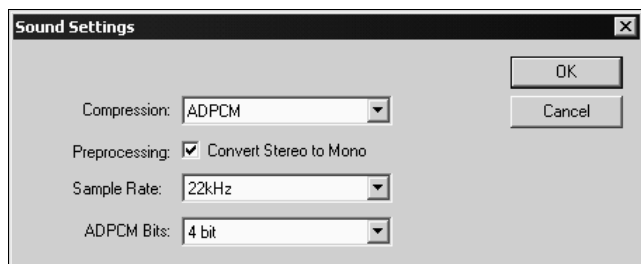


Рис. 17.5. Диалоговое окно **Sound Settings**

Итак, в окне **Sound Settings** вы видите раскрывающийся список **Compression**, с помощью которого вы можете выбрать алгоритм сжатия звука. Список содержит пять пунктов, которые мы подробно рассмотрим ниже.

Если в списке **Compression** выбран пункт **Default**, то к этому звуку будут применены параметры по умолчанию, заданные при экспорте фильма. Эти параметры будут применяться также ко всем звукам, для которых был выбран этот пункт. А поскольку данный пункт выбран изначально, то эти параметры будут применены ко всем звукам, если вы не зададите для них другие.

Если выбран пункт **ADPCM**, то для сжатия этого звука будет применен одноименный алгоритм, разработанный фирмой Microsoft для кодирования коротких звуковых фрагментов. Этот алгоритм не обеспечивает высокой степени сжатия, но сжатые с его помощью звуки могут быть проиграны даже на маломощных компьютерах. Одновременно в диалоговом окне появятся дополнительные элементы управления, позволяющие задать параметры кодирования.

Флажок **Convert Stereo to Mono** позволит вам преобразовать стереофонический звук в монофонический, благодаря чему массив звуковых данных уменьшится в два раза. Этот флажок включен по умолчанию.

Раскрывающийся список **Sample Rate** позволит вам задать частоту оцифровки звука. Эта величина определяет частоту, с которой электронная схема, кодирующая звук, выполняет *дискретизацию* сигнала. Чем выше частота оцифровки, тем выше качество звука. Список **Sample Rate** имеет четыре пункта:

- ☐ **5kHz** — 5 кГц (вполне разборчивая речь);
- ☐ **11kHz** — 11 кГц, минимальная допустимая частота оцифровки музыки (допускается для коротких фрагментов);
- ☐ **22kHz** — 22 кГц, наиболее часто используемая в фильмах Flash частота оцифровки звука, обеспечивающая достаточное его качество;
- ☐ **44kHz** — 44 кГц, частота оцифровки звука, записываемого на звуковых компакт-дисках, очень высокое качество, но и весьма большой объем звуковых данных.

Учтите, что Flash может понизить частоту оцифровки вашего звука, но не в состоянии ее повысить. Например, если вы импортировали звук, оцифрованный с частотой 22 кГц, то вы можете экспортировать его с частотой 5, 11 или 22 кГц, но не 44 кГц.

Раскрывающийся список **ADPCM Bits** позволяет вам косвенно задать ширину потока данных вашего звука. Ширина потока данных характеризует объем данных, отводимых на хранение звука. Этот список содержит четыре пункта: **2 bit**, **3 bit**, **4 bit** и **5 bit**, задающих ширину потока от минимальной до максимальной.

Если в списке **Compression** выбран пункт **MP3**, то этот звук будет сжат с помощью известного алгоритма сжатия MPEG 1 уровень 3. Рекомендуется выбирать его, если вы имеете импортированные звуки, сжатые в этом формате. С помощью появившихся в окне дополнительных элементов управления вы можете задать параметры сжатия.

Если данный звук уже сжат алгоритмом MPEG 1 уровень 3, то будет включен по умолчанию флажок **Use Imported MP3 Quality**. Это значит, что данный звук будет экспортирован с теми же параметрами, что и был импортирован. Чтобы задать другие параметры экспорта, отключите этот флажок.

Раскрывающийся список **Bit Rate** позволит вам задать ширину потока данных вашего звука. Он имеет двенадцать пунктов от **8 kbps** до **160 kbps**, с помощью которых выбирается нужное значение ширины потока в килобитах в секунду. Поэкспериментируйте с разными значениями данного параметра, чтобы подобрать самые подходящие для конкретного звука.

Если вы выбрали ширину потока данных от 20 килобит в секунду и выше, становится доступным флажок **Convert Stereo to Mono**. Он позволит вам преобразовать стереофонический звук в монофонический с двукратным уменьшением массива звуковых данных. Если же ширина потока слишком низка, этот флажок будет включен и недоступен для отключения.

Раскрывающийся список **Quality** позволяет вам задать скорость сжатия и качество результирующего звука. Он имеет три пункта:

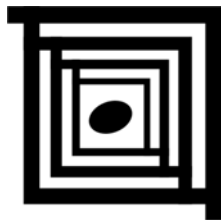
- ☐ **Fast** — быстрое сжатие, но низкое качество;
- ☐ **Medium** — средняя скорость сжатия и среднее качество звука;
- ☐ **Best** — низкая скорость сжатия и высокое качество звука.

Если в списке **Compression** выбран пункт **Raw**, то звук вообще не будет сжат при экспорте. Рекомендуется выбирать его только для кодирования коротких звуковых фрагментов.

Раскрывающийся список **Sample Rate** позволит вам задать частоту оцифровки звука. Он имеет четыре пункта, которые были описаны выше.

Если в списке **Compression** выбран пункт **Speech**, звук будет сжат с помощью особого алгоритма, специально созданного для кодирования речи. Разумеется, выбирать его стоит только для речи "персонажей" вашего фильма. Здесь доступен только хорошо знакомый вам раскрывающийся список **Sample Rate**, позволяющий задать частоту оцифровки звука.

Глава 18



Подготовка к экспорту

Все хорошее когда-нибудь кончается... Увы, таков закон природы! Не успели мы привыкнуть к нашему документу Flash, не успели как следует намучиться с ним, пытаясь заставить все работать, как надо, не успели насладиться чувством власти и могущества, все же заставив его работать, как его уже нужно отдавать заказчику. Как говорил персонаж одного старого мультфильма, "на самом интересном месте"...

У вас осталось совсем немного времени, чтобы проверить, все ли сделано правильно, и нет ли каких ошибок. Не торопитесь, но и не мешкайте. Потратьте эти минуты как можно плодотворнее. Как? Сейчас мы это расскажем.

Прежде всего, вам нужно оптимизировать ваш фильм. Что такое *оптимизация*? Это довольно долгий и подчас трудоемкий процесс по отыскиванию скрытых ошибок и "узких мест" и устранению их. Оптимизация — это задание правильных параметров экспорта импортированной растровой графики и звука, удаление ненужных образцов из библиотеки, "подчистка" рабочего листа, проверка всех встроенных в ваш фильм сценариев на предмет ошибок и их исправление. После правильной и тщательной оптимизации ваш фильм будет выглядеть лучше, проигрываться плавнее, загружаться быстрее, а сценарии будут работать быстро и не засорять понапрасну память компьютера.

Оптимизация — вот чем зачастую пренебрегают начинающие Flash-художники, аниматоры и программисты. Да и не только начинающие...

Flash предоставляет мощные средства по выявлению "узких мест", препятствующих загрузке файла фильма. Воспользуйтесь ими, чтобы выяснить, какая часть фильма будет загружаться быстрее, а какая — медленнее. Располагая такими данными, вы сможете ускорить загрузку фильма, а значит, сделать его просмотр более плавным.

Второе, о чем вам следует позаботиться после оптимизации, — это *доступность* фильма (по-английски *accessibility*). В терминологии Flash — это набор средств, с помощью которых вы можете сделать свой фильм доступным

людям с различными физическими недостатками. В частности, вы можете сделать свои фильмы доступными для слепых и слабовидящих пользователей. Специальное программное обеспечение через звуковую карту компьютера будет зачитывать им то, что находится на экране и в активном на данный момент поле ввода.

Чтобы сделать свой фильм доступным для людей с физическими недостатками зрения, нужно совсем немного усилий. Так что трудоемкость создания доступных (accessible) фильмов в данном случае минимальна. Чтобы немного помочь ближнему своему, которому не повезло в жизни, программисты фирмы Macromedia сделали все, что от них зависело. Теперь очередь за вами.

Оптимизация фильма

Здесь мы опишем средства, предлагаемые Flash для анализа процесса загрузки и проигрывания фильма. Благодаря им вы можете выявить "узкие места" вашего фильма. Также мы перечислим действия, которые помогут вам сделать ваши фильмы компактнее и быстрее.

Профилировщик загрузки

Обычно проигрыватель Flash начинает проигрывание фильма сразу же после того, как загрузит полностью несколько первых его кадров. В дальнейшем загрузка остальных кадров идет одновременно с проигрыванием. Благодаря такому подходу пользователю не нужно ждать, пока весь фильм будет загружен.

Однако, если по каким-то причинам загрузка фильма приостанавливается, проигрыватель Flash не сможет получить данные для дальнейшего проигрывания этого фильма. Тогда проигрывание фильма останавливается, пока не будет получено достаточно данных для его продолжения. Профессиональные "флэшеры" говорят в таком случае, что проигрывание фильма "рваное".

Причин, по которым загрузка фильма может остановиться, много. Не последнюю очередь в них занимают проблемы с аппаратурой и системным программным обеспечением. Например, при загрузке фильма из Интернета модем пользователя может внезапно оборвать соединение, или прокси-сервер организации может не пропустить подозрительный файл. В таком случае говорят о внешних причинах прерывания загрузки, с ними вы ничего сделать не сможете.

Однако вы можете устранить внутренние причины. Таковыми причинами могут быть слишком большое растровое изображение, импортированное в фильм, большой звуковой файл, большой видеоклип и т. п. Как видите, в основном, причины вызваны слишком большими графическими фрагмен-

тами, которые нужно загрузить и отобразить проигрывателю. А на их размеры мы вполне можем влиять.

Но прежде всего, нужно выяснить, на что именно влиять. И в этом нам поможет инструмент Flash, называемый *Профилировщиком загрузки*. С его помощью вы без труда выясните, что именно "тормозит" загрузку вашего фильма.

Чтобы подвергнуть "профилировке" какой-либо фильм, откройте его в среде Flash и выберите пункт **Test Movie** в меню **Control** или нажмите комбинацию клавиш <Ctrl>+<Enter>. Чтобы протестировать только текущую сцену, выберите пункт **Test Scene** в меню **Control** или нажмите комбинацию клавиш <Ctrl>+<Alt>+<Enter>. Как вы знаете, после этого фильм (или его текущая сцена) будет открыт в отдельном окне Flash и проигран.

Если ваш фильм уже экспортирован в распространяемом формате Shockwave/Flash и сохранен в файле, вам следует поступить по-другому. Откройте нужный SWF-файл, воспользовавшись пунктом **Open** меню **File** или комбинацией клавиш <Ctrl>+<O>. Открытый вами фильм также будет проигран в отдельном окне Flash.

Не закрывая окна, в котором проигрывается фильм, откройте меню **Debug**. Вы увидите группу пунктов-переключателей: **14.4 (1.2 KB/s)**, **28.8 (2.3 KB/s)**, **56K (4.7 KB/s)**, **User Setting 4 (2.3 K)**, **User Setting 5 (2.3 K)** и **User Setting 6 (2.3 K)**. С их помощью вы можете выбрать скорость, с которой будет загружаться фильм. Если нужной вам скорости здесь нет, вы сами можете задать ее, выбрав пункт **Customize** в том же меню **Debug**. После этого на экране появится диалоговое окно **Custom Modem Settings** (рис. 18.1), в котором и задаются скорости загрузки фильма.

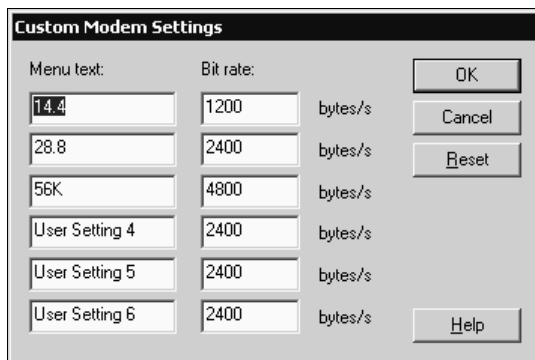


Рис. 18.1. Диалоговое окно **Custom Modem Settings**

Задать нужные скорости загрузки фильма в этом окне очень просто. Оно содержит шесть пар полей ввода, соответствующих каждому из перечисленных выше пунктов меню. В левом поле ввода каждой пары (**Menu text**) зада-

ется текст пункта меню, а в правом (**Bit rate**) — числовое значение скорости загрузки в байтах в секунду. Введите нужные данные и нажмите кнопку **OK**. Если вы передумали задавать новые значения, нажмите кнопку **Cancel**. Если же вы хотите вернуться к значениям по умолчанию, нажмите кнопку **Reset**.

Выбрав скорость загрузки, откройте окно Профилировщика. Для этого включите пункт-выключатель **Bandwidth Profiler** в меню **View** или нажмите комбинацию клавиш <Ctrl>+. Окно, в котором проигрывается фильм, разделится на две части. Сам фильм будет проигрываться в нижней части, а в верхней появятся данные Профилировщика (рис. 18.2). Проверьте, включен ли в меню **View** пункт-переключатель **Streaming Graph**, и, если он не включен, включите его.

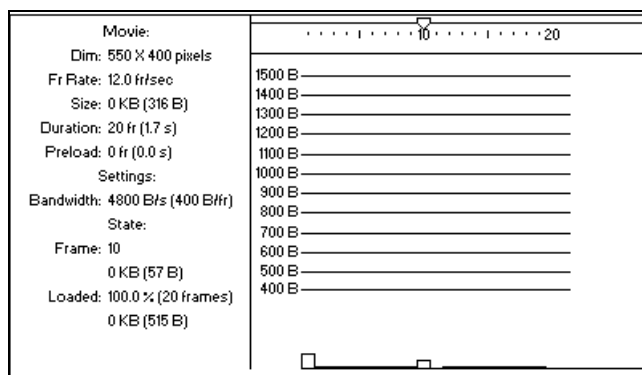


Рис. 18.2. Данные Профилировщика загрузки

Сама верхняя часть окна, где отображаются данные Профилировщика, также разделена на две части. Мы подробно опишем их ниже.

В левой части отображаются данные о фильме. В частности:

- ☐ **Dim** — геометрические размеры фильма в пикселах;
- ☐ **Fr Rate** — частота кадров фильма;
- ☐ **Size** — размер SWF-файла, содержащего фильм;
- ☐ **Duration** — продолжительность фильма в кадрах и секундах;
- ☐ **Preload** — количество кадров, которое должно быть загружено перед тем, как фильм начнет проигрываться, и продолжительность задержки, необходимой для загрузки этих кадров;
- ☐ **Bandwidth** — выбранная вами скорость загрузки фильма;
- ☐ **Frame** — номер текущего кадра и (ниже) размер данных, необходимых для его отображения;
- ☐ **Loaded** — количество загруженных к данному моменту кадров и размер загруженных данных.

Как видите, Профилировщик отображает довольно много данных, которые вам могут помочь в доводке фильма до ума. Но самая ценная информация показывается в правой части.

Там располагается график загрузки фильма. По горизонтальной оси этого графика отложены все кадры вашего фильма, в верхней части графика вы также можете видеть шкалу времени и указатель текущего кадра, показывающий, какой кадр фильма вы сейчас видите. А по вертикальной оси графика отложен размер данных, необходимых для отображения кадра. Соответственно, серые столбики, которые вы видите внизу, показывают, сколько данных требуется для отображения соответствующего кадра.

В нижней части графика вы можете увидеть красную горизонтальную линию. Она показывает предел, при превышении которого проигрыватель Flash не сможет загружать данные достаточно быстро и будет вынужден приостановить фильм. Таким образом, вы без проблем узнаете, какой кадр "тормозит" ваш фильм, и сможете принять необходимые меры.

Вы можете остановить проигрывание фильма, нажав клавишу <Enter> или <Esc> или выбрав пункт **Stop** в меню **Control**, и выделить нужный кадр, просто щелкнув мышью на соответствующей позиции временной шкалы. После этого Flash покажет вам данные по выделенному вами кадру в левой части Профилировщика.

Если вы включите пункт-выключатель **Show Streaming** в меню **View** или нажмете комбинацию клавиш <Ctrl>+<Enter>, в верхней части графика, где находится указатель кадра, будет показан индикатор выполнения загрузки вашего фильма. Таким образом, когда вы запустите фильм на воспроизведение, выбрав пункт **Play** в меню **Control** или нажав клавишу <Enter>, Flash покажет вам, с какой скоростью будет загружаться ваш фильм.

Если вы включите пункт-переключатель **Frame by Frame Graph** в меню **View** или нажмете комбинацию клавиш <Ctrl>+<F>, вид графика изменится. Серые столбики покажут не размер данных каждого кадра, а число кадров, которые могут быть загружены при проигрывании каждого кадра фильма (рис. 18.3). Это также может быть полезно для выявления "узких мест" вашего фильма.

Например, в нашем примере видно, что все кадры фильма будут загружены уже при проигрывании шестого его кадра. Значит, беспокоиться незачем — весь фильм будет загружен и проигран без всяких задержек (если, конечно, у пользователя не возникнет проблема с оборудованием или другим программным обеспечением). Эх, вот бы все фильмы Flash так грузились!..

Чтобы вернуться к обычному режиму показа размера кадров, выберите в меню **View** пункт-переключатель **Streaming Graph** или нажмете комбинацию клавиш <Ctrl>+<G>.

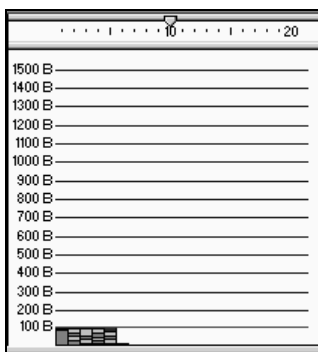


Рис. 18.3. Профилировщик загрузки показывает, сколько новых кадров фильма успеет загрузиться при проигрывании очередного его кадра

Оптимизация фильмов

А теперь настала пора поговорить об оптимизации фильмов. (Надо сказать, что некоторые советы подходят и для неподвижной графики, если вы вдруг решите сохранить ее в формате распространения Shockwave/Flash.)

Твердо запомните одно простое правило: чем длиннее и сложнее фильм Flash, тем больше места он требует на диске, тем дольше он загружается по Сети, тем дольше он обрабатывается перед выводом на экран, тем медленнее выводится. Поэтому следует сделать все, чтобы он стал насколько возможно менее сложным и менее длинным. И делать это нужно еще до его экспорта.

Прежде всего, по возможности для создания множества похожих элементов применяйте образцы. Объединяйте графические фрагменты в группы: это сильно облегчит вам манипуляцию с ними и позволит Flash лучше их оптимизировать. Отделяйте анимированные фрагменты от статичных с помощью слоев. И, по возможности, используйте разделяемые образцы. При всех их недостатках это неплохое решение проблемы больших SWF-файлов.

Старайтесь не создавать в изображении слишком много линий различных типов (точечные, пунктирные и др.) — они требуют больше памяти, чем сплошные линии. (Это относится как к обычным линиям, так и к контурам фигур.) Пользуйтесь только необходимым минимумом мазков "кистью" — они также требуют много памяти. По возможности выполняйте оптимизацию кривых (пункт **Optimize** меню **Modify**, за подробностями обращайтесь к главе 5); благодаря этому они станут проще, а значит, займут меньше памяти. Вообще, старайтесь не использовать слишком сложные кривые.

При выборе цвета старайтесь, чтобы он по возможности совпадал с одним из цветов безопасной палитры Web. Старайтесь обходиться минимумом гради-

ентных заливок — они требуют на 50 байт больше памяти, чем сплошные. Имейте в виду, что чрезмерное увлечение полупрозрачными элементами может замедлить вывод изображения.

Для набора текста применяйте только необходимый минимум шрифтов. Будет лучше, если вы ограничитесь шрифтами-псевдонимами (см. главу 7). Если же вы создали поля ввода или динамические текстовые блоки, воспользуйтесь редактором свойств, чтобы задать набор символов, который будет внедрен в результирующий файл Shockwave/Flash. Помните, что Flash при экспорте внедряет в результирующий файл информацию обо всех использованных в изображении шрифтах, что может сильно увеличить этот файл.

Везде где это возможно заменяйте покадровую анимацию трансформационной. Для создания вложенной анимации используйте образцы-клипы, а не анимированные графические образцы. При создании трансформационной анимации делайте все изменения как можно меньшими — слишком большие изменения увеличивают размер фильма и замедляют его проигрывание. Откажитесь от анимации импортированной растровой графики. По возможности, используйте импортированный звук в формате MP3 — он имеет меньшие размеры.

Пожалуй, самый лучший способ сделать фильм меньше — сделать его проще. Будьте проще — и пользователи вас поймут!

Доступность фильма

Здесь мы поговорим о доступности ваших фильмов. Термином "доступность" или, по-английски, *accessibility*, обозначаются особые свойства компьютеров или программ, позволяющие людям с физическими недостатками, работать с ними. Это вы уже знаете.

Технология "чтение с экрана"

Увы, не всем радоваться яркими красками ваших изображений и фильмов Flash! Несчастные, потерявшие зрение, не смогут увидеть ничего, как бы вы не старались. Вы только представьте себе, что такое — ничего не видеть.

К счастью, не могущие видеть смогут вас услышать. Пользуясь специальными компьютерами, оснащенные специальным программным обеспечением, они все еще могут общаться с окружающим миром. И в ваших силах помочь им в этом.

Как? Сейчас мы все расскажем.

Но сначала давайте рассмотрим, как слепые и слабовидящие могут получить возможность работать с компьютером.

Для этого используется особая технология, называемая очень просто, — *"чтение с экрана"*. Заключается она в следующем. Предположим, за спиной слепого, сидящего за компьютером, стоит помощник, который говорит ему о том, что находится в данный момент на экране и в активных элементах управления, скажем, полях ввода. А слепой, пользуясь специальной клавиатурой, поддерживающей шрифт Брайля, перемещается по элементам управления и вводит в них нужные данные. Не бог весть что, но, по крайней мере, он сможет ввести данные в регистрационную форму почтового сервера и написать письмо. Значит, он сможет общаться с друзьями, а это уже немало!

Роль такого помощника выполняет специальное программное обеспечение, синтезирующее речь на основе считанного из окна или элемента управления текста. Проигрыватель Flash использует это программное обеспечение для вывода имен различных элементов проигрываемого фильма. А выводит-ся звук через звуковую карту, которая ныне стала стандартным компонентом компьютера.

Внимание!

Технологию "чтение с экрана" поддерживает только ActiveX-версия проигрывателя Flash, предназначенная для Web-обозревателя Microsoft Internet Explorer и других программ, поддерживающих элементы ActiveX. Кроме того, не поддерживается доступность для беззаконных режимов вывода фильмов Flash (подробнее см. главу 19).

Поддержка "чтения с экрана" Flash

Как уже говорилось, Flash при создании доступных фильмов и приложений максимально идет навстречу разработчику. В частности, он автоматически обеспечивает доступность следующих графических элементов:

- ☐ текстовых блоков;
- ☐ полей ввода;
- ☐ кнопок;
- ☐ образцов-клипов;
- ☐ всего фильма.

Для текстовых блоков (обычных и динамических) Flash просто произносит их содержимое, например, "Имя пользователя" или "Адрес". Для других элементов все несколько сложнее.

Начнем с того, что поля ввода, кнопки и образцы-клипы, помещенные на рабочем листе, должны иметь уникальные имена. Данные имена необходимы для управления этими элементами из сценариев ActionScript. Эти же имена произносятся программным обеспечением "чтения с экрана" слепому пользователю.

Поэтому, если вы хотите сделать свои фильмы доступными, всегда задавайте имена, по крайней мере, для полей ввода и кнопок. Если же какой-либо элемент, например, поле ввода, не имеет имени, программа прочитает что-то вроде "поле ввода". Не очень-то информативно, не так ли?

Для задания имени графического элемента служит особое поле ввода, находящееся в верхнем левом углу редактора свойств (рис. 18.4). Естественно, оно доступно только тогда, когда на рабочем листе выделен элемент, могущий иметь имя. Это может быть поле ввода, динамический текстовый блок, кнопка или образец-клип. Просто введите в него имя, состоящее только из символов латинского алфавита, цифр и знаков подчеркивания.

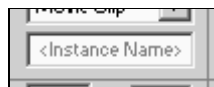


Рис. 18.4. Поле ввода имени графического элемента, расположенное в редакторе свойств

Задайте имена только для тех графических элементов, которые вы хотите сделать доступными. В частности, не стоит давать имена всем образцам-клипам. Дайте имена только тем из их, кто выступает в качестве элемента управления.

Также может быть полезно дать полям ввода, кнопкам и другим элементам управления, расположенным на рабочем листе, текстовые *надписи*. Пример такой надписи показан на рис. 18.5. Когда изображение будет выведено на экран, программное обеспечение "чтения с экрана" зачитает содержимое всех текстовых блоков, и пользователь будет знать, что это изображение содержит. В этом случае может даже не понадобится давать полю ввода имя — Flash "поймет", что вы сделали надпись для элемента управления, а не простой текстовый блок, и прочитает эту надпись пользователю.

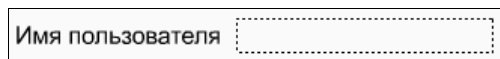


Рис. 18.5. Поле ввода с текстовой надписью

Дополнительные параметры доступности

Вы можете задать дополнительные параметры доступности того или иного графического элемента и всего фильма, воспользовавшись панелью **Accessibility**. Чтобы вызвать ее на экран, выберите пункт **Accessibility** в меню **Window**, нажмите комбинацию клавиш **<Alt>+<F2>** или щелкните кнопку, расположенную в правой части редактора свойств (рис. 18.6). Сама панель **Accessibility** показана на рис. 18.7.



Рис. 18.6. Кнопка вызова панели **Accessibility**



Рис. 18.7. Панель **Accessibility**

Итак, что же вы сможете сделать с помощью этой панели?

Вы можете задать *альтернативное имя* для какого-либо элемента или всего фильма Flash. Такое имя будет произнесено системой "чтения с экрана" вместо того, что вы задали в редакторе свойств. Для этого выберите нужный элемент (для выбора всего фильма щелкните по пустому пространству рабочего листа) и укажите в поле ввода **Name** это имя.

Можно задать дополнительное описание для какого-либо элемента или всего фильма, произносимое вместе с именем. Это описание вводится в область редактирования **Description**.

Для кнопок и полей ввода вы можете задать "горячие клавиши". Это особые комбинации клавиш, при нажатии которых данное поле ввода будет активизироваться, а кнопка — срабатывать. "Горячие клавиши" задаются в поле ввода **Shortcut**. Функциональные клавиши задаются своими названиями: Shift, Ctrl или Alt, обычные клавиши — большими буквами латинского алфавита, а знак "плюс" обозначает, что эти клавиши должны нажиматься одновременно. Например, значение Ctrl+N обозначает, что нужно одновременно нажать клавиши <Ctrl> и <N>.

Вы можете скрыть какой-либо элемент от системы "чтения с экрана". Для этого отключите флажок **Make Object Accessible**. Если же вы по каким-то причинам хотите скрыть весь фильм, щелкните мышью по пустому месту рабочего листа и отключите флажок **Make Movie Accessible**.

К сожалению, для текстовых блоков так сделать нельзя. Для этого вам придется преобразовать обычный текстовый блок в динамический, а уж потом вам станет доступен флажок **Make Object Accessible**, который вам и нужно

отключить. Чтобы превратить текстовый блок в динамический, выберите пункт **Dynamic Text** в раскрывающемся списке вида текстового блока (см. рис. 7.15).

Вы можете скрыть от системы "чтения с экрана" все содержимое фильма или приложения. Для этого щелкните мышью по пустому месту рабочего листа и отключите флажок **Make Child Objects Accessible**.

Вы уже знаете, что если поместить текстовый блок достаточно близко к элементу управления, то текстовый блок станет надписью. Однако вы можете заставить Flash считать такие текстовые блоки обычными текстовыми блоками, не привязанными к элементу управления. Для этого щелкните мышью по пустому месту рабочего листа и отключите флажок **Auto Label**.

Flash также предоставляет вам еще одну интересную возможность. Вы можете задавать разные параметры доступности в разных ключевых кадрах, т. е. менять их во время анимации. Но имейте при этом в виду, что не все программы "чтения с экрана" смогут воспринять такие метаморфозы.

Полезные советы по созданию доступных фильмов

Здесь мы перечислим еще кое-какие полезные советы, которые помогут вам создать доступные фильмы и приложения Flash.

Прежде всего, запомните, что программное обеспечение "чтения с экрана" читает только текст, помещенный в текстовые блоки. Оно не понимает графику. Поэтому, если вы правили какой-либо текст, превратив его в графику пунктом **Break Apart** меню **Modify**, программа "чтения с экрана" прочитать его не сможет — для нее такой текст неотличим от графики. Поэтому либо используйте для описания полей ввода только текст, либо давайте им альтернативное имя и описание, либо давайте дополнительное описание для всего фильма или приложения.

Решите, будете ли вы давать описания для каждого элемента или дадите общее описание для всего фильма или приложения. Если вы считаете, что опишете ваш фильм одной фразой текста, вызовите панель **Accessibility**, щелкните по пустому пространству рабочего листа, отключите флажок **Make Child Objects Accessible** и введите в область редактирования **Description** описание всего фильма.

Постарайтесь не анимировать поля ввода, кнопки и надписи. Некоторые программы "чтения с экрана" некорректно обрабатывают анимированные элементы управления.

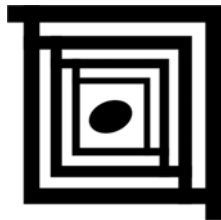
Если вы используете в своем фильме импортированный звук, постарайтесь сделать так, чтобы он не мешал пользователям слушать синтезиро-

ванную программами "чтения с голоса" речь. Не используйте слишком много громких звуков. Лучше сделайте так, чтобы звук, наоборот, помог им воспринимать фильм. Например, включите в фильм его голосовое описание.

Проверьте, можно ли "путешествовать" по элементам управления вашего Flash-приложения, пользуясь только клавиатурой. Помните, что клавиатура — единственный инструмент ввода информации в компьютер, который может использовать слепой пользователь.

Исключите в своем фильме (приложении) ситуации, когда обширный текст показывается на экране очень короткое время. Помните, что в этом случае программа "чтения с экрана" просто не сможет прочитать его пользователю.

Глава 19



Публикация и экспорт анимации

Ну вот и готов наш фильм. Что с ним теперь нужно сделать? Правильно, опубликовать или экспортировать. Точно так же, как мы публиковали и экспортировали статичные изображения в *главе 11*.

Напомним, что такое публикация и экспорт. Во-первых, повторенье — мать ученья. А во-вторых, и публикация, и экспорт анимированной графики имеют ряд особенностей.

Публикацией в терминологии Flash называется сохранение готовой графики в одном из форматов, пригодных для ее распространения. В случае анимированной графики таких форматов три: распространяемая графика Shockwave/Flash, QuickTime и анимированный GIF. При публикации Flash выполняет очень сильную оптимизацию фильма: удаляет часть графики, которая никогда не появится на экране, перекодирует в свой внутренний формат импортированную графику, видео и звуки, удаляет ненужные цвета и "лишнее" пустое пространство. "На выходе", таким образом, получается очень компактный файл, который можно распространять.

Flash при публикации выполняет еще одну задачу. Он создает Web-страницу, содержащую весь HTML-код, необходимый для проигрывания вашего фильма в Web-обозревателе. Эта страница содержит теги HTML `<OBJECT>` и `<EMBED>`, которые обеспечивают вывод фильма с заданными вами при публикации параметрами. В результате вам остается всего лишь поместить готовый фильм и эту Web-страницу на Web-сервер.

При публикации вашего фильма вы можете также создать файлы форматов GIF, JPEG или PNG, содержащие статичное представление первого кадра фильма. Это бывает полезно для того, чтобы на компьютерах, не имеющих установленного проигрывателя Flash, показать хотя бы что-то. Flash сам добавит в Web-страницу необходимый код, проверяющий, установлен ли на компьютере проигрыватель Flash, и выводящий подходящее изображение.

Экспорт же — это просто создание файла фильма в одном из вышеперечисленных форматов. Воспользуйтесь этим, если вы хотите использовать ваш фильм в других программах или создать Web-страницу для его вывода самостоятельно.

При создании фильма вы можете указать, что какие-то (или все) его кадры могут быть распечатаны на принтере. Как это сделать, мы рассмотрим в конце этой главы.

Публикация фильма

Фильм готов. Осталось донести его до широких масс поклонников. Но перед этим его нужно опубликовать: сохранить в одном из распространяемых форматов, подготовить все сопутствующие файлы и каким-то способом сделать его доступным. Сейчас мы опишем, как все это делается, за исключением самого процесса распространения.

Выбор формата публикации

Выбрать формат публикации вашего фильма проще, чем формат статичного изображения. Повторим, что существует всего три подходящих формата: распространяемая графика Shockwave/Flash, QuickTime и анимированный GIF. К тому же на выбор формата накладываются весьма жесткие ограничения, о которых сказано ниже.

Чаще всего фильмы, созданные во Flash, публикуют в формате распространяемой графики Shockwave/Flash. Это вполне логично: только Flash может адекватно воспроизвести созданную им же графику. К тому же, файлы Shockwave/Flash, особенно последней версии (MX), очень компактны, а значит, быстрее передаются по Сети и занимают меньше места на дисках. Ну, а уж если вы создали в своем фильме какие-либо ActionScript-сценарии, то вам сама судьба велела опубликовать фильм в формате Shockwave/Flash. Ни QuickTime, ни анимированный GIF не поддерживают сценарии.

Надо еще добавить, что файл Shockwave/Flash может быть опубликован в трех различных формах. Во-первых, это собственно файл с расширением swf. Во-вторых, это исполняемый файл exe, предназначенный для операционных систем Microsoft Windows. В-третьих, это такой же исполняемый файл, но для операционных систем Apple Macintosh OS (расширение hqx). То есть, вы можете сделать ваш фильм доступным даже тем, у кого на компьютере нет установленного проигрывателя Flash. Однако пользуйтесь такой возможностью осторожно и с оглядкой: дело в том, что при переводе в исполняемый файл размер фильма Shockwave/Flash увеличивается на 800 килобайт. Разумеется, для распространения через Интернет такой файл однозначно не подходит.

Если же вы привязали к вашему фильму файл QuickTime, то сможете опубликовать фильм только в формате QuickTime — и никаком другом. В противном случае проигрыватель Flash не сможет воспроизвести ваш фильм.

Здесь надо рассказать о том, как анимация Flash преобразуется в фильм QuickTime. Дело в том, что фильм QuickTime может состоять из нескольких

дорожек, каждая из которых представляет собой как бы отдельный небольшой фильм в фильме. Все эти дорожки проигрываются одновременно. Так вот, вся анимация Flash помещается на одну дорожку целиком, становясь единым целым. Но каждому импортированному клипу QuickTime (если таковые есть) выделяется своя отдельная дорожка. Это может привести к интересным эффектам, которые мы опишем далее в этой главе.

Таким образом, если вы хотите распространить свое творчество среди пользователей платформы Apple Macintosh, то лучше опубликовать его в QuickTime. Хотя, это тоже по вашему желанию.

Осталось сказать об анимированном GIF. Здесь правило самое простое: в этом формате публикуйте только совсем простые и короткие фильмы. Например, рекламные баннеры и элементы оформления Web-страниц.

Выбрав формат публикации вашего фильма, не забудьте о сопутствующих файлах. А именно, о Web-странице и статических изображениях, представляющих первый кадр вашего фильма. Конечно, если вы публикуете файл не для помещения в Интернет, а для распространения на других носителях, скажем, на компакт-дисках или по электронной почте, сопутствующие файлы вряд ли окажутся вам нужны.

Публикация фильма

Фильм публикуется точно так же, как статичное изображение, все это было подробно описано в *главе 11*. Выберите пункт **Publish Settings** в меню **File** или нажмите комбинацию клавиш <Ctrl>+<Shift>+<F12>. После этого на экране появится диалоговое окно **Publish Settings**, показанное на рис. 11.1. Переключитесь на вкладку **Formats** и задайте с помощью набора флажков **Type** форматы публикуемых файлов. Задайте имена файлов с помощью группы полей ввода **Filename**, если нужно. Сама публикация осуществляется после нажатия кнопки **Publish**.

Здесь нужно дать небольшое пояснение. Флажок **Windows Projector** позволяет вам создать файл Shockwave/Flash, "запакованный" в исполняемый EXE-файл для Windows. Флажок **Macintosh Projector** создает такой же исполняемый файл, но для системы Macintosh OS.

Если вы хотите сохранить заданные вами параметры публикации, нажмите кнопку **OK**. В противном случае нажмите кнопку **Cancel**.

Вы можете настроить параметры публикации для различных форматов файлов. Для этого вам следует переключиться на нужную вкладку окна **Publish Settings** и задать требуемые параметры с помощью расположенных на этой вкладке элементов управления. Ниже мы опишем все эти параметры.

Shockwave/Flash

Если на вкладке **Formats** диалогового окна **Publish Settings** включены флажки **Flash**, **Windows Projector** или **Macintosh Projector**, становится доступной

вкладка **Flash**. Ее содержимое показано на рис. 19.1. Давайте рассмотрим имеющиеся там многочисленные элементы управления.

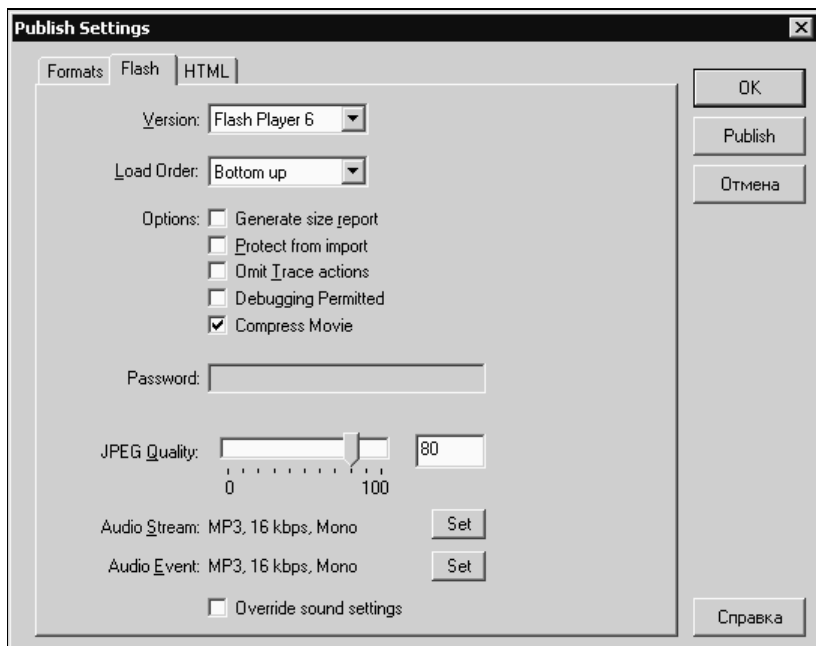


Рис. 19.1. Диалоговое окно **Publish Settings** (вкладка **Flash**)

С помощью раскрывающегося списка **Version** вы можете выбрать версию создаваемого файла Shockwave/Flash. Этот список содержит шесть пунктов: **Flash Player 1**, **Flash Player 2**, **Flash Player 3**, **Flash Player 4**, **Flash Player 5** и **Flash Player 6** (то есть, MX; этот пункт выбран по умолчанию). Назначение данных пунктов понятно без дополнительных разъяснений. Учтите только, что, если вы создаете фильм в более новой версии Flash, а сохраняете в более старой, то некоторые особенности могут быть потеряны из-за того, что более старая версия их не поддерживает.

Раскрывающийся список **Load Order** задает порядок, в котором будут загружаться слои вашего изображения. Благодаря этому, вы можете выбрать, какие части изображения будут грузиться в первую очередь, какие — во вторую и т. д. Список содержит два пункта: **Bottom up** (слои грузятся в порядке снизу вверх; этот пункт выбран по умолчанию) и **Top down** (слои грузятся в порядке сверху вниз).

Флажок **Generate size report** включает или отключает создание при публикации текстового отчета, содержащего данные Профилировщика загрузки. Такой отчет представляет собой текстовый файл с именем вида "<Имя файла Shockwave/Flash> Report.txt" и создается в той же папке, где будет находить-

ся сформированный файл Shockwave/Flash. Имеющиеся в нем данные могут помочь вам в оптимизации фильма.

Флажок **Protect from import** позволяет вам запретить кому-либо импортировать ваш фильм в документы Flash без вашего разрешения. Если включить этот флажок, становится доступным поле ввода **Password**, в которое вы сможете ввести пароль, разрешающий импорт фильма. После этого при попытке импортировать ваш фильм Flash будет спрашивать пароль, разумеется, импортировать его сможет только тот, кто этот пароль знает.

Флажок **Omit Trace actions** включает или отключает использование в фильме действия trace. Если вы включите этот флажок, никто не сможет трассировать сценарии, помещенные в этом фильме, чтобы понять, как они работают.

Флажок **Debugging Permitted** включает или выключает удаленную отладку. Если включить этот флажок, становится доступным поле ввода **Password**, в которое вы сможете ввести пароль, разрешающий удаленную отладку фильма.

Флажок **Compress Movie** включает или отключает сжатие полученного файла Shockwave/Flash. По умолчанию он включен, и выключать его не рекомендуется, т. к. сжатое изображение получается намного меньше по размерам, чем несжатое. Особенно эффективно сжимаются изображения, содержащие много текста или сценариев. Имейте только в виду, что сжиматься могут только файлы версии 6 (MX).

Регулятор **JPEG Quality** позволяет задать качество всех импортированных растровых изображений, для которых не были заданы индивидуальные параметры экспорта (об этом см. главу 8). Доступны для ввода значения в диапазоне от 0 до 100. Чем выше качество, тем лучше выглядит изображение и тем больше по размеру получается файл Shockwave/Flash. В поле ввода, расположенном правее регулятора, можно указать значение качества вручную.

При нажатии на кнопку **Audio Stream: Set** на экране появится диалоговое окно **Sound Settings** (см. рис. 17.5). С его помощью вы можете задать параметры экспорта потоковых звуков, для которых это не было сделано индивидуально (об этом см. главу 17). Аналогично, нажав кнопку **Audio Event: Set**, вы можете в таком же окне задать параметры экспорта сигналов.

Если вы хотите, чтобы заданные вами параметры экспорта звуков перекрыли параметры, действующие для отдельных звуков индивидуально, включите флажок **Override sound settings**. Это может понадобиться, например, если вы хотите создать специальную "компактную" версию фильма, содержащую звуки низкого качества.

HTML

Если на вкладке **Formats** диалогового окна **Publish Settings** включен флажок **HTML** (то есть, включено создание Web-страницы), становится доступной вкладка **HTML**. Ее содержимое показано на рис. 19.2. Рассмотрим его также многочисленные элементы управления.

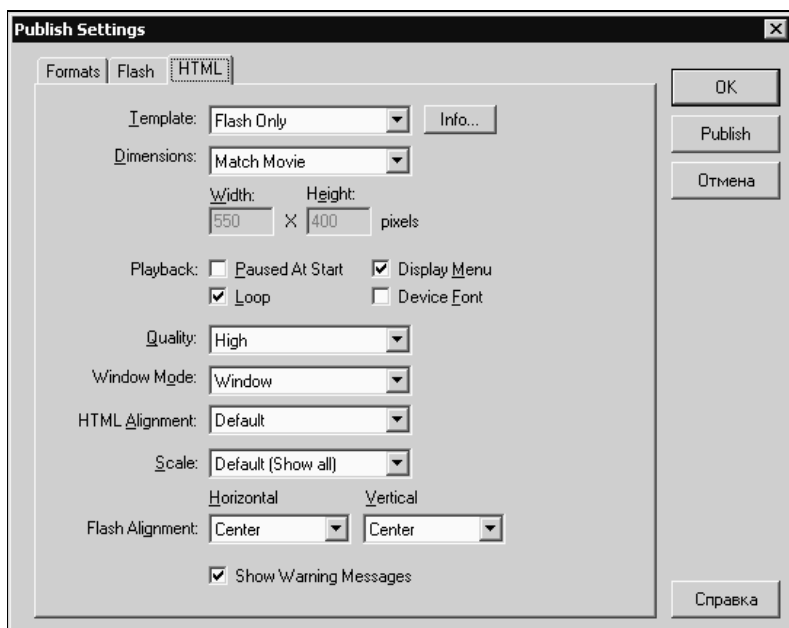


Рис. 19.2. Диалоговое окно **Publish Settings** (вкладка **HTML**)

С помощью раскрывающегося списка **Template** выбирается шаблон, на основе которого будет сгенерирована Web-страницы. Таких шаблонов довольно много:

- ☐ **Detect for Flash 3** — включает код, проверяющий наличие проигрывателя Flash 3;
- ☐ **Detect for Flash 4** — включает код, проверяющий наличие проигрывателя Flash 4;
- ☐ **Detect for Flash 5** — включает код, проверяющий наличие проигрывателя Flash 5;
- ☐ **Detect for Flash 6** — включает код, проверяющий наличие проигрывателя Flash 6;
- ☐ **Flash Only** — просто показывает фильм Flash;
- ☐ **Flash Only for Pocket PC** — то же, что **Flash Only**, но рассчитан на карманные компьютеры;
- ☐ **Flash w/AICC Tracking** и **Flash w/SCORM Tracking** — рекомендуются, если в фильме используются компоненты;
- ☐ **Flash with FSCommand** — включает код, необходимый для поддержки действия FSCommand;
- ☐ **Flash with Named Anchors** — включает код, необходимый для поддержки именованных "якорей";

- ❑ **Image Map** — создает карту-изображение HTML. Также требует, чтобы был создан файл GIF, JPEG или PNG;
- ❑ **QuickTime** — показывает фильм QuickTime.

Если вы хотите получить сведения по тому или иному шаблону, нажмите кнопку **Info**. После этого на экране появится диалоговое окно **HTML Template Info** (рис. 19.3), содержащее краткие сведения о выбранном в списке **Template** шаблоне. Прочитайте их и закройте окно, нажав кнопку **OK** или **Cancel**.

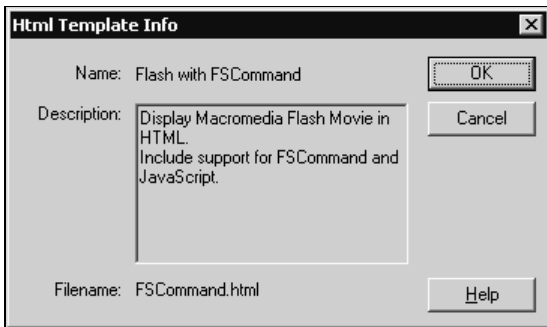


Рис. 19.3. Диалоговое окно **HTML Template Info**

Шаблоны, на основе которых создаются Web-страницы, — это обычные файлы HTML. Добраться до них не очень сложно. Если вы работаете в Windows 95/98/Me, то откройте подкаталог Application Data/Macromedia/Flash MX/Configuration/HTML, расположенный в каталоге Windows. Если же вы работаете в Windows NT/2000/XP, то откройте этот же подкаталог Application Data/Macromedia/Flash MX/Configuration/HTML, расположенный в каталоге вашего пользовательского профиля. Вы можете удалить ненужные шаблоны или, наоборот, добавить свои собственные. Можно также изменить существующие шаблоны.

Раскрывающийся список **Dimensions** и поля ввода **Width** и **Height** позволят вам задать геометрические размеры вашего фильма. Если в списке **Dimensions** выбран пункт **Match Movie**, то они будут равны размерам, заданным при создании фильма. Если выбран пункт **Pixels**, то вы можете сами задать размеры фильма с помощью полей ввода **Width** (ширина) и **Height** (длина) в пикселах. Если же выбран пункт **Percent**, то размеры фильма задаются в процентах относительно размеров окна Web-обозревателя.

Если включен флажок **Paused At Start**, фильм не будет запущен на воспроизведение сразу же после загрузки. Вместо этого пользователю самому придется запускать этот фильм, воспользовавшись контекстным меню или иным способом. Если флажок **Paused At Start** отключен (а он отключен по умолчанию), фильм будет воспроизведен сразу же после загрузки.

Флажок **Loop** включает или отключает "зацикливание" фильма.

Флажок **Display Menu** разрешает или запрещает вывод контекстного меню проигрывателя Flash при щелчке на нем правой кнопкой мыши. С помощью такого меню пользователь может запускать и останавливать фильм, задавать качество его воспроизведения и некоторые другие параметры. Если этот пункт отключен, то в контекстном меню остается один-единственный пункт — **About Flash**, выводящий сведения о программе проигрывателя. Запретить контекстное меню бывает нужно в тех случаях, когда оно может помешать, скажем, при создании Flash-приложений.

Флажок **Device Font** разрешает или запрещает Flash подставлять вместо всех использованных в фильме шрифтов шрифты-псевдонимы. Такая подстановка работает только для обычных текстовых блоков.

Раскрывающийся список **Quality** позволяет задать качество вывода изображения. Это может понадобиться на медленных компьютерах, которые не справляются с выводом графики высокого качества. Список содержит шесть пунктов:

- ☐ **Low** — скорость воспроизведения имеет приоритет перед качеством графики, сглаживание не используется. Самое низкое качество;
- ☐ **Auto Low** — изначально качество графики низкое, сглаживание не используется. Потом, если компьютер окажется достаточно мощным, Flash повысит качество графики и включит сглаживание;
- ☐ **Auto High** — изначально качество графики высокое, используется сглаживание. Потом, если компьютер окажется недостаточно мощным, Flash понизит качество графики и отключит сглаживание;
- ☐ **Medium** — используется сглаживание для векторной, но не для растровой графики. Среднее качество;
- ☐ **High** — качество графики имеет приоритет над скоростью воспроизведения, используется сглаживание векторной графики, а если нет анимации, то и растровой графики. Этот пункт выбран по умолчанию;
- ☐ **Best** — используется сглаживание и векторной, и растровой графики. Самое высокое качество, которое не уменьшается ни в каких случаях.

Раскрывающийся список **Window Mode** позволяет задать вид фильма на Web-странице. Этот список содержит три пункта:

- ☐ **Window** — фильм отображается на Web-странице в собственном "окошке". Самая высокая скорость воспроизведения. Этот пункт выбран по умолчанию;
- ☐ **Opaque Windowless** — фильм не будет отображаться в собственном "окошке". Элементы, находящиеся на Web-странице "ниже" этого фильма, не будут видны;

- ❑ **Transparent Windowless** — фильм не будет отображаться в собственном "окошке". Элементы, находящиеся на Web-странице "ниже" этого фильма, будут видны. Учтите, что в этом случае может ухудшиться качество проигрывания фильма.

Раскрывающийся список **HTML Alignment** задает местоположение фильма на Web-странице. Здесь имеется пять пунктов:

- ❑ **Default** — фильм отображается в центре страницы. Если окно Web-обозревателя меньше, чем нужно, границы фильма будут обрезаны;
- ❑ **Left, Top, Bottom** и **Right** — фильм выравнивается соответственно по левой, верхней, нижней или правой границе Web-страницы. Если окно Web-обозревателя меньше, чем нужно, три другие границы фильма будут обрезаны.

С помощью раскрывающегося списка **Scale** вы можете задать параметры масштабирования изображения Flash, если оно не помещается в отведенные ему размеры. Этот список содержит четыре пункта:

- ❑ **Default (Show all)** — будет показано все изображение, для чего может быть применено масштабирование. Однако пропорции изображения искажены не будут, в результате этого вдоль горизонтальных или вертикальных сторон его могут появиться границы;
- ❑ **No Border** — то же самое, что **Default (Show all)**, но границы появляться не будут — Flash обрежет изображение по горизонтали или вертикали, чтобы их избежать;
- ❑ **Exact Fit** — будет показано все изображение, для чего может быть применено масштабирование, в результате которого могут исказиться размеры изображения;
- ❑ **No Scale** — изображение ни в каком случае не будет масштабироваться, в результате чего может оказаться обрезанным.

Набор из двух раскрывающихся списков **Flash Alignment** позволяет задать выравнивание изображения внутри "окошка" проигрывателя Flash. В списке **Horizontal** задается выравнивание по горизонтали: **Left** (по левому краю), **Center** (по центру; значение по умолчанию) или **Right** (по правому краю). В списке **Vertical** задается выравнивание по вертикали: **Top** (по верхнему краю), **Center** (по центру; значение по умолчанию) или **Bottom** (по нижнему краю).

При задании параметров публикации фильма Flash немудрено допустить ошибку. Например, возможна такая ситуация: если для данного шаблона требуется статичное представление первого кадра фильма, чтобы заменить его в случае невозможности запуска проигрывателя Flash, а вы не включили его создание. Поэтому включите флажок **Show Warning Messages** — и Flash предупредит вас о таких ошибках. (Впрочем, этот флажок включен по умолчанию.)

QuickTime

Если на вкладке **Formats** диалогового окна **Publish Settings** включен флажок **QuickTime**, становится доступной вкладка **Flash**. Ее содержимое показано на рис. 19.4. Давайте рассмотрим находящиеся там элементы управления.

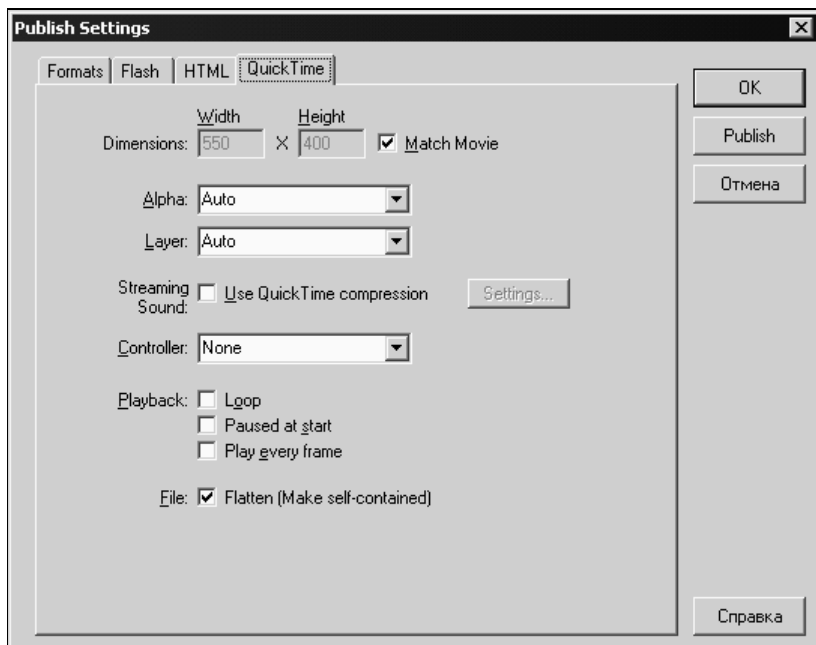


Рис. 19.4. Диалоговое окно **Publish Settings** (вкладка **QuickTime**)

Группа **Dimensions** из полей ввода **Width** и **Height** служит для задания соответственно ширины и высоты результирующего изображения. Если включен флажок **Match Movie**, эти размеры совпадают с размерами рабочего листа.

С помощью раскрывающегося списка **Alpha** задается прозрачность видеодорожки, содержащей анимацию Flash. (Точнее, прозрачность не самой дорожки, а пустого пространства — сама графика останется непрозрачной, иначе пользователь ничего не увидит.) Этот список содержит три пункта:

- ☐ **Alpha-transparent** — дорожка с анимацией Flash делается прозрачной, и сквозь нее будет видно содержимое остальных дорожек;
- ☐ **Copy** — дорожка с анимацией Flash делается непрозрачной, и содержимое остальных дорожек сквозь нее видно не будет;
- ☐ **Auto** — если слои с графикой Flash находятся выше слоев с импортированными клипами QuickTime, то они делаются прозрачными, если ниже — непрозрачными. Этот пункт выбран по умолчанию.

Раскрывающийся список **Layer** задает размещение дорожки, содержащей анимацию Flash, относительно других дорожек клипа. Этот список содержит три пункта:

- ☐ **Top** — дорожка с анимацией Flash будет всегда помещаться наверху, выше всех остальных дорожек;
- ☐ **Bottom** — дорожка с анимацией Flash будет всегда помещаться внизу, ниже всех остальных дорожек;
- ☐ **Auto** — если слои с графикой Flash находятся выше слоев с импортированными клипами QuickTime, то и дорожка с анимацией Flash будет помещена наверху, в противном случае — внизу. Этот пункт выбран по умолчанию.

Флажок **Use QuickTime compression** включает или отключает сохранение звукового сопровождения фильма в формате QuickTime. Вероятно, его стоит всегда держать включенным, чтобы у пользователей не возникло проблем с воспроизведением звука.

Если этот флажок включен, становится доступной кнопка **Settings**. При ее нажатии на экране появляется диалоговое окно задания параметров сохранения звука. Задайте нужные параметры и нажмите кнопку **OK** для их сохранения или кнопку **Cancel** для отмены всех изменений.

Раскрывающийся список **Controller** позволяет выбрать, какой набор элементов управления проигрыванием будет использован при воспроизведении видеоклипа. Этот список содержит три пункта:

- ☐ **None** — элементы управления проигрыванием вообще не будут присутствовать (значение по умолчанию);
- ☐ **Standard** — стандартный набор элементов управления;
- ☐ **QuickTime VR** — набор элементов управления, реализующий функции "виртуальной реальности".

Флажок **Loop** включает или отключает "зацикливание" фильма.

Если включен флажок **Paused at start**, фильм не будет запущен на воспроизведение сразу же после загрузки. Вместо этого пользователю самому придется запускать этот фильм. Если флажок **Paused at start** отключен (а он отключен по умолчанию), фильм будет воспроизведен сразу же после загрузки.

Если включен флажок **Play every frame**, проигрыватель QuickTime будет проигрывать каждый кадр фильма, не пропуская их. Звуковое сопровождение в этом случае проигрывается не будет.

Если включен флажок **Flatten (Make self-contained)**, Flash включит все импортированные файлы в один результирующий файл QuickTime. Если он отключен, то в результирующий файл будут помещены только ссылки на соответствующие внешние файлы.

GIF

Если на вкладке **Formats** диалогового окна **Publish Settings** включен флажок **GIF**, то соответствующая вкладка становится доступной. Содержимое вкладки **GIF** было показано на рис. 11.2. Мы рассмотрим только элементы управления, задающие настройки анимации. Остальные элементы управления были подробно описаны в *главе 11*.

Группа переключателей **Playback** позволит вам выбрать между созданием статичного и анимированного изображения GIF. Если выбран переключатель **Static**, будет создано статичное изображение, если выбран переключатель **Animated** — анимированное.

Если в группе **Playback** включен переключатель **Animated**, становится доступной еще одна группа переключателей, задающая параметры "зацикливания" фильма. При включении переключателя **Loop Continuously**, фильм будет проигрываться непрерывно, пока не будет остановлен. Если же включен переключатель **Repeat**, вы сможете задать количество его повторений в расположенном правее поле ввода.

Экспорт фильма

Если вы хотите просто сохранить ваш документ Flash в одном из распространенных форматов, прибегните к экспорту. В то время как при публикации создается сразу несколько файлов, представляющих собой готовое решение для публикации в Интернете, при экспорте одновременно создается только один файл — тот самый, который вы заказывали. Например, если вы захотите сохранить документ Flash в формате QuickTime, вы получите файл QuickTime — и больше ничего.

Форматы экспорта, поддерживаемые Flash

Здесь мы перечислим все форматы, в которые Flash может экспортировать анимацию. Результаты приведены в табл. 19.1.

Таблица 19.1. Форматы экспорта анимации, поддерживаемые Flash

Название формата	Расширение файлов
Adobe Illustrator	ai
Apple QuickTime	mov
AutoDesk AutoCAD	dxf
AVI	avi
Encapsulated PostScript	eps

Таблица 19.1 (окончание)

Название формата	Расширение файлов
FutureSplash	spl
GIF (обычный и анимированный)	gif
JPEG	jpg, jpe, jpeg
Macromedia Shockwave/Flash	swf
PNG	png
WAV (только звук)	wav
Метафайлы Windows	wmf
Растровые файлы Windows	bmp
Расширенные метафайлы Windows	emf

Как видите, в табл. 19.1 присутствуют не только форматы анимации, но и форматы статичной графики и даже один звуковой формат. Со звуковым форматом все более-менее понятно: в него записывается все звуковое сопровождение фильма. Иногда это полезно, если вы хотите создать звуковую дорожку ("саундтрек", soundtrack) своего фильма. Но как быть с форматами статичной графики? Каким образом фильм может быть экспортирован в такой формат?

Flash имеет одну очень интересную особенность. Он может экспортировать каждый кадр фильма в отдельный графический файл. При этом кадры могут сохраняться в файлах любого формата: статичный GIF, JPEG, PNG, BMP, Adobe Illustrator и др. Эти файлы получают имена вида "<Имя, заданное вами при экспорте><Номер кадра>.<Расширение, соответствующее формату экспорта>". Такие наборы кадров могут пригодиться, например, для создания слайд-шоу.

Экспорт анимации

Для того чтобы экспортировать фильм Flash, выберите пункт **Export Movie** в меню **File** или нажмите комбинацию клавиш <Ctrl>+<Alt>+<Shift>+<S>. На экране появится стандартное диалоговое окно сохранения файла Windows. Выберите нужный формат файла в раскрывающемся списке в нижней части этого окна, задайте имя файла и нажмите кнопку сохранения.

В некоторых случаях Flash просто сохранит созданный в результате экспорта файл на диске. Но чаще на экране появится новое диалоговое окно, где вам будет предложено задать некоторые параметры сохраняемого файла. Задав их, нажмите кнопку **ОК** — и файл будет сохранен. Если вы передумали экспортировать графику, нажмите кнопку **Cancel**.

Ниже будут описаны все эти диалоговое окна и особенности экспорта в различные видеоформаты. Экспорт в форматы статичной графики (в смысле, когда фильм сохраняется как последовательность отдельных кадров) был подробно описан в *главе 11*, поэтому повторяться мы не будем. В случае если вы выбрали в качестве формата экспорта формат статичной графики (все они имеют в названии слово "Sequence"), обратитесь за описанием его параметров к *главе 11*.

Shockwave/Flash и FutureSplash

После нажатия кнопки сохранения в диалоговом окне сохранения файла Windows на экране появится другое диалоговое окно, показанное на рис. 19.5. В нем вы можете задать параметры создаваемого файла Shockwave/Flash или FutureSplash — для обоих форматов эти параметры одинаковы.



Рис. 19.5. Диалоговое окно **Export Flash Player**

Как видите, все элементы управления совершенно аналогичны элементам управления вкладкой **Flash** диалогового окна **Publish Settings** (см. рис. 19.1). Поэтому мы не будем их описывать еще раз.

AVI

После нажатия кнопки сохранения в диалоговом окне сохранения файла Windows на экране появится другое диалоговое окно, показанное на рис. 19.6. В нем вы можете задать параметры создаваемого файла AVI.

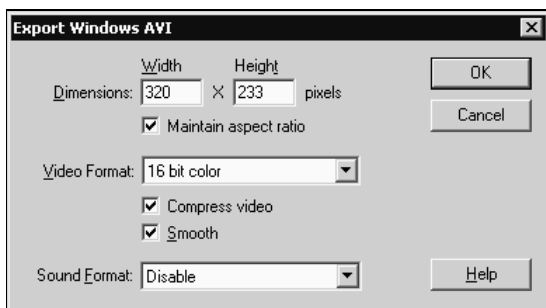


Рис. 19.6. Диалоговое окно **Export Windows AVI**

Группа **Dimensions** из полей ввода **Width** и **Height** служит для задания соответственно ширины и высоты результирующего изображения. Если вы включите флажок **Maintain aspect ratio**, то при вводе данных в одно поле ввода Flash сам введет в другое поле ввода такое значение, чтобы пропорции изображения сохранились.

Раскрывающийся список **Video Formal** задает цветовой режим изображения. Доступно четыре пункта:

- ☐ **8 bit color** — 256-цветное изображение;
- ☐ **16 bit color** — изображение с 16-битным цветом (HiColor);
- ☐ **24 bit color** — изображение с 24-битным цветом (TrueColor);
- ☐ **32 bit color w/ alpha** — изображение с 24-битным цветом и каналом прозрачности (альфа-канал).

Если включен флажок **Compress video**, то после нажатия кнопки **OK** Flash выведет на экран стандартное диалоговое окно Windows задания параметров сжатия видео. Это окно показано на рис. 19.7. С помощью раскрывающегося списка **Программа сжатия** выбирается кодек для сжатия видео, а с помощью расположенных ниже элементов управления задаются дополнительные параметры сжатия. После этого нажмите кнопку **OK** для сохранения файла AVI или кнопку **Отмена** для отказа от этого.

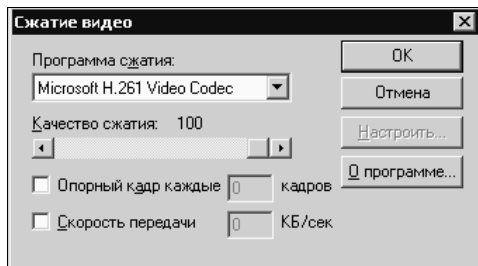


Рис. 19.7. Стандартное диалоговое окно Windows задания параметров сжатия видео

Если флажок **Compress video** отключен, видео вообще не будет сжиматься. Имейте при этом в виду, что результирующий файл AVI может оказаться очень большим. Поэтому без сжатия следует сохранять только наиболее короткие и очень маленькие фильмы.

Флажок **Smooth** включает или отключает сглаживание контуров результирующего изображения.

Раскрывающийся список **Sound Format** позволяет задать наличие и качество звукового сопровождения фильма. Этот список содержит очень много пунктов, задающих параметры качества звука. Названия этих пунктов имеют вид "<Частота оцифровки><Количество разрядов оцифровки><Моно или стерео>". Например, пункт **22kHz 16 Bit Mono** задает шестнадцатиразрядный монофонический звук с частотой оцифровки 22 кГц. А пункт **Disable** позволяет вообще убрать звук из фильма.

QuickTime

После нажатия кнопки сохранения в диалоговом окне сохранения файла Windows на экране появится другое диалоговое окно, показанное на рис. 19.8. В нем вы можете задать параметры создаваемого файла QuickTime.

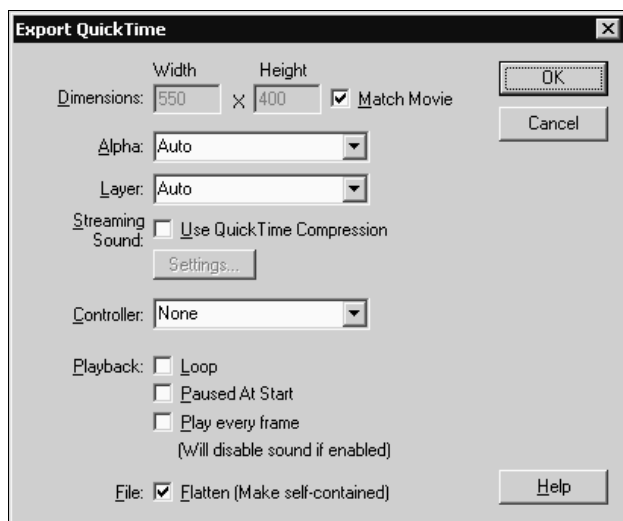


Рис. 19.8. Диалоговое окно **Export QuickTime**

Все элементы управления этого окна совершенно аналогичны элементам управления вкладки **QuickTime** диалогового окна **Publish Settings** (см. рис. 19.4). Поэтому мы не будем их описывать еще раз.

GIF (анимированный)

После нажатия кнопки сохранения в диалоговом окне сохранения файла Windows на экране появится другое диалоговое окно, показанное на рис. 19.9. В нем вы можете задать параметры создаваемого файла GIF.

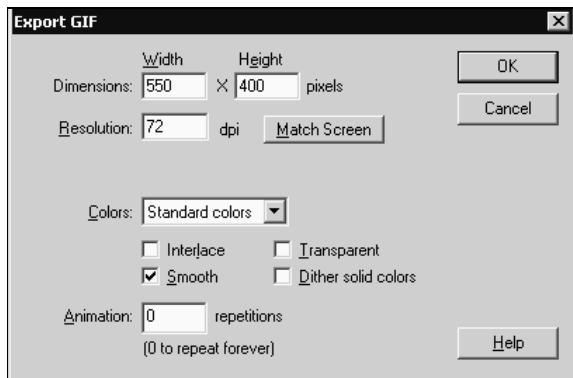


Рис. 19.9. Диалоговое окно **Export GIF**

Группа **Dimensions** из полей ввода **Width** и **Height** служит для задания соответственно ширины и высоты результирующего изображения. В поле ввода **Resolution** задается разрешающая способность изображения в точках (пикселах) на дюйм. Если вы нажмете кнопку **Match Screen**, Flash установит эти параметры сам, основываясь на размерах рабочего листа и разрешении экрана.

Раскрывающийся список **Colors** задает количество доступных цветов в цветовой палитре изображения. Доступно восемь пунктов:

- ☐ **Black & White** — черно-белое изображение, два цвета;
- ☐ **4, 8, 16, 32, 64, 128, 256 colors** — соответствующее количество цветов;
- ☐ **Standard colors** — безопасная палитра Web.

Группа флажков, расположенных ниже этого раскрывающегося списка, вам уже большей частью знакома. Флажок **Interlace**, будучи включенным, вызывает создание изображения GIF с чередованием. Флажок **Smooth** включает или отключает сглаживание контуров результирующего изображения. Флажок **Transparent** при включении делает цвет фона рабочего листа прозрачным. А флажок **Dither solid colors** включает или отключает использование составных цветов.

В поле ввода **Animation** вводится количество повторений фильма. Если вы хотите, чтобы он повторялся бесконечно ("зациклился"), введите туда значение, равное нулю.

По умолчанию в формат GIF экспортируется весь фильм. Однако вы можете сохранить в этом формате только определенный фрагмент фильма. Для этого задайте для начального и конечного кадров нужного фрагмента имена #First и #Last.

WAV (только звук)

После нажатия кнопки сохранения в диалоговом окне сохранения файла Windows на экране появится другое диалоговое окно, показанное на рис. 19.10. В нем вы можете задать параметры создаваемого файла WAV.

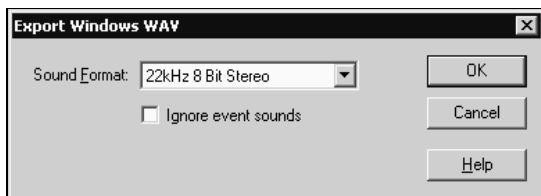


Рис. 19.10. Диалоговое окно **Export Windows WAV**

Раскрывающийся список **Sound Format** позволяет задать наличие и качество звука. Этот список содержит очень много пунктов, задающих параметры качества. Названия этих пунктов имеют вид "<Частота оцифровки><Количество разрядов оцифровки><Моно или стерео>". Например, пункт **22kHz 16 Bit Mono** задает шестнадцатиразрядный монофонический звук с частотой оцифровки 22 кГц.

Очень странно, но в этом списке также предусмотрен пункт **Disable**. Если его выбрать, Flash создаст "пустой" звуковой файл. Зачем это было сделано, непонятно, ведь если звук не нужен, его можно и не экспортировать в отдельный файл WAV.

Если включить флажок **Ignore event sounds**, Flash не включит в звуковое сопровождение фильма все сигналы. Такое звуковое сопровождение будет содержать только потоковые звуки.

Создание фильмов, предназначенных для печати

Говорят, что компьютер — один из "убийц" деревьев. В смысле, компьютеры, а точнее, принтеры потребляют столько бумаги, что экологи всего мира хватаются за голову. Говорят, что концепция "безбумажного делопроизводства", когда все необходимые документы существуют только в электронном виде, осталась, извините за каламбур, только на бумаге. Но что делать, если за несколько тысячелетий человечество так привыкло к рукописным и печатным документам, что до сих пор не может от них отказаться. Вот и работают без устали в конторах принтеры, и порождают горы нужной, а чаще ненужной макулатуры, которая выбрасывается зачастую сразу же после печати.

Конечно, экономить бумагу важно и нужно — этим вы спасаете леса, единственный источник кислорода на нашей многострадальной планете. Но часто бывает так, что без твердой копии не обойтись. Вот как раз для таких случаев Flash и позволяет вам распечатать весь фильм или его избранные кадры. И ни для каких других!

Для пользователя проще всего распечатать фильм из проигрывателя Flash, представляющего собой отдельное приложение. (Сюда же относятся фильмы Shockwave/Flash, оформленные в виде исполняемых файлов Windows или Macintosh.) В этом случае пользователь просто выбирает пункт **Print** в контекстном меню проигрывателя, задает в появившемся на экране стандартном диалоговом окне печати Windows необходимые параметры и нажимает кнопку **ОК**. Остальное, как говорится, дело техники: принтер сразу же напечатает все кадры фильма.

Изначально все кадры фильма могут быть распечатаны. Однако вы можете разрешить печать только избранных кадров фильма, остальные кадры автоматически станут непечатаемыми. Например, если вы создаете каталог товаров в формате Flash, вы можете предусмотреть печать только одного-единственного кадра — формы заказа. Ниже мы рассмотрим, как это делается.

Чтобы сделать какой-либо кадр печатаемым, выделите его на временной шкале и обратите внимание на редактор свойств. В его верхнем левом углу вы увидите поле ввода, в котором задается имя кадра (см. рис. 14.1). Введите в него значение #p. Вот и все.

Повторите эту операцию для всех кадров, которые вы хотите сделать печатаемыми. Остальные кадры автоматически станут непечатаемыми.

По умолчанию границы печатаемой области кадра совпадают с границами рабочего листа фильма. Все, что выходит за эти границы, не будет отпечатано (чем, кстати, тоже можно воспользоваться, дабы скрыть информацию, не предназначенную для копирования). Однако вы можете задать другую *область печати*, например, чтобы распечатать упоминавшуюся выше форму заказа на весь бумажный лист. Делается это следующим способом.

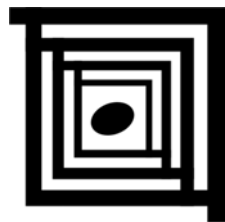
Выберите один из кадров, который вы не поместили для печати именем #p. Вы можете также вставить в последовательность новый кадр специально для этого. Нарисуйте прямоугольник, чьи размеры совпадут с желаемой областью печати. После этого выделите этот кадр на временной шкале и задайте для него имя #b. Будьте внимательны: кадр, задающий размеры области печати, должен быть только один на фильм.

Вы можете также запретить печать всех кадров фильма без исключения. Например, если вы хотите запретить пользователям делать любые копии своего фильма. Сделать это можно двумя различными путями.

Первый способ — воспользоваться все тем же редактором свойств. Выделите на временной шкале самый первый кадр вашего фильма и задайте для него имя !#p. Если пользователь при просмотре такого фильма откроет контекстное меню, пункт **Print** будет недоступен.

Второй способ — вообще убрать контекстное меню при публикации фильма. Для этого достаточно отключить флажок **Display Menu** на вкладке **HTML** диалогового окна **Publish Settings** (см. рис. 19.2).

Часть IV



Программирование

Глава 20. Основы программирования

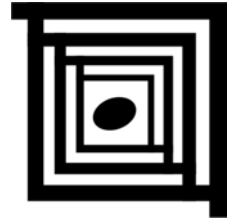
Глава 21. Язык ActionScript

Глава 22. Создание интерактивных фильмов

Глава 23. Создание приложений Flash

Глава 24. Работа с внешними приложениями

Глава 25. Средства отладки сценариев ActionScript



Глава 20

Основы программирования

Вы уже, вероятно, слышаны о программировании во Flash. Более того, вам уже, наверно, прожужжали этим программированием все уши. Только и слышно: программирование, программы, приложения, сценарии, ActionScript. Да что же это такое? И зачем оно нужно?

В начале этой книги говорилось, что Flash — замечательный графический пакет, предоставляющий сочетание векторной графики, анимации и интерактивности. Это означает, что вы можете нарисовать в нем какое-либо изображение, анимировать его и написать небольшие программы — *сценарии*, — выполняющиеся при наступлении некоторого *события*, например, щелчка мышью. Иными словами, если пользователь щелкнет мышью по какому-либо графическому элементу (например, кнопке), сценарий будет выполнен. А результатом работы такого сценария может быть переход на другую Web-страницу, загрузка другого фильма Flash или появление всплывающего меню.

Никакие другие графические программы таких возможностей не предоставляют. Поэтому Flash и приобрел такую популярность. Более того, он стал стандартом де-факто для интернет-графики, не будучи принятым и утвержденным комитетом W3C. А такое случается нечасто.

Во Flash сценарии пишутся на особом языке программирования, называемом *ActionScript*. Этот язык разработан на основе известного языка JavaScript, содержит общие с ним команды и следует тем же принципам написания программ. ActionScript описывает поведение графического элемента, то, что он должен делать в ответ на наступившее событие. Синтаксис этого языка, т. е. принципы написания команд и программ, очень прост, и, если вы владеете английским языком на уровне, необходимом для овладения англоязычными программами, то без труда в нем разберетесь. Тем более, что большинство сценариев обычных изображений и фильмов Flash состоят максимум из пяти команд.

Но что делать, если вы вообще не владеете никаким языком программирования? Ничего страшного! Разработчики Flash много поработали над тем,

чтобы сделать Flash-программирование доступным даже для начинающих. Для этого они создали исключительно мощный инструмент для написания программного кода — панель **Actions**. Пользуясь ей, вы сделаете несравнимо меньше ошибок, чем если бы вы писали код вручную. Но если уж вы настолько уверены в себе, что всяческие дополнительные "навороты" вам только мешают, то можете отключить их и пользоваться обычным текстовым редактором.

В этой главе мы рассмотрим использование панели **Actions** и сопутствующих ей инструментов. Собственно изучение языка ActionScript и всех его средств, а также принципов написания сценариев мы изучим в последующих главах *части 3*. А полное описание языка ActionScript приведено в *приложении 1*.

Но сначала давайте немного узнаем о том, как сценарии привязываются к элементам вашего фильма или изображения. И вообще, что можно делать, какие задачи можно решать с помощью сценариев.

Зачем нужны сценарии

Итак, зачем нужны сценарии?

Предположим, вам нужно "зациклить" фильм. Вы скажете, что это совсем просто — достаточно включить флажок **Loop** на вкладке **HTML** диалогового окна **Publish Settings** (см. рис. 19.2), и все! (Подробнее о публикации и экспорте фильмов см. *главу 19*.) Зачем в этом простом деле применять еще какие-то там сценарии?

Все это, конечно, и в самом деле очень просто. Но что, если вам нужно "зациклить" фильм так, чтобы повторялась только часть его, например, начиная с одиннадцатого кадра? Уже знакомыми вам средствами Flash так сделать нельзя. Зато можно написать простенький сценарий, состоящий из одного-единственного *действия*:

```
gotoAndPlay(11);
```

которое просто переместит указатель кадра на кадр № 11 и продолжит проигрывание фильма с него. Действие `gotoAndPlay` принимает один *параметр* (число в скобках), задающий номер кадра, с которого начнется воспроизведение фильма.

Также вы можете привязать к кнопке сценарий, выводящий на экран Web-страницу (подробнее о кнопках см. *главу 23*):

```
getURL("http://www.macromedia.com", "_blank");
```

Как видите, этот сценарий также состоит из одного действия — `getURL`, которое выводит на экран Web-страницу. Оно принимает два параметра в виде строк, заключенных в кавычки, эти строки задают соответственно гиперссылку и ее цель.

Итак, к каким же элементам фильма можно привязывать сценарии?

Во-первых, их можно привязывать к кадрам. При этом соответствующий сценарий выполнится, когда указатель достигнет этого кадра. Такие сценарии могут, например, "зацикливать" фильм или подгружать другие фильмы, содержащие продолжение текущего фильма, формы ввода данных или какие-либо приложения. Также очень часто к первому кадру фильма привязывается сценарий, выполняющий какие-либо подготовительные действия, — это лучшее место для такого рода операций.

Во-вторых, сценарии можно привязывать к кнопкам (подробнее о кнопках см. главу 23). Такие сценарии могут делать все, что угодно. Практика показывает, что большинство сценариев, реализованных во Flash-приложениях, привязаны как раз к кнопкам.

В-третьих, сценарии могут быть привязаны к экземплярам образцов-клипов. Такие сценарии тоже могут делать все, что угодно, благо вы сами можете задавать событие, в ответ на которое сценарий будет срабатывать.

Теперь следует сказать еще вот о чем. Если вы хотите управлять из сценария каким-либо элементом вашего фильма, вам нужно будет присвоить этому элементу уникальное имя. Исключение из этого правила — кадры. Кадры могут не иметь имени — в таком случае доступ к ним осуществляется по номеру (что, надо сказать, не всегда удобно). Чтобы задать это имя, вам нужно будет ввести его в поле ввода, расположенное в верхнем левом углу редактора свойств (см. рис. 14.1). Естественно, перед этим требуемый элемент (кадр, кнопка или экземпляр образца-клипа) должен быть выделен. Запомните еще, что имя может содержать только буквы латинского алфавита, цифры и знак подчеркивания, но начинаться обязано только с буквы.

Создание сценариев и работа с ними

Теперь, когда мы познакомились со сценариями и выяснили их пользу, пора приступить к их написанию. А именно, к святой святых Flash — программированию.

Знакомство с панелью *Actions*

Прежде чем описывать программирование в среде Flash, познакомимся с инструментом, дающим нам такую возможность. Это та самая панель **Actions**, о которой вы, конечно, уже слышаны. Мощнейший инструмент, который не дает совершать ошибки новичку, но и не мешает работать профессионалу.

Чтобы вывести панель **Actions** на экран, выполните одно из четырех действий:

- ☐ выберите пункт **Actions** в меню **Window**;
- ☐ нажмите клавишу <F9>;

- ❑ выберите пункт **Actions** в контекстном меню кадра, кнопки или образца-клипа;
- ❑ нажмите кнопку (рис. 20.1), расположенную в правой части редактора свойств.

Панель **Actions** показана на рис. 20.2. Как видите, она разделена на две части, которые мы сейчас рассмотрим.



Рис. 20.1. Кнопка вызова панели **Actions**

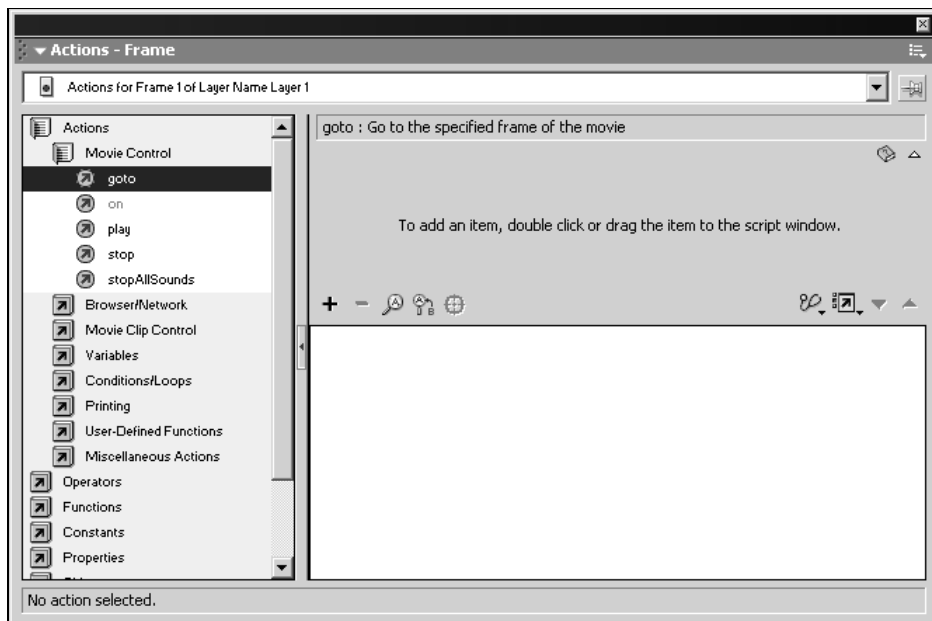


Рис. 20.2. Панель **Actions** (обычный режим)

В левой части панели **Actions** находится иерархический список доступных в языке **ActionScript** операций, которые вы можете использовать в своих сценариях. Таких операций очень много, так много, что они разбиты на несколько групп. Список в левой части панели представляет собой несколько соответствующих им "ветвей". Если в сценарии, привязанном к какому-то элементу фильма, нельзя использовать данную операцию, она показана в списке серым цветом.

В правой части панели находится текст вашего сценария. Он представляет собой список входящих в него операций языка **ActionScript**, вы можете вы-

бирать в этом списке нужную операцию и выполнять над ней какие-то действия. В частности, вы можете удалить ее, переместить вверх или вниз в списке или задать для нее какие-то параметры. Параметры задаются с помощью набора элементов управления, появляющихся выше этого списка (рис. 20.3). Номер и текст выбранной в списке операции показывается строке статуса панели **Actions**.



Рис. 20.3. Набор элементов управления, служащих для задания параметров выбранной в списке операции

Нет, конечно, мы не будем рассматривать все элементы управления для задания всех параметров всех операций, доступных в языке ActionScript! (Для этого есть *приложение 1*.) Мы просто рассмотрим их для случая с действием `gotoAndPlay`, опишем общие моменты использования этих элементов управления. Мы дадим вам только основные знания, которыми вы и будете пользоваться в дальнейшем.

Здесь надо сказать следующее. Значение параметра обычно вводится в поле ввода или раскрывающийся список, соответствующий этому параметру. Но вы также можете задать параметр как результат вычисления какого-либо *выражения* языка ActionScript. (Например, номер кадра у вас может быть не 10, а $x + 2$.) Чтобы сделать это, введите соответствующее выражение в поле ввода или раскрывающийся список и не забудьте включить расположенный правее флажок **Expression**.

Выше набора элементов управления, с помощью которого задаются параметры, находится небольшое серое текстовое поле. В этом поле отображается краткое описание выбранной вами в списке текста сценария операции. Также в нем показывается описание операции, выбранной вами в левом списке. Таким образом, вы всегда будете в курсе, что у вас в настоящее время выбрано в любом из этих списков.

Если вы хотите убрать эту строку, нажмите кнопку, расположенную под правым нижним углом строки справки и имеющую вид треугольной стрелки, направленной вверх. После щелчка по этой кнопке строка справки пропадет, а кнопка примет вид стрелки, направленной вниз, щелкнув по ней, вы вернете строку справки на место.

Также вы сможете получить более подробную информацию о выбранной операции. Для этого нажмите кнопку, показанную на рис. 20.4. После этого на экране появится панель **Reference** (см. рис. 2.12) с текстом справки по выбранной операции. Можно также выбрать пункт **View Reference** контекстного меню списка доступных операций, расположенного в левой части панели **Actions**.



Рис. 20.4. Кнопка вызова справки по выбранной операции языка ActionScript

Левая (список доступных операций) и правая (сам сценарий) части панели **Actions** отделены друг от друга довольно толстой серой линией. Вы можете перетаскивать ее мышью, меняя относительные размеры этих частей. Можно также нажать небольшую кнопку, расположенную как раз на этой линии, чтобы убрать совсем левую часть панели. Повторный щелчок по этой кнопке восстановит левую панель. Вы также можете делать двойные щелчки по самой серой линии — по ней попасть много легче, чем по кнопке.

Вдоль верхнего края панели расположен раскрывающийся список, в котором перечислены все сценарии, созданные вами в этом фильме. Пользуйтесь им, если сценариев в вашем фильме не очень много. (Если же их много, вероятно, лучше пользоваться Проводником Flash.)

Панель **Actions** в данный момент показывает все сценарии, привязанные к выделенному элементу фильма: кадру, кнопке или образцу-клипу. Если же вы хотите, чтобы она показывала какой-то выбранный вами сценарий, включите кнопку-выключатель, расположенную в правом верхнем углу панели **Actions**, как раз правее раскрывающегося списка сценариев.

Надо сказать также, что панель **Actions** может работать в двух режимах: обычном и профессиональном. В обычном режиме она имеет вид, показанный на рис. 20.1. Если же вы выберете профессиональный режим, то панель **Actions** примет вид, показанный на рис. 20.5. В профессиональном режиме в правой части панели появится окно текстового редактора, в котором вы будете вводить сценарий вручную. Такое достоинство панели **Actions**, без сомнения, оценят опытные пользователи Flash.

Чтобы переключиться в профессиональный режим, выберите пункт **Expert Mode** дополнительного меню панели или нажмите комбинацию клавиш <Ctrl>+<Shift>+<E>. Чтобы вернуться в обычный режим, вам следует выбрать пункт **Normal Mode** дополнительного меню панели или нажать комбинацию клавиш <Ctrl>+<Shift>+<N>. Можно также воспользоваться кнопкой **View Options** (рис. 20.6).

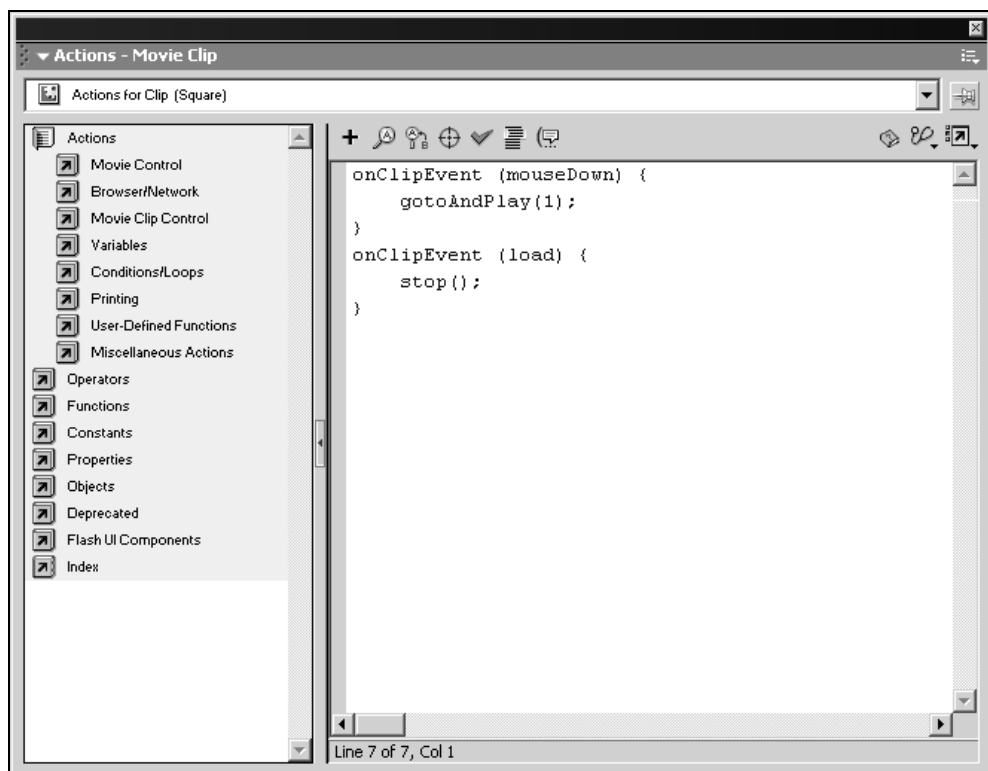


Рис. 20.5. Панель **Actions** (профессиональный режим)

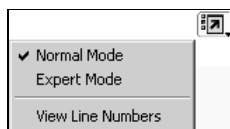


Рис. 20.6. Кнопка **View Options** и ее меню

Написание сценариев

Теперь мы рассмотрим, каким образом различным элементам фильма присваиваются сценарии. Откройте какой-либо фильм, сделанный нами ранее, например, фильм, описывающий процесс деления клетки (он был сделан в *главе 13*). На нем мы будем изучать написание сценариев.

Прежде всего, проверьте, находится ли панель **Actions** в обычном режиме работы. Откройте список доступных операций, если он закрыт. Переместите панель так, чтобы она не закрывала рабочий лист и временную шкалу. Теперь все готово к работе.

Первый свой сценарий мы привяжем к кадру — это проще всего. Помните, мы собирались "зациклить" наш фильм, но так, чтобы повторялась только его вторая сцена, где демонстрируется собственно деление клетки. (Первая сцена этого фильма, как вы помните, показывала его название, поэтому "зацикливать" фильм полностью не имеет смысла.) Для этого мы привяжем к последнему кадру второй сцены небольшой сценарий, "откручивающий" фильм на первый кадр этой сцены и начинающий воспроизведение с него. Этот сценарий выполнится, как только Flash воспроизведет данный кадр.

Сказано — сделано! Выделите последний кадр второй сцены. И переключитесь на панель **Actions**.

Выше уже говорилось, что для того, чтобы переместить указатель на другой кадр, нужно использовать действие `gotoAndPlay`. Все действия находятся в ветви **Actions** списка доступных операций (action — по-английски "действие"). В этой ветви мы раскрываем подветвь **Movie Control** (по-английски "управление фильмом") и находим пункт **goto**. Это как раз то, что нам надо. Теперь нам нужно добавить это действие в текст нашего сценария, который пока что пуст.

Чтобы добавить какую-либо операцию языка ActionScript из списка доступных операций в текст сценария, вы можете выполнить одно из перечисленных ниже действий:

- ☐ перетащить ее мышью в нужное место текста сценария;
- ☐ дважды щелкнуть по ней;
- ☐ выбрать пункт **Add to Script** в контекстном меню выделенной позиции списка;
- ☐ нажать кнопку, показанную на рис. 20.7, "проследовать" по иерархии подменю и выбрать нужный пункт;
- ☐ нажать комбинацию клавиш, соответствующую требуемой операции.

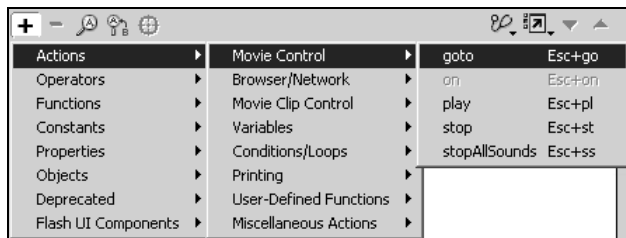


Рис. 20.7. Кнопка добавления операции в текст сценария и ее раскрытое подменю

О комбинациях клавиш, служащих для добавления в сценарий операций языка ActionScript, нужно рассказать подробнее. Эти комбинации состоят из трех клавиш: двух алфавитно-цифровых и клавиши `<Esc>`. Нажимать их

нужно одновременно — запомните это. И запомните еще, что перед тем, как пользоваться этими комбинациями, активизируйте список, в котором отображается текст сценария, в иерархическом списке доступных операций они не действуют.

А где узнать эти комбинации? Они написаны в подменю кнопки, показанной на рис. 20.7. Поэтому, если вы часто пользуетесь этой кнопкой, вы очень быстро выучите их наизусть. Однако, если не пользуетесь этой кнопкой, то можете включить отображение кодов прямо в списке доступных операций. Для этого вам следует включить пункт-выключатель **View Esc Shortcut Keys** в дополнительном меню панели.

Итак, вы выбрали действие `gotoAndPlay` и добавили его в текст сценария одним из описанных выше способов. Это действие тотчас появится в списке текста сценария, а над этим списком вы увидите набор элементов управления, служащих для задания параметров действия (см. рис. 20.3). Рассмотрим их подробнее.

С помощью набора из двух переключателей — **Go to and Play** и **Go to and Stop** — вы можете задавать поведение Flash после перехода на заданный вами кадр. Первый переключатель заставляет Flash продолжить воспроизведение с заданного кадра, а второй — остановиться на нем. Фактически эти два переключателя позволяют вам выбрать два различных действия: `gotoAndPlay` или `GotoAndStop`.

В раскрывающемся списке **Scene** можно выбрать сцену фильма, в которой находится нужный вам кадр. Здесь имеются как стандартные пункты, позволяющие выбрать текущую (**<current scene>**), предыдущую (**<previous scene>**) или следующую (**<next scene>**) сцену, так и пункты, позволяющие выбрать сцену по ее имени.

Раскрывающийся список **Type** позволяет выбрать способ задания нужного кадра. В нем имеется пять пунктов:

- ☐ **Frame Number** — задается номер кадра;
- ☐ **Frame Label** — задается имя кадра;
- ☐ **Expression** — номер кадра вычисляется с помощью арифметического выражения;
- ☐ **Next Frame** — переход на следующий кадр;
- ☐ **Previous Frame** — переход на предыдущий кадр.

В раскрывающемся списке **Frame** вводится номер, имя кадра или арифметическое выражение, результат которого и станет номером либо именем кадра. Этот список доступен, если в раскрывающемся списке **Type** был выбран пункт **Frame Number**, **Frame Label** или **Expression**.

Вот и все элементы управления. В нашем случае нужно включить переключатель **Go to and Play**, выбрать пункт **<current scene>** в списке **Scene**, пункт

Frame Number — в списке **Type** и ввести 1 в список **Frame**. После этого ваш сценарий примет такой вид:

```
gotoAndPlay(1);
```

А кадр, к которому вы привязали сценарий, будет выглядеть так, как показано на рис. 20.8.



Рис. 20.8. Кадр, к которому был привязан сценарий

Теперь вы можете проверить ваш сценарий в действии. Для этого запустите просмотр вашего фильма, выбрав пункт **Play** меню **Control** или нажав клавишу <Enter>. Но перед этим проверьте, включен ли пункт-выключатель **Enable Simple Frame Actions** в меню **Control**, иначе ваши сценарии не будут работать. И не забудьте сохранить фильм.

Все! Мы "зациклили" свой фильм так, что повторяется только сцена деления клетки. И сделали это добавлением совсем короткого сценария на языке ActionScript.

В этом примере мы поместили свой сценарий в том же слое, где "крутится" наша анимация. Но если таких сценариев много, рекомендуется помещать их в отдельный слой под названием **Actions** или **Сценарии**. Если вы последуете этому совету, ваши сценарии всегда будут на виду, и вам не придется искать их по всем слоям, которых в сложном фильме может быть не один десяток.

Привязать сценарий к кадру — нетрудная задача. А как насчет экземпляра образца-клипа (далее — *экземпляр-клип* или просто *клип*)?

Для экспериментов возьмем фильм, где демонстрировались возможности вложенной анимации. (Помните этот фильм: прямоугольник, кувыркаясь, летит по рабочему листу?) Загрузите его. И выполните несколько подготовительных действий:

- ☐ преобразуйте экземпляр графического образца в экземпляр-клип, для чего выберите в раскрывающемся списке, расположенном в верхнем левом углу редактора свойств, пункт **Movie Clip**;
- ☐ задайте для клипа имя, например, `Clip`.

После этого сохраните фильм. Вот теперь все готово для написания сценариев.

Давайте дадим нашему фильму немного "жизни". А именно, сделаем его интерактивным. Путь при щелчке мышью на вложенном клипе он останавливает свое проигрывание, переставая крутиться. Конечно, пользы от такой интерактивности никакой, но, по крайней мере, это наш первый опыт в создании интерактивных фильмов Flash.

Выделите клип на рабочем листе. Выведите на экран панель **Actions**. Переместите так, чтобы она не загромождала клип. И приготовьтесь к небольшой лекции по теории Flash-программирования.

Образец-клип — очень мощное средство придания фильмам Flash интерактивности. Язык **ActionScript** позволяет привязывать сценарии различным событиям, которые могут произойти с клипом: завершение загрузки, щелчок мышью, нажатие клавиши и др. Поэтому, если вы собираетесь создавать интерактивные фильмы, следует элементы, "отвечающие" за эту интерактивность, оформлять в виде образцов-клипов.

Для того чтобы присвоить какому-либо событию сценарий, нужно воспользоваться специально для этого предусмотренным действием `onClipEvent`. Это действие задает событие, в ответ на которое будет срабатывать сценарий. А сам сценарий должен иметь такой вид:

```
onClipEvent(<Обозначение события>) {  
    <Текст сценария>  
}
```

Пользуясь действием `onClipEvent`, вы можете привязывать к одному и тому же клипу несколько обработчиков разных событий. Так часто и делается при создании сложных интерактивных фильмов.

Итак, найдем в списке доступных операций действие `onClipEvent`. Оно находится в подветви **Movie Clip Control** ветви **Actions**. Добавьте это действие в текст сценария, пока что пустой. У вас должно получиться нечто, похожее на рис. 20.9.

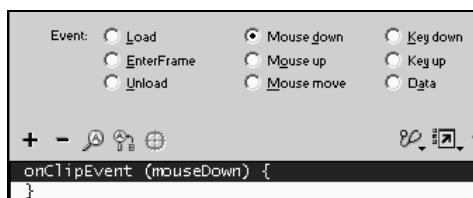


Рис. 20.9. Действие в тексте сценария и набор элементов управления для задания его свойств

Для задания единственного параметра этого действия — типа события — служит набор переключателей **Event**. Мы не будем перечислять все переключатели и события, им соответствующие, — все это во всех подробностях описано в *приложении 1*. Нас в данном случае интересует только переключатель **Mouse down**, задающий событие `mouseDown`, наступающее, когда, пользователь помещает над клипом курсор мыши и нажимает ее левую кнопку. Включите этот переключатель.

Мы только что задали *заголовок* сценария-обработчика события. Flash будет знать, на какое событие откликаться. Но он пока еще не знает, что нужно

делать в ответ на это событие. Нам нужно задать *тело* обработчика, которое и определяет поведение клипа в ответ на событие.

Немедленную остановку клипа выполняет действие **stop**. Никаких параметров у него нет. Оно находится в подветви **Movie Control** ветви **Actions**. Переносим его мышью в текст сценария так, чтобы оно "попало" точно между открывающей и закрывающей фигурными скобками. Окончательный текст сценария будет выглядеть так:

```
onClipEvent(mouseDown) {  
    stop();  
}
```

Теперь запустите фильм на проигрывание. Просто нажать <Enter> здесь будет мало — придется выбрать пункт **Test Movie** в меню **Control** или нажать комбинацию клавиш <Ctrl>+<Enter>. После того, как фильм будет открыт в новом окне и запущен на проигрывание, попробуйте щелкнуть мышью по вращающемуся прямоугольнику (как вы помните, это и есть наш клип). Если же вы все-таки попадете по нему (а это непросто), он тотчас остановится и долетит до места назначения, застыв в той позе, в которой его настиг роковой щелчок мыши.

На этой возвышенной ноте закончим рассказ о написании сценариев. Сохраните фильм и закройте его — больше он нам не понадобится.

Работа с панелью **Actions**

Здесь мы рассмотрим различные приемы работы с панелью **Actions**. Мы опишем все ее инструменты, которые помогут вам в работе над сценариями. Также мы расскажем о том, как работать в профессиональном режиме, предусмотренном для этой панели.

Работа в обычном режиме

Панель **Actions**, находящаяся в обычном режиме работы, была показана на рис. 20.1. Чтобы переключиться в обычный режим, выберите пункт **Normal Mode** дополнительного меню панели или меню кнопки **View Options** (см. рис. 20.6) или нажмите комбинацию клавиш <Ctrl>+<Shift>+<N>.

Как вы помните, в этом режиме Flash берет на себя написание кода сценария, а вам остается только задавать параметры операций, пользуясь привычными для вас элементами управления. Поскольку Flash ошибается очень редко, да и то, когда этого захотят разработчики, сценарии, написанные в этом режиме, избавлены от ошибок в синтаксисе (но не от ошибок в логике работы сценария). Обычный режим идеален для начинающих и малоопытных Flash-программистов.

Текст сценария в обычном режиме отображается как список с набором строк, каждая из которых представляет собой отдельную операцию сцена-

рия. Вы можете выбирать любую такую строку-операцию и выполнять над ней какие-либо действия. В частности, можно задавать параметры операции с помощью набора элементов управления, появляющихся над списком при выборе строки.

Вы также можете выделять сразу несколько строк. Для этого щелкните первую из них и продолжайте щелкать по остальным строкам, удерживая нажатой клавишу <Ctrl>. Если же вам нужно выделить непрерывную группу строк, то сначала щелкните первую строку в группе, а потом — последнюю, удерживая при этом нажатой клавишу <Shift>.

Предусмотрена возможность добавить какую-либо операцию языка ActionScript в текст сценария. Для этого существует целых пять способов, но самым удобным, на наш взгляд, является перетаскивание требуемой операции из списка доступных операций в нужное место сценария. Таким образом вы можете точно задать, где будет находиться новая операция.

Вы можете удалить ненужную операцию из списка, выбрав ее и нажав кнопку, показанную на рис. 20.10. Можно также нажать клавишу или выбрать пункт **Delete** в контекстном меню выделенной строки-операции.



Рис. 20.10. Кнопка удаления операции

Вы можете "вырезать" и копировать выделенную строку или строки в буфер обмена Windows. Для "вырезания" выберите пункт **Cut** в контекстном меню выделенной строки или нажмите комбинацию клавиш <Ctrl>+<X>. Чтобы скопировать выделенные строки в буфер обмена, воспользуйтесь пунктом **Copy** или комбинацией клавиш <Ctrl>+<C>.

Помещенные в буфер обмена строки вы можете вставить в другой сценарий. Для этого выделите строку, над которой они должны быть вставлены, и выберите пункт **Paste** контекстного меню или нажмите комбинацию клавиш <Ctrl>+<V>.

Есть возможность отменять последнюю совершенную операцию или повторять ее после отмены. Для этого выберите соответственно пункт **Undo** или **Redo** контекстного меню; также вы можете нажать комбинацию клавиш <Ctrl>+<Z> или <Ctrl>+<Y>.

Можно менять порядок строк в списке, перемещая их вверх или вниз. Для этого вам нужно воспользоваться парой кнопок, показанных на рис. 20.11. Левая кнопка перемещает выделенную в списке строку вниз, а правая — вверх. Также вы можете просто перетаскивать нужную строку на нужное место.



Рис. 20.11. Кнопки перемещения выделенной операции вниз и вверх по списку

Если вы случайно допустите ошибку в параметре какой-либо операции, Flash автоматически выявит ошибку и подсветит место, где она появилась, красным цветом (рис. 20.12). Таким образом, вы всегда будете знать, где допущен промах. Если Flash почему-то автоматически не определил ошибку, вы можете запустить проверку ошибок вручную, выбрав пункт **Check Syntax** в дополнительном меню панели или нажав комбинацию клавиш <Ctrl>+<T>. Flash проверит введенный вами код на правильность и выведет сообщение "The script contains no errors" (сценарий не содержит ошибок) или "This script contains errors" (сценарий содержит ошибки). Все найденные ошибки будут подсвечены красным цветом.

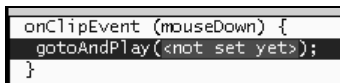


Рис. 20.12. В этом месте сценария допущена ошибка (не задан параметр — номер или имя кадра)

Если сценарий очень велик, а вам нужно быстро найти и заменить параметр какой-либо операции, вы можете воспользоваться встроенными в него возможностями поиска. Нажмите кнопку, показанную на рис. 20.13. Можно также выбрать в дополнительном меню панели пункт **Find** или нажать комбинацию клавиш <Ctrl>+<F>. В результате этого на экране появится диалоговое окно **Find** (рис. 20.14).



Рис. 20.13. Кнопка поиска



Рис. 20.14. Диалоговое окно **Find**

Задайте в поле ввода **Find what** требуемый параметр — строку или числовое значение, которое вам нужно найти, включите флажок **Match case**, если хотите, чтобы Flash учитывал при поиске регистр символов, и нажмите кнопку **Find Next**. Flash выберет строку, в которой встретился данный параметр.

Если же такого параметра найдено не будет, будет выведено предупреждение "Cannot find the string '<Ваша строка или числовое значение>'".

Чтобы продолжить поиск введенного значения далее по тексту сценария, снова нажмите кнопку **Find What** диалогового окна **Find**. Если же это окно уже закрыто вами (для чего достаточно нажать кнопку **Close**), выберите в дополнительном меню панели пункт **Find Next** или нажмите клавишу <F3>.

Диалоговое окно **Replace** позволяет вам сразу найти и заменить нужный параметр. Чтобы вызвать его на экран, нажмите кнопку, показанную на рис. 20.15. Вы также можете выбрать в дополнительном меню панели пункт **Replace** или нажать комбинацию клавиш <Ctrl>+<H>. Само окно **Replace** показано на рис. 20.16.



Рис. 20.15. Кнопка поиска и замены

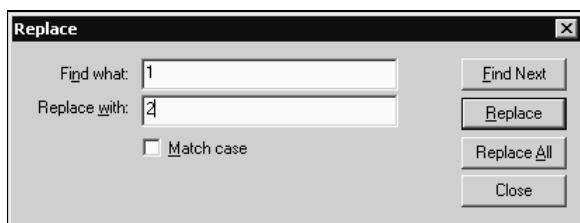


Рис. 20.16. Диалоговое окно **Replace**

Введите в поле **Find what** параметр, который вы хотите найти, а в поле **Replace with** — параметр, которым вы хотите его заменить, включите флажок **Match case**, если хотите, чтобы Flash учитывал при поиске регистр символов, и нажмите кнопку **Find Next**. Flash выберет строку, в которой встретился данный параметр. После этого вы можете нажать кнопку **Replace**, чтобы выполнить замену. Для продолжения поиска снова нажмите кнопку **Find Next**. Если же такого параметра найдено не будет, будет выведено уже знакомое вам предупреждение "Cannot find the string '<Ваша строка или числовое значение>'".

Вы можете одновременно заменить все найденные параметры. Для этого сразу же после ввода значений в поля нажмите кнопку **Replace All**. Для закрытия окна **Replace** нажмите кнопку **Close**.

Иногда бывает полезно пронумеровать строки сценария. Чтобы это сделать, включите пункт-выключатель **View Line Numbers** в дополнительном меню панели или меню кнопки **View Options** (см. рис. 20.6) или нажмите комбинацию клавиш <Ctrl>+<Shift>+<L>. А, выбрав пункт **Go to Line** или нажав

комбинацию клавиш <Ctrl>+<G>, можно быстро перейти на нужную строку, зная ее номер. Просто укажите этот номер в поле ввода **Line Number** диалогового окна **Go to Line** (рис. 20.17) и нажмите кнопку **OK**.



Рис. 20.17. Диалоговое окно **Go to Line**

Пункт **Print** дополнительного меню панели позволит вам распечатать все сценарии, созданные в фильме. При его выборе на экране появится стандартное диалоговое окно задания параметров печати Windows. Задайте эти параметры и нажмите кнопку **OK**.

Работа в профессиональном режиме

Панель **Actions**, находящаяся в профессиональном режиме, была показана на рис. 20.5. Чтобы переключиться в обычный режим, выберите пункт **Expert Mode** дополнительного меню панели или меню кнопки **View Options** (см. рис. 20.6) или нажмите комбинацию клавиш <Ctrl>+<Shift>+<E>.

В профессиональном режиме панель **Actions** предоставляет вам возможность самим вводить текст сценария. При этом вы будете сами отвечать за все допущенные синтаксические ошибки, однако и возможностей по написанию кода получите несравнимо больше, чем в обычном режиме. Поэтому профессиональный режим будет очень полезен опытным Flash-программистам.

В профессиональном режиме панель **Actions** выводит обычное поле текстового редактора, где вы и правите свои сценарии. Вы можете использовать любые приемы, знакомые вам по работе в программах редактирования текста, в частности, операции с буфером обмена Windows, "откат" последнего совершенного действия и "откат" "отката". Эти операции для обычного режима были описаны выше, но в профессиональном они работают точно так же.

Вы можете добавлять различные операции языка ActionScript в текст сценария. Все способы сделать это были описаны ранее. Однако, чтобы удалить ненужные операции или изменить их порядок в сценарии, вам придется пользоваться полем текстового редактора.

В профессиональном режиме также работают поиск и замена. В данном случае можно искать и заменять не только параметры операций, но и любой текст, который есть в вашем сценарии. Например, вы можете заменить одно действие на другое, воспользовавшись диалоговым окном **Replace** (см. рис. 20.16).

Есть также возможность выводить номера строк и переходить на нужные строки по их номеру. Эта операция работает так же, как и в обычном режиме.

Когда вы вводите текст сценария, Flash выполняет его анализ и раскрашивает различные его части в разные цвета. Это может помочь, чтобы найти ошибки в только что введенном коде. Но самый лучший способ — разумеется, проверка ошибок, которая запускается так же, как и в обычном режиме.

Внимание!

Если на вкладке **ActionScript Editor** диалогового окна **Preferences** был отключен флажок **Syntax Coloring**, код не будет расцвечиваться. С помощью расположенных ниже этого флажка селекторов цвета вы можете задать цвета различных элементов кода.

Flash предусматривает несколько возможностей, облегчающих работу программиста в профессиональном режиме панели **Actions**. Одна из них — *всплывающие подсказки* по коду. Сейчас мы расскажем, что это такое, и как ими пользоваться.

Когда вы открываете круглую скобку, готовясь ввести значение параметра какой-либо операции, чуть ниже текстового курсора появляется небольшая подсказка, описывающая назначение этого параметра (рис. 20.18). Когда вы введете значение первого параметра и поставите запятую, чтобы отделить его от второго, подсказка выдаст описание второго параметра, и т. д. Щелкая по небольшим стрелкам, направленным вправо и влево, вы можете перемещаться соответственно на следующий и предыдущий параметры и просматривать их описания.

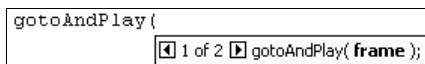


Рис. 20.18. Всплывающая подсказка

Есть еще одна разновидность подсказок по коду — *подсказки-списки*. Когда вы открываете скобку, чтобы задать параметр, значения которого должны выбираться из ограниченного списка, на экране появится небольшой список, в котором будут перечислены все эти значения (рис. 20.19). Вам останется только выбрать в списке нужное значение либо щелчком мыши, либо выделив его клавишами-стрелками и нажав клавишу <Enter>.

Также вы можете вызвать подсказку по коду на экран, нажав кнопку, показанную на рис. 20.20, выбрав пункт **Show Code Hint** в дополнительном меню панели или, что проще всего, если вы работаете с клавиатурой, нажав комбинацию клавиш <Ctrl>+<Space>.

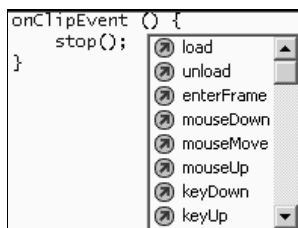


Рис. 20.19. Подсказка по коду



Рис. 20.20. Кнопка **Show Code Hint**

Внимание!

Если на вкладке **ActionScript Editor** диалогового окна **Preferences** был отключен флажок **Code Hints**, подсказки по коду не будут появляться автоматически. Однако даже в этом случае вы всегда сможете вывести их вручную.

Другая такая возможность — автоматическое создание отступов при вводе строк кода. В этом случае, если вы ввели строку с отступом от ее начала, то при нажатии клавиши <Enter> новая строка будет иметь такой же отступ. Также отступы формируются, если предыдущая строка заканчивается круглой или фигурной скобкой. Снабженный отступами код лучше читается, и разработчики Flash знают это.

Внимание!

Если на вкладке **ActionScript Editor** диалогового окна **Preferences** был отключен флажок **Automatic Indentation**, отступ не будет создаваться автоматически. В поле ввода **Tab Size** этого же окна вы можете задать величину отступа.

Можно установить отступ вручную, для чего выделите нужную строку целиком и нажмите клавишу <Tab>. Чтобы убрать отступ, также выделите строку и нажмите комбинацию клавиш <Shift>+<Tab>.

И еще одна возможность — автоматическое форматирование кода. Это означает, что вы можете кое-как ввести код, запустить автоматическое форматирование, и Flash сам его отформатирует. Для запуска автоматического форматирования кода нажмите кнопку, показанную на рис. 20.21. Можно также выбрать в дополнительном меню панели пункт **Auto Format** или нажать комбинацию клавиш <Ctrl>+<Shift>+<F>. Код будет автоматически отформатирован.



Рис. 20.21. Кнопка **Auto Format**

Вы можете задать параметры автоматического форматирования кода, выбрав пункт **Auto Formal Options** в дополнительном меню панели. На экране появится диалоговое окно **Auto Formal Options** (рис. 20.22). Это окно содержит набор флажков, с помощью которых и задаются параметры автоматического форматирования кода. Все эти флажки описаны в табл. 20.1.

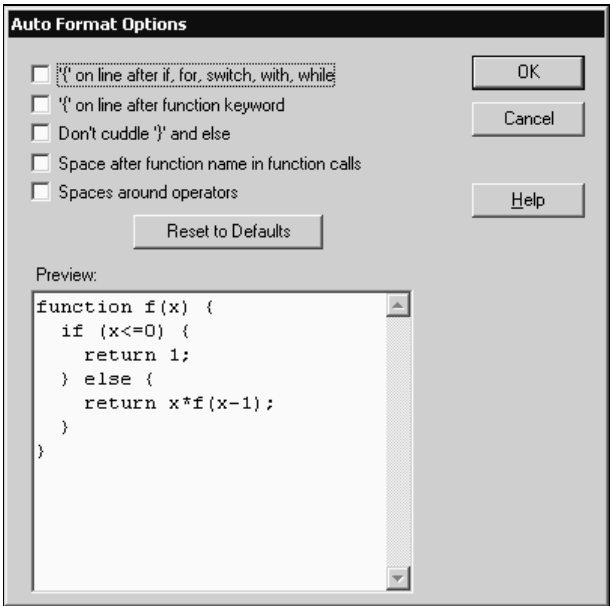


Рис. 20.22. Диалоговое окно **Auto Formal Options**

Таблица 20.1. Флажки диалогового окна **Auto Formal Options**

Название	Описание
'{' on line after If, for, switch, with, while	Если включен, Flash помещает открывающуюся фигурную скобку после ключевого слова на новой строке. Если отключен, Flash помещает ее на той же строке
'{' on line after function keyword	Если включен, Flash помещает открывающуюся фигурную скобку после определения функции на новой строке. Если отключен, Flash помещает ее на той же строке

Таблица 20.1 (окончание)

Название	Описание
Don't cuddle '}' and else	Если включен, то, по возможности, Flash помещает закрывающую фигурную скобку и ключевое слово <code>else</code> на разных строках. Если отключен, Flash помещает их на одной строке
Space after function name in function calls	Если включен, между именем функции и списком ее параметров вставляется пробел. Если отключен, пробел не вставляется
Spaces around operators	Если включен, перед каждым оператором и после него вставляются пробелы. Если отключен, пробелы не вставляются

В расположенной в нижней половине окна панели просмотра можно видеть, как включенные или отключенные флажки будут влиять на форматирование кода. А, нажав кнопку **Reset to Defaults**, вы сможете вернуться к установкам по умолчанию.

Задав нужные параметры форматирования кода, нажмите кнопку **ОК**. Если вы передумали менять их, нажмите кнопку **Cancel**.

Использование внешнего текстового редактора

Как ни силен Flash в редактировании сценариев, но некоторые программисты могут им пренебречь. Причин тому может быть несколько. Во-первых, опытный программист, как правило, использует специализированный текстовый редактор, "заточенный" именно под правку исходных текстов программ. Таких редакторов сейчас создано довольно много как универсальных, так и специализирующихся на каком-то одном языке программирования. Во-вторых, программист "старой закалки" может предпочесть простейшую программу наподобие Блокнота, поставляемого с Windows. Что ж, это его дело...

Flash допускает использование внешнего текстового редактора. И предлагает для работы с ним несколько полезных возможностей.

Прежде всего, вы можете импортировать текст сценария на языке ActionScript, хранящийся в текстовом файле. Такие файлы должны иметь расширение `as` (сокращение от "ActionScript"); Впрочем, это необязательно. Чтобы импортировать код из такого файла, выберите пункт **Import From File** в дополнительном меню панели или нажмите комбинацию клавиш `<Ctrl>+<Shift>+<I>`. На экране появится стандартное диалоговое окно открытия файла Windows, выберите в нем нужный файл и нажмите кнопку открытия.

Если вы хотите продолжить работу над уже созданным в среде Flash сценарием во внешнем текстовом редакторе, то вам будет полезна функция экс-

порта сценария в текстовый файл. Выберите пункт **Export As File** в дополнительном меню панели или нажмите комбинацию клавиш **<Ctrl>+<Shift>+<X>**. На экране появится стандартное диалоговое окно сохранения файла Windows, задайте имя сохраняемого файла и нажмите кнопку сохранения.

Есть еще один способ импортировать в фильм текстовый файл со сценарием — использовать действие `#include`. Это действие заставляет Flash импортировать сценарий из внешнего файла только когда файл начнет проигрываться, и для его проигрывания понадобится этот сценарий. При публикации или экспорте фильма внешние сценарии будут включены в его состав, и для проигрывания готового SWF-файла уже не понадобятся.

Действие `#include` находится в подветви **Miscellaneous Actions** ветви **Actions**. Оно принимает единственный параметр — путь и имя файла, содержащего сценарий. Запомните, что этот путь должен быть задан относительно FLA-файла фильма.

Поддержка сценариев Проводником Flash

Уже знакомый вам Проводник Flash, описанный в *главе 10*, кроме всего прочего, поддерживает и сценарии. Чтобы просмотреть их, включите кнопку-выключатель **Show Action Scripts**, находящуюся в окне Проводника. Также проверьте, включен ли флажок **ActionScript** в окне **Movie Explorer Settings** (см. рис. 10.31). На рис. 20.23 показано окно Проводника, в котором вы можете видеть текст созданного нами сценария, привязанного к экземпляру-клипу.

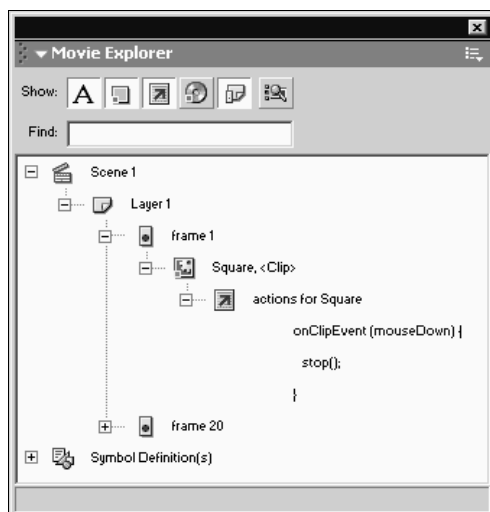
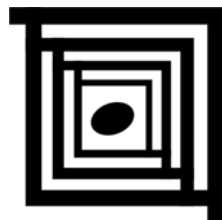


Рис. 20.23. Окно Проводника, отображающее сценарий, привязанный к экземпляру-клипу

Глава 21



Язык ActionScript

В этой главе будет описан язык ActionScript. Будет приведено описание основных конструкций этого языка, рассмотрен синтаксис и самые основные принципы написания сценариев. Конкретные примеры использования сценариев в интерактивных фильмах будут даны в последующих главах этой части. А полное описание языка ActionScript содержится в *приложении 1*.

Язык ActionScript — полноценный язык программирования, позволяющий описывать любое, даже очень сложное поведение элементов фильма. Однако он не имеет многих средств, доступных в других языках программирования (C, C++, Pascal, Basic и др.), например, не позволяет работать с дисками и файлами. Это сделано из соображений безопасности, чтобы злоумышленник не смог получить доступ к файлам пользователя, используя специально написанные сценарии.

Как уже упоминалось, язык ActionScript основан на популярном языке JavaScript, широко используемом в Web-дизайне для написания сценариев для Web-страниц. Поэтому, если вы знакомы с языком JavaScript, вам не составит труда изучить и ActionScript. Конечно, ActionScript имеет свои отличия, но основной синтаксис этих двух языков весьма схож.

Начала ActionScript

Здесь мы рассмотрим основные понятия этого языка программирования.

Что такое сценарий

Рассмотрим небольшой пример сценария, написанного на языке ActionScript:

```
function someFunction(x, y, z, t) {  
    var a, b, c;  
    a = x * y;  
    b = z/t;
```

```
c = a - b;  
return c;  
}
```

На первый взгляд, это набор каких-то формул (надо сказать, вполне понятных). Эти формулы описывают функцию, принимающую в качестве параметров четыре числа и производящую над ними какие-то вычисления. Так выглядит один из простейших сценариев ActionScript.

Приведенный выше сценарий, все-таки, несколько "абстрактен", что ли. Давайте вспомним сценарий, написанный нами в *главе 20*:

```
onClipEvent(mouseDown) {  
    stop();  
}
```

Мы знаем, что это такое. Это обработчик события, срабатывающий при щелчке мышью по экземпляру-клипу и останавливающий его проигрывание (экземпляра-клипа, а не всего фильма).

Итак, что такое сценарий? Или (для общего случая) что такое программа?

Сценарий (мы будем говорить о сценариях) — это набор *выражений*, написанных в соответствии с правилами данного языка программирования. Каждое выражение — это описание одного действия, выполняемого программой. Flash читает выражения одно за другим и выполняет их.

Выражения могут содержать *команды* и *данные*. Команды задают, что нужно сделать с данными. К командам относятся, в частности, хорошо знакомые вам *действия*, а также *операторы*, например, оператор умножения. Некоторые команды могут иметь *модификаторы*, изменяющие область действия или поведение команды. Данными же могут быть *константы* и *переменные*. Выражения могут включать также *ключевые слова*, имеющие особое значение.

Давайте возьмем одну строчку из нашего первого примера:

```
a = x * y;
```

В ней присутствуют две переменные и два оператора, переменные — *a*, *x* и *y*, а операторы — присваивания *=* и умножения ***. Сначала оператор умножения перемножает значения, хранящиеся в переменных *x* и *y*, а затем оператор присваивания помещает полученное произведение в переменную *a*. Как видите, приведенное выше выражение ничем не отличается от обычной математической формулы.

Запомните еще вот что. Каждое выражение ActionScript должно заканчиваться знаком точки с запятой *;*. Этот знак обозначает конец выражения.

Поскольку все сценарии (и вообще, все программы) занимаются тем, что преобразуют данные из одного вида в другой, сначала мы рассмотрим данные и их представление. А уже потом займемся командами.

Типы данных

Любые данные, которыми может манипулировать сценарий на языке ActionScript, должны относиться к определенному типу. *Тип данных* — это своего рода их разновидность: число, текстовая строка или что-то еще. Он определяет множество возможных значений данных и применимых к ним операций. Ниже перечислены и описаны все типы данных, поддерживаемые ActionScript.

Строковый

Строковые данные (или строки) — это последовательности символов, букв, цифр и знаков препинания, заключенные в одинарные или двойные кавычки. Например, такие:

```
"Flash MX"
```

```
"1234567"
```

```
'Строковые данные — это последовательности символов.'
```

Строки могут иметь любую длину, ограниченную только объемом свободной памяти компьютера. Точнее, существует предел в 2 гигабайта, но вряд ли вы будете использовать столь длинные строки.

Здесь нужно сказать немного о символах, из которых состоят строки. Каждый символ в компьютере обозначается особым *кодом*, однозначно его определяющим. Поэтому все строки представляют собой фактически набор кодов составляющих их символов.

Коды всех возможных символов составляют *кодировку* (также ее называют кодовой таблицей или набором символов). Таких кодировок существует довольно много, как правило, каждая операционная система поддерживает свою кодировку. Так, русская версия MS-DOS поддерживает кодировку 866, русская версия MS Windows — кодировку 1251, английские версии этих операционных систем — соответственно кодировки 437 и 1250. Такое многообразие кодировок вызвано тем, что приходится кодировать символы различных языков, в которых размер алфавита может быть очень разным.

Все упомянутые выше кодировки используют для обозначения символа двухбайтный код *ASCII*. В последнее время появилась кодировка *Unicode*, заменяющая все эти многочисленные кодировки и поддерживающая все символы всех популярных на Земле языков. Именно Unicode используется в MS Windows 2000 и XP, а также Flash MX для хранения текстовых данных.

Кроме букв, цифр и знаков препинания, строки могут содержать также *специальные символы*, служащие для особых целей. Все специальные символы, поддерживаемые Flash, перечислены в табл. 21.1.

Таблица 21.1. Специальные символы, поддерживаемые Flash

Символ	Описание	Код
\b	Забой	8
\f	Прогон листа	12
\n	Прогон строки	10
\r	Возврат каретки	13
\t	Табуляция	9
\"	Двойная кавычка	22
\'	Одинарная кавычка	27
\\	Обратная косая черта	92
\999	Любой символ по его восьмеричному коду (обозначен как 999)	—
\xFF	Любой символ по его шестнадцатеричному коду ASCII (обозначен как FF)	—
\xFFFF	Любой символ по его шестнадцатеричному коду Unicode (обозначен как FFFF)	—

Таким образом, если нам требуется поместить в строку двойные кавычки, нужно записать так:

```
"\"Macromedia Flash\" - пакет векторной графики и анимации"
```

Числовой

Числовые данные — это обычные числа, которые можно складывать и вычитать, извлекать из них квадратный корень и вычислять тангенс. Эти числа могут быть как целыми, так и дробными, в последнем случае целая и дробная части разделяются точкой (не запятой!). Примеры чисел:

```
13756
454.7873
0.5635
```

Целые числа могут также быть заданы в восьмеричной и шестнадцатеричной системах счисления. (Нецелые числа могут быть только десятичными.) Восьмеричные числа задаются цифрами от 0 до 7 и обязательно должны начинаться с нуля. А шестнадцатеричные числа задаются цифрами от 0 до F и должны начинаться с 0x. Примеры восьмеричного и шестнадцатеричного чисел:

```
035652
0x5F8C
```

Для задания нецелых чисел может быть использована экспоненциальная форма вида <мантисса>Е<порядок>. Примеры таких чисел (в скобках написано "обычное" математическое представление):

1E-5 (10^{-5})

8.546E23 ($8,546 \times 10^{23}$)

Логический

Логическая величина может принимать только два значения: `true` и `false` — "истина" и "ложь" — обозначаемые соответственно ключевыми словами `true` и `false`. Логические величины часто используются в операциях сравнения.

Объект, клип и функция

Объект, клип и функция — это сложные структуры данных, которые мы рассмотрим далее в этой главе. Здесь мы упомянем их только для того, чтобы вы знали об их существовании.

null

Величина *null* обозначает отсутствие данных. Обозначается ключевым словом `null`.

undefined

Величина *undefined* указывает, что переменной не было присвоено никакое значение. Обозначается ключевым словом `undefined`. Обратите внимание: это не то же самое, что `null`!

Константы

Константы (постоянные величины) — это данные, значение которых не может изменяться. Константами являются, в частности, все числа, строки, логические и специальные значения, записанные в соответствии с правилами конкретного типа данных:

26756

"Строка"

true

null

ActionScript предоставляет разработчику также некоторое количество других констант, заданных ключевыми словами. Подобные константы мы рассмотрим далее в этой главе.

Переменные

Для того чтобы успешно обрабатывать данные, сценарию нужно где-то их хранить. Для этого выделяются специальные участки памяти, называемые

переменными. Сценарий может обращаться к этим участкам памяти по имени, помещать в них данные любого типа и читать их.

Именованние переменных

Каждая переменная должна иметь имя, которое однозначно идентифицирует ее. Короче говоря, не должно быть двух переменных с одним и тем же именем (об исключениях из этого правила поговорим попозже). Также не допускается совпадение имени переменной и ключевого слова языка ActionScript, т. е. слова, используемого для написания выражений.

Имя переменной должно состоять из латинских букв, цифр, знаков доллара \$ и подчеркивания _, причем первым символом должны быть либо латинская буква, либо знак доллара или подчеркивания. Например, `FrameNumber`, `_link`, `UserName` — правильные имена переменных, а `678vasya`, `НомерКадра` — неправильные. Язык ActionScript нечувствителен к регистру символов, которыми набраны имена переменных, т. е. `framenumber` и `FrameNumber` — одна и та же переменная.

Для выбора имен переменных специалисты фирмы Macromedia рекомендуют следовать одному простому правилу: имя должно быть "говорящим". Это значит, что имя должно говорить о назначении переменной. Например, переменную, в которой хранится номер кадра, лучше всего назвать `frameNumber` — так будет сразу понятно, зачем она нужна. Но не переусердствуйте: имена типа `lastVisitedFormFrameNumber` очень трудно набирать. Для временных переменных лучше всего брать однобуквенные имена: `i`, `a`, `x`, `y` и т. п.

Объявление и использование переменных

Прежде, чем переменная будет использована, она должна быть *объявлена*. Чтобы объявить переменную, ей достаточно присвоить какое-либо значение, воспользовавшись оператором присваивания:

```
x = 34;
```

Мы только что объявили переменную по имени `x`.

Вы также можете использовать для присвоения значения переменной действие `set(<имя переменной>, <значение>):`

```
set(x, 34);
```

Но этот способ менее нагляден.

Что можно делать с объявленными переменными? Все, что угодно: присваивать им какое-либо число, извлекать из них содержимое, производить над ним какие-то операции. В общем, хранить исходные значения, промежуточные или окончательные результаты вычислений, делать то, для чего они, собственно, и предназначены.

```
a1 = x + y;  
x = x + 3;
```

Второе выражение, кстати, абсолютно правильно с точки зрения синтаксиса ActionScript, хоть и выглядит нелепым. Сначала мы извлекаем значение переменной `x`, прибавляем к нему 3 и вновь помещаем в переменную `x`. Такие выражения часто используются не только в ActionScript, но и в других языках программирования.

Вы можете присвоить переменной `x` значение любого другого типа. После этого старое значение пропадет, и переменная `x` фактически сменит тип:

```
x = "Flash MX";  
x = someObject;
```

где `someObject` — переменная объектного типа.

Имейте только в виду, что объявление переменной должно предшествовать ее использованию. Если же вы попытаетесь обратиться к переменной перед тем, как объявить ее, переменная вернет значение `undefined` (значение не определено).

Видимость переменных

Теперь нам нужно поговорить об одном важном моменте в "жизни" любой переменной. Это ее *видимость*, определяющая, какие выражения смогут получить доступ к этой переменной, а какие — не смогут.

Если вы объявили переменную, как было показано выше, к ней сможет обратиться любое выражение в любом сценарии, присвоенном текущему клипу. Таким образом, если вы объявили переменную `a` в первом кадре клипа, то к ней может получить доступ любой сценарий, присвоенный любому кадру, кнопке или вложенному клипу, содержащемуся в этом клипе. Однако, из клипов, не вложенных в текущий клип, и основного фильма доступ к ней не получит никто.

Такие переменные называются *переменными уровня клипа* или уровня временной линии. Говорят, что такие переменные "видны" только в "своем" клипе.

Если вы хотите, чтобы ваша переменная была "видна" в основном фильме и всех вложенных в него клипах, вам нужно специально уведомить об этом Flash. Делается это с помощью указания модификатора `_global`, который добавляется перед именем переменной и отделяется от него точкой:

```
_global.someVar = 0;
```

Такие переменные называются *глобальными*. Говорят, что глобальные переменные "видны отовсюду".

Существует еще один тип видимости переменных — *локальные* переменные. Локальные переменные объявляются внутри функции и видимы только в ней. Для объявления локальных переменных используется действие `var`:

```
var i, a, x, someVar = 0;
```

Выше говорилось, что каждая переменная должна иметь уникальное имя, по которому к ней можно будет обратиться. Из этого правила есть одно исключение: переменные, имеющие разную видимость, могут иметь одинаковые имена. При этом обращение происходит к переменной, имеющей более "узкую" область видимости.

Проиллюстрируем сказанное на примере:

```
_global.someVar = "Value";
a = someVar;
. . .
someVar = 3;
s = someVar;
. . .
function someFunc() {
    var someVar = false;
    es = someVar;
    . . .
}
```

В переменную `a` попадет значение `"Value"`, взятое из глобальной переменной `someVar`. Переменная `s` получит значение `3`, т. к. будет выполнено обращение к переменной уровня клипа `someVar`, которая к тому времени уже будет объявлена. Ну, а переменная `es` получит значение `false`, взятое из локальной переменной `someVar`.

Как видите, переменная с более "узкой" областью видимости как бы скрывает собой переменную с более "широкой" областью видимости. Это может принести пользу, но часто служит источником досадных ошибок. Будьте внимательны!

Если же вы хотите обратиться к переменной, находящейся в другом клипе, укажите имя этого клипа перед именем переменной, разделив их точкой:

```
someClip.someVar = 0;
```

Системные переменные

Язык ActionScript также предлагает несколько заранее определенных переменных. Эти переменные используются для различных служебных целей и называются *системными*.

Так, переменная `_quality` задает качество воспроизведения фильма Flash. Она может принимать одно из четырех значений: `"LOW"` (низкое качество), `"MEDIUM"` (среднее), `"HIGH"` (высокое) и `"BEST"` (наивысшее). Эта переменная уже определена, вам остается только задать нужное значение:

```
_quality = "MEDIUM";
```

Полный список и подробное описание всех системных переменных см. в *приложении 1*.

Операторы

Мы только что закончили рассмотрение данных, констант и переменных. Теперь пора приступить к рассмотрению операторов, предназначенных для манипуляций с этими данными.

Арифметические

Арифметические операторы служат для выполнения математических действий над значениями констант и переменных. Все арифметические операторы, поддерживаемые `ActionScript`, перечислены в табл. 21.2.

Таблица 21.2. Арифметические операторы `ActionScript`

Оператор	Описание
-	Смена знака
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления
++	Инкремент (увеличение на единицу)
--	Декремент (уменьшение на единицу)

Примеры выражений, в которых использовались арифметические операторы, приводились уже неоднократно. На всякий случай, приведем еще несколько:

```
l = r * 3.14;
```

```
t = r++;
```

```
f = -e / 2;
```

Арифметические операторы делятся на две группы: *унарные* и *бинарные*. Унарные операторы выполняются над одним операндом, к ним относятся операторы смены знака, инкремента и декремента. Бинарные операторы, к которым относятся все остальные перечисленные в табл. 21.2, всегда выполняются над двумя операндами.

Здесь нужно сказать несколько слов об операторах инкремента и декремента. Дело в том, что они могут ставиться как перед операндом, так и после

него. Если оператор инкремента стоит перед операндом, то сначала значение операнда инкрементируется, а уже потом используется в дальнейших вычислениях. Если же оператор инкремента стоит после операнда, то его значение сначала используется в других вычислениях, а уже потом инкрементируется. Это проще показать на примере:

```
r = 3;  
s = ++r;  
t = r++;
```

После выполнения первого выражения в переменной *r* окажется 3. После выполнения второго в переменных *s* и *r* окажется значение 4. А после третьего в переменной *t* будет 4, а в *r* — 5.

Точно так же ведет себя оператор декремента.

Операторы инкремента и декремента рекомендуется использовать, если значение какой-либо переменной нужно увеличить или уменьшить на единицу. Эти операторы выполняются быстрее, чем операторы сложения и вычитания.

Объединения строк

Оператор объединения строк + позволит соединить две строки в одну. Например, сценарий:

```
s1 = "Flash";  
s2 = "MX";  
s = s1 + s2;
```

поместит в переменную *s* строку "FlashMX".

Двоичные

Двоичные операторы (называемые также побитными или поразрядными) затрагивают двоичное представление чисел. Все они перечислены в табл. 21.3.

Таблица 21.3. Двоичные операторы ActionScript

Оператор	Описание
~	Инверсия (НЕ)
&	Умножение (И)
	Сложение (ИЛИ)
^	Исключающее сложение (исключающее ИЛИ)
<<	Сдвиг разрядов влево

Таблица 21.3 (окончание)

Оператор	Описание
>>	Сдвиг разрядов вправо
>>>	Сдвиг разрядов вправо с заполнением нулями

Двоичные операторы также делятся на унарные и бинарные. К унарным операторам относится оператор инверсии, к бинарным — все остальные. Операторы сдвига — также бинарные: слева указывается число, разряды которого нужно сдвинуть, а справа — на сколько знаков их нужно сдвинуть. Например:

```
f = a >> 3;
```

Присваивания

Нам уже знаком оператор простого присваивания `=`. С его помощью можно выполнять присвоение значения переменной:

```
a = 2;
```

```
b = c = 3;
```

Второе выражение примера выполняет присвоение значения 3 сразу двум переменным — `b` и `c`.

Кроме того, ActionScript поддерживает *операторы сложного присваивания*. Такие операторы позволяют выполнять присваивание одновременно с другой операцией:

```
a = a + b;
```

```
a += b;
```

Два приведенных выше выражения эквивалентны. Просто во втором был использован оператор сложного присваивания `+=`.

Все эти операторы сложного присваивания, поддерживаемые ActionScript, и их эквиваленты приведены в табл. 21.4.

Таблица 21.4. Операторы сложного присваивания

Оператор	Эквивалентное выражение
<code>a += b;</code>	<code>a = a + b;</code>
<code>a -= b;</code>	<code>a = a - b;</code>
<code>a *= b;</code>	<code>a = a * b;</code>
<code>a /= b;</code>	<code>a = a / b;</code>
<code>a %= b;</code>	<code>a = a % b;</code>

Таблица 21.4 (окончание)

Оператор	Эквивалентное выражение
<code>a <= b;</code>	<code>a = a << b;</code>
<code>a >= b;</code>	<code>a = a >> b;</code>
<code>a >>= b;</code>	<code>a = a >>> b;</code>
<code>a &= b;</code>	<code>a = a & b;</code>
<code>a ^= b;</code>	<code>a = a ^ b;</code>
<code>a = b;</code>	<code>a = a b;</code>

Операторы сложного присваивания выполняются быстрее, т. к. фактически представляют собой один оператор.

Сравнения

Операторы сравнения сравнивают два операнда и возвращают логическое значение. Если условие сравнения выполняется, возвращается "истина" (true), если не выполняется — "ложь" (false).

```
a1 = 2 < 3;  
a2 = -4 > 0;  
a3 = r < t;
```

Переменная `a1` получит значение `true` (2 больше 3), переменная `a2` — значение `false` (-4 не может быть больше нуля), а значение переменной `a3` будет зависеть от значений переменных `r` и `t`.

Все поддерживаемые ActionScript операторы сравнения сведены в табл. 21.5.

Таблица 21.5. Операторы сравнения

Оператор	Описание
<code><</code>	Меньше
<code>></code>	Больше
<code>==</code>	Равно
<code><=</code>	Меньше или равно
<code>>=</code>	Больше или равно
<code>!=</code>	Не равно
<code>===</code>	Строго равно
<code>!==</code>	Строго не равно

С первыми шестью операторами все понятно. Но на двух последних операторах — "строго равно" и "строго не равно" — нужно остановиться подробнее. Это операторы так называемого *строгого сравнения*. Обычные операторы "равно" и "не равно", если встречаются операнды разных типов, пытаются преобразовать их к одному типу (о преобразованиях типов см. далее в этой главе). Операторы строгого равенства и строгого неравенства такого преобразования не делают, а в случае несовместимости типов операндов возвращают `false`. Иногда это бывает полезно, например, при проверке ввода пароля пользователя.

Логические

Логические операторы сравнивают два логических значения и возвращают третье. Все они перечислены в табл. 21.6. А в табл. 21.7 и табл. 21.8 показаны результаты выполнения этих операторов.

Таблица 21.6. Логические операторы

Оператор	Описание
!	НЕ (логическая инверсия)
&&	И
	ИЛИ

Таблица 21.7. Результаты выполнения операторов И и ИЛИ

Операнд 1	Операнд 2	&& (И)	(ИЛИ)
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Таблица 21.8. Результаты выполнения оператора НЕ

Операнд	! (НЕ)
true	false
false	true

Основная область применения логических операторов — операции над логическими значениями и выражения сравнения. Приведем пару примеров таких выражений:

```
a = (b > 0) && (c + 1 != d);
flag = !(status == 0);
```

Оператор *typeof*

Оператор `typeof` принимает в качестве операнда переменную или выражение и возвращает строку, описывающую тип данных операнда. Все значения, которые он может вернуть, перечислены в табл. 21.9.

Таблица 21.9. Значения, возвращаемые оператором *typeof*

Тип данных	Возвращаемое значение
Строковый	"string"
Клип	"movieclip"
Объект	"object"
Числовой	"number"
Логический	"boolean"
Функция	"function"
null	"null"
undefined	"undefined"

Оператор `typeof` может использоваться, например, так:

```
status = typeof(somevar);
```

Здесь мы присваиваем результат оператора `typeof` переменной `status`. Впоследствии он может быть использован, например, в условном выражении.

Совместимость типов данных

И напоследок рассмотрим еще два важных вопроса: *совместимость типов данных и преобразование* одного типа к другому.

Что случится, если сложить два числовых значения? Правильно, еще одно числовое значение. А если сложить число и строку? Тут Flash сталкивается с проблемой несовместимости типов данных. И пытается сделать эти типы совместимыми, преобразуя один из них к другому. Сначала он пытается преобразовать строку в число и, если это удастся, выполняет сложение. В противном случае число будет преобразовано в строку, и две полученные строки будут объединены:

```
var a, b, c, d, e, f;  
a = 11;  
b = "12";  
c = a + b;  
d = "Flash";  
e = 6;  
f = d + e;
```

Значение переменной `b` будет преобразовано в числовой тип, таким образом, переменная `c` будет содержать значение 23. Но так как значение переменной `d` не может быть преобразовано в число, то значение `e` будет преобразовано в строку, и результат — значение `f` — станет равным "Flash6".

Логические величины преобразуются либо в числовые, либо в строковые, в зависимости от конкретного случая. Значение `true` будет преобразовано в 1 или "1", а значение `false` — в 0 или "0".

Как видите, Flash изо всех сил пытается выполнить ваш сценарий правильно. Иногда это получается, но все работает не так, как планировалось, или, в конце концов, останавливается по ошибке совсем в другом месте, на абсолютно верном операторе. Поэтому лучше всего не допускайте подобных казусов, оперируйте только переменными совместимых типов и выполняйте преобразование типов сами.

Для выполнения преобразования типов вы можете воспользоваться особыми функциями преобразования типов. Эти функции описаны в *приложении 1*.

Приоритет операторов

Последний вопрос, который мы здесь на конкретных примерах рассмотрим, — это порядок выполнения операторов.

Пусть имеется следующее выражение:

```
a = b + c - 10;
```

В этом случае сначала к значению переменной `b` будет прибавлено значение `c`, а потом из суммы будет вычтено 10. Операторы этого выражения выполняются строго слева направо.

```
a = b + c * 10;
```

А здесь этот принцип не действует! Сначала будет выполнено умножение значения `c` на 10, а уже потом к произведению будет прибавлено значение `b`. Обычный порядок выполнения операторов "строго слева направо" будет нарушен. А все потому, что умножение имеет больший *приоритет*, чем сложение.

А оператор присваивания `=` имеет самый низкий приоритет. Вот поэтому сначала вычисляется выражение, а потом его результат присваивается какой-либо переменной.

В общем, основной принцип выполнения всех операторов таков: сначала выполняются операторы с более высоким приоритетом, а уже потом — операторы с более низким. Операторы с одинаковым приоритетом выполняют-ся слева направо.

В табл. 21.10 перечислены все изученные нами операторы в порядке убывания приоритета. Более полная таблица приоритетов приведена в *приложении 1*.

Таблица 21.10. Приоритет операторов (в порядке убывания)

Операторы	Описание
++ -- - ~ ! typeof	Инкремент, декремент, смена знака, двоичная инверсия, логическое НЕ, типа
* / %	Умножение, деление, взятие остатка
+ -	Сложение, объединение строк, вычитание
<< >> >>>	Двоичные сдвиги
< > <= >= = !=	Сравнения
&	Двоичное умножение
^	Двоичное исключающее сложение
	Двоичное сложение
&&	Логическое И
	Логическое ИЛИ
= {Оператор}=	Присваивание, простое и сложное

Вам следует запомнить эту таблицу. Неправильный порядок выполнения операторов может стать причиной трудно выявляемых ошибок, когда внешне абсолютно правильное выражение дает неверный результат.

Но что делать, если нам нужно нарушить обычный порядок выполнения операторов? Воспользуйтесь скобками:

```
a = (b + c) * 10;
```

Здесь сначала выполняется сложение `b` и `c`, а уж потом сумма будет умножена на 10. Вы видите, что первыми выполняются операторы, заключенные в скобки.

Операторы в скобках также подчиняются приоритету. Поэтому часто используют многократно вложенные скобки:

```
a = ((b + c) * 10 - d) / 2 + 9;
```

Здесь операторы будут выполнены в такой последовательности:

1. Сложение `b` и `c`.
2. Умножение суммы на 10.
3. Вычитание `d` из произведения.
4. Деление разности на 2.
5. Прибавление 9 к частному.

Если удалить скобки:

```
a = b + c * 10 - d / 2 + 9;
```

то порядок выполнения операторов будет таков:

1. Умножение `c` и 10.
2. Деление `d` на 2.
3. Сложение `b` и произведения `c` и 10.
4. Вычитание из полученной суммы частного от деления `d` на 2.
5. Прибавление 9 к полученной разности.

Совсем другой результат, не так ли?

Действия

Действия служат для выполнения каких-либо (извините за каламбур) действий над фильмом, встроенными клипами или другими его элементами. Действиями (actions) в языке ActionScript называется то, что в других языках программирования называется командами, но мы не будем употреблять этот термин, чтобы избежать путаницы. Благо термин "команда" мы употребили ранее совсем в другом контексте.

Собственно, вы уже знакомы с действиями. Действия стали основой для самых простых наших сценариев. Многие действия самодостаточны, как, например, уже знакомое вам действие `stop`, останавливающее проигрывание фильма или клипа. Иногда действия требуют параметров, к таким действиям относится `gotoAndPlay`. Многие действия входят в состав сложных выражений, которые мы рассмотрим далее, и не могут использоваться вне их.

Полное описание всех действий приведено в *приложении 1*.

Комментарии

Очень часто при написании сценариев возникает потребность встроить в код ActionScript примечания для себя или коллег по работе. Для этого используются особые выражения языка ActionScript, так называемые *комментарии*. Комментарии не обрабатываются Flash и исключаются из сценариев при экспорте фильма, поэтому в них можно писать все что угодно.

Для вставки комментариев в код ActionScript предусмотрены два оператора: `//` и `/*...*/`.

```
// Строка комментария
```

Этот оператор позволяет вставить в конец выражения однострочный комментарий.

```
a = b + c; // Это однострочный комментарий
```

Заметьте, что комментарий находится после точки с запятой, обозначающей конец выражения.

```
/*  
Комментарий  
*/
```

А этот оператор позволяет вставить в код программы комментарий любого размера.

```
/*  
В этом выражении мы складываем содержимое двух переменных  
и помещаем результат в третью  
*/  
a = b + c;
```

Сложные выражения

Сложные выражения называются так потому, что состоят из нескольких простых выражений. Как правило, сложные выражения выполняются особым образом и служат для особых целей.

Блоки

ActionScript позволяет вам объединить несколько выражений в одно, называемое *блоком выражений*. Для этого нужный участок кода заключается в фигурные скобки { и }.

```
a = 11;  
{  
b = "12";  
c = a - b;  
}
```

Как правило, блоки не используются сами по себе. Чаще всего они входят в состав других сложных выражений.

Условные выражения

Условное выражение позволяет вам, в зависимости от выполнения или невыполнения какого-либо условия, выполнить один из двух фрагментов кода. Существует также другой, вырожденный вариант условного выражения, выполняющий фрагмент кода при выполнении условия и не выполняющий его, если условие не выполнено.

Что может быть таким условием? Значение логической переменной или результат вычисления логического выражения. Здесь надо сказать, что логические выражения в большинстве случаев используются как раз в условных выражениях для проверки условия. А фрагменты кода, которые нужно вы-

полнить (или не выполнить), оформляются как блоки либо, если они достаточно просты, как одиночные выражения.

Условное выражение имеет следующий формат:

```
if (<Условие>) . . .
```

```
    Блок "то"
```

```
else
```

```
    Блок "иначе"
```

Вырожденный формат его выглядит так:

```
if (<Условие>) . . .
```

```
    Блок "то"
```

т. е. без блока `else`.

Для написания условных выражений используются особые действия `if` и `else` (выделены полужирным шрифтом). Эти действия применяются только в условных выражениях.

Условие — это и есть логическое выражение, согласно которому Flash принимает решение, какой блок кода выполнить. Если условие имеет значение `true` ("истина"), то выполняется блок "то". Если же условие имеет значение `false` ("ложь"), то выполняется блок "иначе", если он есть. Если же блок "иначе" отсутствует, выполняется следующее выражение программы.

Рассмотрим несколько примеров.

```
if (x == 1) {  
    a = "Единица";  
    b = 1;  
} else {  
    a = "Не единица";  
    b = 22222;  
}
```

Здесь мы сравниваем значение переменной `x` с единицей и в зависимости от результатов сравнения присваиваем переменным `f` и `h` разные значения. Обратите внимание на условие — именно так записывается оператор сравнения.

Условие может быть довольно сложным.

```
if (x == 1 && y > 10)  
    f = 3;  
else  
    f = 33;
```

Здесь мы использовали сложное условие, возвращающее `true` в случае, если значение переменной `x` равно единице. И значение переменной `y` больше

десяти. Заметьте также, что мы подставили одиночные выражения, т. к. фрагменты кода слишком просты, чтобы оформлять их в виде блоков.

Если ваше условное выражение совсем простое, вы можете записать его немного по-другому. А именно, воспользоваться *условным оператором* ? :

```
<Условие> ? <Выражение "то"> : <Выражение "иначе">;
```

Достоинство этого оператора в том, что он может быть частью выражения. Смотрите сами:

```
f = (x == 1 && y > 10) ? 3 : 33;
```

Фактически мы записали предыдущий пример, но в виде обычного арифметического выражения. Компактность кода налицо. Недостаток же оператора ? в том, что с его помощью можно записать только самые простые условные выражения.

Выражения выбора

Выражение выбора — это несколько условных выражений, объединенных в одно. Его формат таков:

```
switch (<Выражение>) {  
    case <Значение 1> :  
        <Блок 1>  
        [break;]  
    [case <Значение 2> :  
        <Блок 2>  
        [break;]]  
    ... Другие секции case  
    [default :  
        <Блок, исполняемый для остальных значений>]  
}
```

В выражениях выбора используются действия `switch`, `case` и `default` (выделены полужирным шрифтом). Эти действия используются только в условных выражениях.

В результате вычисления выражения получается результат, который последовательно сравнивается со значением 1, значением 2 и т. д. Если такое сравнение увенчалось успехом, выполняется соответствующий блок кода (блок 1, блок 2 и т. п.). Если же ни одно сравнение не увенчалось успехом, выполняется блок кода, находящийся в секции `default`, если, конечно, она есть.

Пример использования выражения выбора:

```
switch (a) {  
    case 1 :  
        ...
```

```
    out = "Единица";  
    break;  
case 2 :  
    out = "Двойка";  
    break;  
case 3 :  
    out = "Тройка";  
    break;  
default :  
    out = "Другое число";  
}
```

Если одна из секций не имеет действия `break`, то выполнение кода не прервется на этой секции, а продолжится дальше. Так, если результат выражения совпал с условием 1, и была выполнена секция 1, не содержащая действия `break`, будут выполнены также секция 2, секция 3 и все последующие вплоть до секции `default`.

И если мы уберем все действия `break` в своем примере:

```
switch (a) {  
    case 1 :  
        out = "Единица";  
    case 2 :  
        out = "Двойка";  
    case 3 :  
        out = "Тройка";  
    default :  
        out = "Другое число";  
}
```

то в любом случае получим в переменной `out` строку "Другое число".

Циклы

Циклы — это особые выражения, позволяющие выполнить один и тот же блок кода несколько раз. Выполнение кода прерывается по наступлению некоего условия. (Есть, правда, бесконечные циклы, но необходимость в них возникает довольно редко.) Здесь напрашивается аналогия с условными операторами, когда при наступлении некоего условия исполняется или не исполняется какой-либо блок кода.

ActionScript предлагает программистам несколько разновидностей циклов. Ниже мы опишем их все.

Цикл со счетчиком

Цикл со счетчиком используется, если какой-то код нужно выполнить определенное число раз. Это наиболее часто используемый вид циклов.

Для подсчета, сколько раз был выполнен этот фрагмент кода, используется переменная, называемая *счетчиком цикла*. Перед каждым выполнением блока кода выполняется проверка, достигло ли значение счетчика предельного значения, и, если не достигло, выполняется очередное повторение. А блок кода, который выполняется в цикле, называется *телом цикла*.

for (<Выражение инициализации>; <Условие>; <Приращение счетчика>)

Тело цикла

Для задания цикла со счетчиком используется действие **for**. Поэтому такие циклы часто называют "циклами **for**".

Выражение инициализации присваивает счетчику начальное значение. Далее проверяется условие цикла, и, если его значение истинно, тело цикла выполняется. После этого выполняется приращение счетчика, изменяющее значение счетчика, затем снова проверяется условие, и т. д. пока условие не станет ложным (*false*), т. е. пока счетчик не дойдет до предельного значения.

Пример цикла со счетчиком:

```
for (i = 1; i < 11; i++) {  
    a += 3;  
    b = i * 2 + 1;  
}
```

Этот цикл будет выполнен 10 раз. Мы присваиваем счетчику *i* начальное значение 1 и после каждого выполнения тела цикла увеличиваем его на единицу. Цикл перестанет выполняться, когда значение счетчика увеличится до 11, и условие станет ложным.

Заметьте также, что мы использовали счетчик цикла в одном из выражений тела цикла — это допустимо. Счетчик *i* будет содержать последовательно возрастающие значения от 1 до 10, которые можно использовать в вычислениях. (И используется, как вы видите.)

Еще два примера цикла со счетчиком:

```
for (i = 10; i > 0; i--) {  
    a += 3;  
    b = i * 2 + 1;  
}
```

Здесь значение счетчика декрементируется. Начальное его значение равно 10. Цикл выполнится 10 раз и завершится, когда счетчик будет содержать 0, при этом значения последнего будут последовательно уменьшаться от 10 до 1.

```
for (i = 2; i < 21; i += 2) b = i * 2 + 1;
```

А в этом примере начальное значение счетчика равно 2, а конечное — 21, цикл выполнится опять же 10 раз. А все потому, что значение счетчика увеличивается на 2 и последовательно принимает значения 2, 4, 6 ... 20.

В особом вырожденном случае цикл `for` может даже не содержать тела. В этом случае "полезную нагрузку" цикла несет на себе выражение приращения.

Цикл с постусловием

Цикл с постусловием во многом похож на цикл со счетчиком, а именно, тем, что он выполняется до тех пор, пока остается истинным условие цикла. Причем условие проверяется не до, а после выполнения тела цикла, отчего цикл с постусловием и получил свое название. Поэтому такой цикл выполнится хотя бы один раз, даже если условие с самого начала ложно.

Форма цикла с постусловием:

```
do
    Тело цикла
while (<Условие>);
```

Для задания цикла с постусловием используются действия `do` и `while`. Поэтому такие циклы часто называют "циклами `do-while`".

Вы можете использовать цикл с постусловием различными способами.

```
do {
    a = a * i + 2;
    i = ++i;
} while (a < 100);
```

В рассмотренном выше примере проверяется наступление некоего отвлеченного условия.

```
var a = 0, i = 1;
do {
    a = a * i + 2;
    i = ++i;
} while (i < 20);
```

А здесь мы уже используем счетчик, чье конечное значение ограничено. Конечно, в таких случаях удобнее будет применять уже знакомый вам и специально предназначенный для таких случаев цикл со счетчиком.

Цикл с предусловием

Цикл с предусловием отличается от цикла с постусловием тем, что условие проверяется перед выполнением тела цикла. Так что, если оно (условие) изначально ложно, цикл не выполнится ни разу.

while (<Условие>)

Тело цикла

Для задания цикла с постусловием используется действие `while`. Поэтому такие циклы называют еще "циклами `while`" (не путать с "циклами `do-while`").

Пример цикла с предусловием:

```
while (a < 100) {  
    a = a * i + 2;  
    i = ++i;  
}
```

Прерывание цикла

Иногда бывает нужно прервать выполнение цикла. Для этого JavaScript предоставляет программистам действия `break` и `continue`.

Действие `break` позволяет *прервать* выполнение цикла и перейти к следующему за ним выражению.

```
while (a < 100) {  
    a = a * i + 2;  
    if (a > 50) break;  
    i = ++i;  
}
```

В этом примере мы прерываем выполнение цикла, если значение переменной `a` превысит 50.

Собственно, мы уже знакомы с действием `break`. Оно использовалось также в выражении выбора, где вело себя точно так же.

Действие `continue` позволяет *перезапустить* цикл, т. е. оставить невыполненными все последующие команды, входящие в тело цикла, и вернуться к тому месту, где проверяется условие цикла.

```
while (a < 100) {  
    i = ++i;  
    if (i > 9 && i < 11) continue;  
    a = a * i + 2;  
}
```

Здесь мы пропускаем выражение вычисления `a` для всех значений `i` от 10 до 20.

Обработчики событий

Мы уже говорили, что некоторые сценарии можно "привязать" к различным событиям, происходящим в системе. Такими событиями могут быть, напри-

мер, щелчок по кнопке или окончание загрузки фильма. И, стало быть, сценарии, привязанные к этим событиям, будут исполняться после щелчка по кнопке и полной загрузки фильма.

Сценарий, привязанный к событию, называется *обработчиком* этого события. Такие обработчики могут быть привязаны только к двум типам элементов фильма: вложенным клипам и кнопкам. И, в зависимости от элемента, к которому привязывается обработчик, для их создания используются два разных действия.

Для привязки обработчика к вложенному клипу используется уже знакомое вам действие `onClipEvent`:

```
onClipEvent(<Событие>) {  
    Код обработчика  
}
```

Например:

```
onClipEvent(mouseDown) {  
    stop();  
}
```

Знакомый сценарий, вам не кажется? Он обрабатывает щелчок мыши по вложенному клипу и в ответ на него останавливает проигрывание этого клипа.

Для привязки обработчика к кнопке вы должны использовать немного другой синтаксис, с использованием действия `on`:

```
on(<Событие>) {  
    Код обработчика  
}
```

Пример обработчика события, возникающего при нажатии кнопки:

```
on(press) {  
    stopAllSounds();  
}
```

Этот сценарий отключает все звуковое сопровождение фильма.

Полное описание действий `onClipEvent` и `on` приведено в *приложении 1*.

Функции

Функция — это особым образом написанный и оформленный фрагмент кода `ActionScript`, который можно вызвать из любого места любого сценария. Фактически функция — это повторно используемый участок кода, который, будучи написанным один раз, может быть вызван где угодно, кем угодно и

сколько угодно раз. Так что, если какой-то фрагмент кода встречается в нескольких местах вашего сценария или сценариев, лучше всего преобразовать его в функцию.

Собственно код, ради которого и была создана функция, называется *телом функции*. Каждая функция, кроме того, должна иметь уникальное имя, по которому к ней можно будет обратиться. Функция также может принимать один или несколько *аргументов* или *параметров* и возвращать *результат*, который можно использовать в программе.

Создание функций

Прежде, чем функция будет использована где-то в сценарии, ее нужно объявить. Объявление функции выполняется с помощью действия `function`.

```
function <Имя>([<Список аргументов, разделенных запятыми>])
```

Тело функции

Имя функции, как уже говорилось, должно быть уникально в пределах области видимости. Для имен функций действуют те же правила, что и для имен переменных.

Список аргументов представляет собой набор переменных, в которые при вызове функций будут помещены нужные аргументы. Вы можете придумать для этих переменных любые имена — все равно они будут использованы только внутри тела функции. Это так называемые *формальные аргументы* функции.

Список аргументов функции помещается в круглые скобки, а аргументы отделяются друг от друга запятыми. Если же функция не требует аргументов, то скобки при этом все равно должны оставаться.

В пределах тела функции, как вы уже поняли, над аргументами выполняются некоторые действия и, возможно, вырабатывается результат. Чтобы вернуть его из функции в место вызова, используется действие `return`:

```
return {Переменная или выражение};
```

Здесь переменная должна содержать возвращаемое значение, а выражение должно его вычислять.

Пример объявления функции:

```
function divide(a, b) {  
    var c;  
    c = a / b;  
    return c;  
}
```

Эта функция принимает два аргумента — `a` и `b` — после чего делит `a` на `b` и возвращает частное от деления. Как видите, она использует для хранения промежуточного результата локальную переменную `c`.

Собственно, эту функцию можно записать в одну строку:

```
function divide(a, b) { return a / b; }
```

Так намного короче и нагляднее.

Объявленные функции видны только в пределах текущего клипа (*функция уровня клипа*). Если вы хотите сделать функцию доступной во всем фильме (*глобальной*), используйте уже знакомый вам модификатор `_global`:

```
function _global.glDivide(a, b) { return a / b; }
```

Проверьте только, чтобы глобальная функция не имела "тезок" внутри одного клипа.

Вызов функций

После того, как вы объявили функцию, вы можете *вызвать* ее из любого места кода в пределах области видимости. Для этого используется следующий формат:

```
<Имя функции>([<Список фактических аргументов, разделенных запятыми>])
```

Здесь указывается *имя* нужной функции, и в круглых скобках перечисляются **ФАКТИЧЕСКИЕ** аргументы, над которыми нужно выполнить соответствующие действия. Запомните: именно фактические аргументы, а не формальные, использованные в объявлении функции. Функция вернет результат, который можно присвоить переменной или использовать в сложном выражении.

Пример вызова объявленной нами ранее функции:

```
d = divide(3, c);  
s = 4 * divide(x, r) + y;
```

Если функция не возвращает результата, то она вызывается немного по-другому:

```
initVars(1, 2, 3, 6);
```

Как видите, если функция не возвращает результата, его можно и не хранить.

Если при написании сценариев вы работаете в обычном режиме панели **Actions**, то вам следует воспользоваться пунктом **Call Function** списка доступных операций. Этот пункт находится в подветви **User-Defined Functions** ветви **Actions**.

Функции можно вызывать друг из друга:

```
function samplefunc1(a, b) {  
    . . .  
}  
  
function samplefunc2(c) {
```

```
. . .  
k = y + samplefunc1(x, 2);  
}
```

Имейте только в виду, что функция должна быть объявлена прежде, чем будет использована.

Когда вы вызываете функцию просто указанием ее имени, то будет вызвана функция уровня клипа. Если же такой функции не будет найдено, Flash вызовет глобальную функцию с таким именем. Если же вы хотите вызвать функцию, находящуюся в другом клипе, укажите имя этого клипа перед именем функции, разделив их точкой:

```
someClip.initVars(1, 2, 3, 6);
```

А теперь небольшой фокус:

```
var varFunc;  
function sampleFunc(a, b) {  
    . . .  
}
```

```
varFunc = sampleFunc;
```

Да, ActionScript допускает присвоение функции переменной. Такая переменная будет иметь тип функции, и оператор `typeof` вернет для нее строку "function". То же самое значение оператор `typeof` вернет, если ему передать само имя функции.

Рекурсия

Здесь мы рассмотрим еще один вопрос, связанный с написанием и вызовом функций.

Вы уже знаете, что функции могут вызывать другие функции, конечно, если те уже определены. Но функции могут также вызывать и сами себя. Такой прием программирования иногда бывает очень полезен и называется *рекурсией* или рекурсивным вызовом.

В обычных условиях, если функция вызовет сама себя, то она войдет в бесконечный цикл вызовов, из которого нет выхода (так называемая *бесконечная рекурсия*). В конце концов, это завершится аварийной остановкой Flash, а то и крахом всей операционной системы. Поэтому функция, предназначенная для рекурсивного вызова, должна предусматривать возможность выхода из этого цикла вызовов. Стандартного способа сделать это не существует, поэтому решение для каждого конкретного случая нужно искать особо.

Рассмотрим пример функции, написанной специально для применения в рекурсивных вызовах:

```
function factorial(a) {
```

```
if (a == 0) {  
    return 1;  
}  
else  
    return (a * factorial(a - 1));  
}
```

Эта функция вычисляет факториал числа *a*, переданного в качестве параметра. Она рекурсивно вызывает сама себя для того, чтобы получить факториал числа *a*-1. Заметьте также, что в теле функции выполняется проверка условия равенства *a* нулю; если это условие истинно, возвращается единица, после чего дальнейших рекурсивных вызовов не производится, а все уже сделанные — "хором" завершаются. Таким образом, данная функция имеет защиту от бесконечной рекурсии, ведь когда-нибудь она все равно получит в качестве аргумента ноль.

Еще один пример функции.

```
function endless(a, b) {  
    var c;  
    c = endless(a, b);  
}
```

Эта функция будет вызывать себя бесконечно и, в конце концов, "обрушит" Flash. А все потому, что в ней нет условия, которое прерывало бы бесконечный цикл вызовов.

Встроенные функции ActionScript

Язык ActionScript предоставляет вам несколько десятков функций, которые уже реализованы в нем. Такие функции называются *встроенными*.

К встроенным, в частности, относятся функции преобразования типов:

- Boolean — преобразование в логическую величину;
- Number — преобразование в число;
- String — преобразование в строку.

Эти функции принимают в качестве параметра "спорное" выражение, значение которого нужно привести к определенному типу.

Полное описание всех встроенных функций ActionScript приведено в *приложении 1*.

Массивы

Массив — это пронумерованный набор переменных одного типа, называемых *элементами* массива. Доступ к нужному элементу массива выполняется

по его порядковому номеру, называемому *индексом*. А общее число элементов массива называется его *размером*.

Массивы идеально подходят в тех случаях, когда нужно хранить в одной переменной упорядоченный набор данных. Ведь массив фактически представляет собой одну переменную.

Чтобы создать массив, нужно просто присвоить любой переменной список его элементов, разделенных запятыми и заключенных в квадратные скобки:

```
var someArray;  
someArray = [1, 2, 3, 4];
```

Здесь мы создали массив, содержащий четыре элемента, и присвоили его переменной `someArray`. После этого мы можем получить доступ к любому из элементов по его индексу, указав его после имени переменной массива в квадратных скобках. Только запомните, что нумерация элементов массива начинается с нуля.

```
a = massive[2];
```

В этом примере мы получили доступ к третьему элементу массива.

Не обязательно определять сразу все элементы массива:

```
someArray2 = [1, 2, , 4];
```

Обратите внимание на то, что мы пропустили третий элемент, и он остался неопределенным. Оператор `typeof` вернет для него значение `"undefined"`.

Если будет нужно, мы легко сможем добавить к массиву еще один элемент, просто присвоив ему требуемое значение. Вот так:

```
someArray[4] = 9;
```

При этом будет создан новый элемент массива с индексом 4 и значением 9.

Можно даже сделать так:

```
someArray[7] = 9;
```

В этом случае будут созданы четыре новых элемента, и восьмой элемент получит значение 9. Пятый, шестой и седьмой останутся неопределенными (`undefined`).

Вы можете присвоить любому элементу массива другой массив (или, как говорят опытные программисты, создать так называемый *вложенный массив*).

```
someArray2[2] = ["n1", "n2", "n3"];
```

После этого можно получить доступ к любому элементу вложенного массива, указав оба индекса:

```
str = someArray2[2][1];
```

Переменная `str` будет содержать строку `"n2"`.

Оператор `typeof` возвращает для массива значение `object`. Это значит, что массив имеет объектный тип.

Объекты

Итак, мы познакомились с типами данных, переменными, константами, операторами, действиями, простыми и сложными выражениями, функциями и массивами. Теперь настала пора узнать о самых сложных структурах данных ActionScript — объектах.

Используя различные объекты, как встроенные в язык ActionScript, так и созданные вами, вы сможете полностью управлять своим фильмом. Более того, вы сможете писать настоящие программы в среде Flash, так называемые Flash-приложения. Вся мощь Flash будет доступна вам через объекты. И пользоваться ей будет не так уж и сложно.

Понятия объекта и экземпляра

Объект — это сложный тип данных, включающий в себя множество переменных — *свойств* — и набор функций для манипуляции этими переменными — *методов*. Свойства хранят данные, а методы их обрабатывают. Каждый объект имеет уникальное имя, по которому к нему можно обратиться.

Объект — это всего лишь тип данных, некое абстрактное описание, включающее в себя набор свойств и методов. От него порождаются конкретные представители — *экземпляры* объекта. А уж они содержат реальные данные, которыми вы можете управлять. Однако очень часто, говоря "объект", фактически имеют в виду экземпляр объекта: объект клипа, объект кнопки. Только если нужно однозначно отличить объект от экземпляра объекта, говорят "экземпляр".

В качестве объекта очень легко представляются различные сущности. Возьмем, например, уже знакомый нам объект `вложенныйКлип`. Он имеет некоторый набор свойств (размеры, местоположение, длина анимационной последовательности в кадрах) и методов (запустить и остановить проигрывание, переместить указатель на другой кадр). Пользуясь этими свойствами и методами, можно управлять объектом `вложенныйКлип` из сценария:

```
клип1.ширина = 200;
```

```
клип1.перейтиНаКадр(3);
```

Как видите, чтобы обратиться к свойству или методу экземпляра объекта, нужно поместить перед именем этого свойства или метода имя объекта и разделить их точкой. Таким образом можно точно указать, к какому свойству или методу какого экземпляра вам нужно обратиться.

Работа с объектами

Перед тем, как начать работу с экземпляром какого-либо объекта, его нужно создать. (Это самый общий случай. Существуют, правда, экземпляры объек-

тов, которые не нуждаются в создании, т. к. создаются самим Flash, но о них мы поговорим потом.) Создание экземпляра нужного объекта выполняется с помощью действия `new`:

```
new <Имя объекта>([<Список параметров, разделенных запятыми>])
```

Результат, возвращаемый действием `new`, может быть помещен в переменную или использован в выражении. Это так называемая *ссылка* на созданный экземпляр объекта. Ссылка указывает на место в памяти компьютера, где хранятся все данные экземпляра.

```
var obj;  
obj = new someObject(a, b);
```

В результате выполнения этого примера в переменную `obj` будет помещена ссылка на созданный экземпляр объекта `someObject`, или, как чаще говорят, был создан экземпляр `obj` объекта `someObject`. Заметьте, что при этом экземпляру были переданы два параметра — `a` и `b` — что, надо сказать, встречается нечасто.

Теперь вы можете обращаться к свойствам и методам созданного экземпляра `obj`, используя уже знакомый вам способ доступа:

```
obj.prop1 = 0;  
a = obj.prop2 + 2;  
obj.method1();
```

Если вы для написания сценариев пользуетесь обычным режимом панели **Actions**, то для вызова метода объекта вам следует выбрать пункт **Call Function** списка доступных операций. Он находится в подветви **User-Defined Functions** ветви **Actions**.

Иногда бывает, что один объект содержит внутри себя другие объекты (так называемые *внутренние* или *вложенные*). Чтобы обратиться к их свойствам и методам, используется похожий синтаксис:

```
outerObject.innerObject.prop = 0;
```

где `outerObject` — экземпляр объекта, имеющего в своем составе другой, внутренний, объект `innerObject`. Как видите, все организовано достаточно логично.

Выражение, реализующее доступ к внутреннему объекту, также называется *путем*. Иногда путь бывает очень длинным:

```
objectOuter.objectInner.objectInnerInner.prop = 10;
```

Разумеется, вы можете присваивать один экземпляр другому:

```
obj2 = obj;
```

Имейте, однако, в виду, что теперь обе переменные — и `obj2`, и `obj` — будут указывать на один и тот же экземпляр. Дело в том, что вы скопировали не сам экземпляр, а указывающую на него ссылку. Так что теперь у вас будут

две ссылки, указывающие на один и тот же экземпляр. (Кстати, точно так же ведут себя и массивы.) Если же вы хотите создать действительно два разных экземпляра, вам придется прибегнуть к действию `new`.

Чтобы удалить ненужный экземпляр, используйте действие `delete`:

```
delete <Имя экземпляра>;
```

Например,

```
delete obj, obj2;
```

Всегда удаляйте не нужные более экземпляры. Помните, что они зря занимают память компьютера и могут вызвать появление трудноуловимых ошибок.

А теперь давайте еще раз взглянем на выражение создания нового экземпляра:

```
obj = new SomeObject(a, b);
```

Оно удивительно похоже на вызов функции, не так ли? На самом деле, это и есть особая функция, называемая *конструктором*. От других методов он отличается тем, что вызывается при создании экземпляра объекта и устанавливает начальные значения его свойств. Имя конструктора всегда совпадает с именем класса объекта.

Но существует один объект, не имеющий конструктора. Это объект `Object` — самый простой из объектов `ActionScript`. Его экземпляры создаются с помощью так называемых *инициализаторов*:

```
new <Имя объекта>([<Список параметров и их значений, разделенных  
⌘запятыми>])
```

Пример использования инициализатора:

```
obj2 = {prop1: 1, prop2: 2, prop3: a + b};
```

Как видите, здесь просто перечисляются свойства объекта и их значения.

Оператор `typeof` возвращает для объекта значение `object`.

Несколько новых операторов и действий

А теперь самое время рассмотреть несколько новых операторов и действий, применяемых при работе с объектами и их экземплярами.

Оператор `instanceof` проверяет, является ли Экземпляр экземпляром заданного Объекта, и возвращает соответственно `true` или `false`.

```
<Экземпляр> instanceof <Объект>
```

Здесь экземпляр задается в виде переменной, а объект — в виде имени объекта.

```
if (obj instanceof someObject) . . .
```

Действия `for` и `in` служат для просмотра всех свойств заданного экземпляра и выполняют для каждого какие-либо действия. Для этого организуется особый цикл — *цикл просмотра*, называемый также "циклом `for-in`":

```
for (<Переменная-ссылка на свойство> in <Экземпляр>)
```

Тело цикла

В Переменной-ссылке на свойство каждый раз оказывается значение очередного свойства Экземпляра. Вы можете использовать эту переменную для получения доступа к найденному свойству.

```
for (k in obj) {  
    k = ' ' + k + ' ';  
}
```

Этот фрагмент сценария просматривает все свойства экземпляра `obj` и добавляет к значению каждого из них пробелы слева и справа. (Предполагается, что значения всех свойств этого экземпляра имеют строковый тип.)

Действие `with` позволяет значительно сократить длину выражений JavaScript, если в них используются свойства или методы какого-либо экземпляра (заметьте: одного экземпляра) объекта. Мы не будем приводить его формат, а сразу покажем пример использования — и вы все поймете:

```
someObject.prop1 = 1;  
someObject.prop2 = 2;  
someObject.prop3 = 3;  
someObject.method1;
```

В этом фрагменте сценария не использовалось действие `with`. Видите, какие длинные строки выражений. Теперь используем действие `with`:

```
with (someObject){  
    prop1 = 1;  
    prop2 = 2;  
    prop3 = 3;  
    method1;  
}
```

Фрагмент сценария сразу стал компактнее. И быстрее, кстати говоря.

Встроенные объекты ActionScript

Встроенными называются объекты, реализованные в самом языке ActionScript. Ниже будут рассмотрены их краткие описания и примеры использования. Полное описание всех этих объектов приведено в *приложении 1*.

Интересной особенностью языка ActionScript является то, что он может представлять обычные типы данных — строковый, числовой, логический — как объекты. Поэтому вы можете обращаться со строками, числами и логическими величинами как с экземплярами соответствующих объектов, вызывать их методы и использовать их свойства. Львиная доля встроенных объектов ActionScript как раз и "отвечает" за объектное представление обычных типов данных.

String

Объект `String` представляет собой обычную строку. Пользуясь его свойствами и методами, вы можете выполнять над строками различные манипуляции.

Создать экземпляр объекта `String` очень просто:

```
s = new String("Flash");  
s = "Flash";
```

Оба этих выражения взаимозаменяемы. Но, вероятно, второе вам все-таки привычнее.

Свойство `length` позволит вам получить длину строки в символах.

```
l = s.length;
```

Метод `charAt` вернет вам символ строки с заданным номером, переданным в качестве единственного параметра. Учтите только, что нумерация символов в строке начинается с нуля, поэтому первый символ будет иметь номер 0, а последний — `s.length-1`.

```
ch = s.charAt(s.length - 1);
```

Метод `charCodeAt` вернет вам код символа строки с заданным номером, переданным в качестве единственного параметра, в формате Unicode.

```
charCode = s.charCodeAt(2);
```

Метод `indexOf` вернет вам номер вхождения подстроки в текущую строку. Если подстрока не найдена, возвращается -1. При этом вы можете указать номер вхождения, с которого начнется поиск:

```
<Строка>.indexOf(<Подстрока>, [<Номер вхождения>])
```

Например:

```
s = "Macromedia Flash MX";  
n = s.indexOf("a", 2);
```

Метод `lastIndexOf` схож с методом `indexOf`, только ищет не слева направо, а справа налево, т. е. с конца строки.

Методы `toLowerCase` и `toUpperCase` преобразуют все символы строки соответственно к нижнему и верхнему регистру.

Number

Объект `Number` представляет собой обычное число. Пользуясь его свойствами и методами, вы можете выполнять над числовыми данными различные манипуляции.

Создается экземпляр объекта `Number` так:

```
n = new Number(232);
```

```
n = 232;
```

Оба этих выражения взаимозаменяемы.

Метод `toString` возвращает строковое представление числа.

```
s = n.toString();
```

Также объект `Number` имеет ряд свойств, возвращающих различные "специальные" значения.

Boolean

Объект `Boolean` представляет собой обычную логическую величину. Пользуясь его свойствами и методами, вы можете выполнять над логическими данными различные манипуляции.

Создается экземпляр объекта `Boolean` так:

```
b = new Boolean(true);
```

```
b = true;
```

Оба этих выражения взаимозаменяемы.

Метод `toString` возвращает строковое представление логической величины — `"true"` и `"false"`.

```
s = b.toString();
```

Date

Объект `Date` представляет собой числовую величину, представляющую значение даты и времени. Дата и время при этом "упаковываются" особым образом и преобразуются в число, которое `ActionScript` при необходимости может обратно "распаковать" и представить как дату.

Экземпляр объекта `Date` создается так:

```
d = new Date([{Год}, {Месяц}, [{Число}[, {Часы}[, {Минуты}[, {Секунды}
✂ [, {Миллисекунды}]]]]]]);
```

Краткое описание всех параметров конструктора этого объекта:

- ❑ Год может быть задан двумя или четырьмя цифрами. С четырьмя цифрами все просто; если же год задан двумя цифрами, то значение 0 соответствует 1900 году, а 99 — 1999 году;
- ❑ Месяц задается значением от 0 (январь) до 11 (декабрь);

- ❑ Дата задается значением от 1 до 31;
- ❑ Минуты и Секунды задаются значением от 0 до 59;
- ❑ Миллисекунды задаются значением от 0 до 999.

Если же ни один из параметров не указан, в экземпляре объекта `Date` заносится текущее значение даты.

Объект `Date` имеет огромное количество методов, возвращающих или задающих различные "части" значения даты. Так, метод `getMonth` возвращает текущее значение месяца, а метод `setMonth` позволяет задать месяц, не меняя других "частей" даты. А уже знакомый вам метод `toString` возвращает строковое представление даты, используя региональные установки вашего компьютера.

Array

Объект `Array` представляет собой массив. Пользуясь его свойствами и методами, вы можете выполнять над массивами различные манипуляции.

Создать массив можно как обычным способом, присвоив нужной переменной список элементов, разделенных запятыми и заключенных в квадратные скобки, так и воспользовавшись конструктором. Ниже перечислены три способа вызова конструктора массива:

```
arr = new Array();  
arr = new Array(<Размер>);  
arr = new Array(<Список элементов, разделенных запятыми>);
```

Если вы не передадите конструктору никакого параметра, созданный массив будет иметь нулевой размер. Если же вы передадите в качестве параметра одно число, то `ActionScript` посчитает его размером массива и создаст массив, все элементы которого будут иметь тип `undefined`.

Свойство `length` позволит вам узнать размер массива.

```
l = arr.length;
```

Метод `reverse` изменяет порядок следования элементов массива на противоположный. А хорошо знакомый вам метод `toString` возвращает строку, содержащую значения всех элементов массива, разделенные запятыми.

Function

Объект `Function` представляет собой функцию. Пользуясь его свойствами и методами, вы можете выполнять над функцией различные манипуляции.

Конструктор объекта-функции имеет следующий формат:

```
{Имя функции} = new Function({Список аргументов, заключенных в кавычки и  
разделенных запятыми} {Тело функции, заключенное в кавычки});
```

Это значит, что вы можете создавать функции как уже знакомым вам путем, с помощью действия `function`, так и тем же образом, каким создаются эк-

земляры объектов. Заметьте, что во втором случае и список аргументов, и тело функции заключаются в кавычки.

Давайте рассмотрим пример объявления функции, сначала традиционным путем, потом — новым, в виде объекта:

```
function someFunc(a, b) {  
    return (a + b) / 2;  
}  
  
someFunc = new Function("a", "b", "(a + b) / 2");
```

И при этом вызывать созданную функцию вы можете по-старому, независимо от способа создания.

```
f = someFunc(s, 2) - 9;
```

Arguments

Объект `Arguments` представляет собой массив аргументов, переданных функции. Этот объект может быть очень полезен, если вашей функции передается переменное число аргументов.

Вам не нужно специально создавать экземпляр объекта `Arguments`. Flash создает его сам. Причем доступен этот экземпляр только внутри тела функции.

```
function AFunc() {  
    var i, s = '';  
    for (i = 0; i < arguments.length; i++) {  
        s += argument[i - 1];  
    }  
    return s;  
}
```

Вышеприведенная функция объединяет все переданные ей аргументы в одну строку и возвращает ее вызвавшему выражению. Как вы поняли, свойство `length` возвращает размер массива аргументов, т. е. количество аргументов, переданных функции.

Math

Объект `Math` служит для предоставления доступа к встроенным константам и встроенным математическим и тригонометрическим функциям языка ActionScript. Единственный экземпляр этого объекта создается самим Flash.

Методы `sin`, `cos` и `tan` позволяют вычислить соответственно синус, косинус и тангенс угла, заданного в радианах. Метод `sqrt` вычисляет квадратный корень. Метод `pow(x, y)` возводит x в степень y .

Кроме того, объект `Math` содержит несколько свойств, возвращающих значение различных математических констант. Так, свойство `PI` возвращает

значение числа π . А свойство `E` возвращает значение основания натурального логарифма.

Object

Объект `Object` — простейший из объектов, предоставляемых языком `ActionScript`. Он содержит минимальный набор свойств и методов и служит для создания на его основе других объектов. (Подробнее о создании пользовательских объектов см. ниже.)

Пользовательские объекты

Пользовательскими называются объекты, созданные самим разработчиком сценариев для своих нужд. Ниже будут рассмотрены создание и использование таких объектов.

Создание экземпляров объекта *Object*

Если вам нужна сложная структура для хранения данных, проще всего будет создать экземпляр объекта `Object` и установить в нем нужные свойства и методы. Так как объект `Object` не имеет конструктора, для создания экземпляра следует просто использовать инициализатор:

```
var somePoint;  
somePoint = {x: 100, y: 50, color: "black"};
```

Вот и все. Таким образом, мы создали экземпляр объекта, представляющий некую точку, и в нем — свойства декартовых координат и цвета. Теперь мы можем его применить:

```
somePoint.y = somePoint.x / 2 + 20;
```

Можно даже добавить в этот объект новое свойство, просто присвоив ему требуемое значение:

```
somePoint.radius = 0.5;  
и потом использовать его:  
c = 2 * somePoint.radius;
```

Однако мы фактически не создали нового объекта. Мы просто расширили один-единственный экземпляр объекта `Object`. И, если мы создадим еще один экземпляр объекта `Object`, он не будет содержать созданные нами свойства и методы.

Создание нового объекта

Если вы создадите новый объект, все добавленные вами свойства и методы появятся во всех его экземплярах. Согласитесь — это несравнимо удобнее, чем добавлять их в каждый вновь создаваемый экземпляр, хоть и отнимает

заметно больше времени. (Конечно, вы можете добавить дополнительные свойства и методы в экземпляры этого объекта, как это вы делали с экземплярами объекта `Object`.)

Чтобы создать новый объект, вам нужно просто создать его конструктор. Имя конструктора станет именем нового объекта.

```
function point(xx, yy, ccolor) {  
    this.x = xx;  
    this.y = yy;  
    this.color = ccolor;  
}
```

Выше мы привели объявление конструктора для объекта `point`, представляющего геометрическую точку. Этому конструктору передаются три аргумента: декартовы координаты `x`, `y` и цвет, а он присваивает эти аргументы свойствам создаваемого объекта.

Здесь надо сказать о модификаторе `this`. Этот модификатор предоставляет доступ к создаваемому объекту и доступен только в теле функции конструктора. Иначе говоря, если вы напишете выражение

```
x = xx;
```

это будет присвоением значения переменной `x`, а выражение

```
this.x = xx;
```

присвоит новое значение свойству `x` создаваемого этим конструктором объекта.

Создав конструктор нового объекта, вы можете создавать его экземпляры:

```
var somePoint;  
somePoint = new point(100, 200, "black");
```

Здесь вам уже все знакомо.

Создание методов и свойств

Выше мы рассмотрели создание свойств в экземплярах объекта `Object` и новых объектах, создаваемых с помощью конструкторов. Но как реализовать в них новые методы?

Очень просто. Если вы помните, функцию можно присвоить переменной:

```
someFunc = func;
```

после чего можно обращаться к этой функции через переменную:

```
a = someFunc(b);
```

А если присвоить функцию свойству объекта, получится новый метод:

```
function fMoveAt(dx, dy) {  
    this.x += dx;
```

```
    this.y += dy;
}
. . .
function point(xx, yy, ccolor) {
    this.x = xx;
    this.y = yy;
    this.color = ccolor;
    this.moveAt = fMoveAt;
}
```

Так мы создали новый метод `moveAt` объекта `point`. Этот метод сдвинет точку на определенные приращения по горизонтали и вертикали. Не забудьте только объявить функцию, которая станет методом, перед тем, как используете ее в объявлении конструктора.

Есть еще один способ создания методов. Функция точно так же присваивается свойству объекта, только присваивание это выполняется не в конструкторе, а вне его:

```
function point(xx, yy, ccolor) {
    this.x = xx;
    this.y = yy;
    this.color = ccolor;
}
point.prototype.moveAt = fMoveAt;
```

Здесь мы используем свойство `prototype`, чтобы обратиться к объявлению (*прототипу*) объекта. Мы создаем в нем свойство `moveAt` и присваиваем ему функцию `fMoveAt`, которая и станет нашим методом.

Пользуясь прототипом, вы также можете создавать новые свойства:

```
point.prototype.z = 0;
```

Но, на наш взгляд, все же удобнее делать это в конструкторе.

Создание динамических свойств

Кроме обычных, хорошо знакомых вам свойств, вы можете создавать так называемые *динамические* свойства. Они отличаются от обычных свойств тем, что при обращении к ним вызывается какая-либо функция, которую вы задали при создании этого свойства. Собственно, таких функций две: одна вызывается при попытке чтения значения свойства (*функция чтения*), а вторая — при попытке записи в него нового значения (*функция записи*).

Эти функции могут выполнять все, что вы хотите. Например, функция записи может модифицировать значения других свойств этого объекта. А функция чтения может вычислять значение этого свойства, т. к. самого этого свойства в классе фактически не существует. Таким образом, пере-

менной, в которой хранится значение свойства, в объекте нет — оно вычисляется всякий раз с помощью функции чтения.

Создание динамического свойства выполняется при помощи метода `addProperty` объекта `Object`. Так как все пользовательские объекты фактически происходят от объекта `Object`, этот метод доступен везде. Формат метода `addProperty` таков:

```
addProperty(<Имя свойства>, <Функция чтения>, <Функция записи>);
```

Пользуясь методом `addProperty`, вы можете создавать свойства, доступные только для чтения или только для записи. Чтобы создать свойство, доступное только для чтения, вам нужно вместо функции записи подставить `null`. А чтобы создать свойство, доступное только для записи, вам нужно подставить `null` вместо функции чтения.

Пример использования этого метода и создания динамического свойства приведен ниже. Создадим в объекте `point` свойство `square`, возвращающее площадь прямоугольника, образованного началом координат и точкой, заданной этим объектом. Причем свойство `square` не будет храниться в объекте, а будет вычисляться при каждой попытке его чтения.

Сначала создадим саму функцию чтения:

```
function fSquareCalc() {  
    return this.x * this.y;  
}
```

Теперь создадим динамическое свойство:

```
point.prototype.addProperty("square", fSquareCalc, null);
```

Вот и все. Теперь, если вы попытаетесь прочитать значение свойства `square`, ActionScript вызовет функцию `fSquareCalc`, которая вычислит площадь и вернет ее. Однако ничего записать в это свойство вы не сможете.

Наследование

Вы можете создавать объекты, которые наследуют свойства и методы других объектов. Таким образом, вы можете расширять функциональность уже существующих объектов, созданных вами или совсем другими разработчиками, причем для этого вам достаточно только написать код, реализующий эту дополнительную функциональность. Такой подход называется *наследованием* объектов. При этом объект, наследующий свойства и методы, называется *потомком* или *дочерним* объектом, а объект, свойства и методы которого наследуют, — *предком* или *родителем*.

Пусть, например, нужно "расширить" объект `point` так, чтобы он, наряду с координатами и цветом, хранил радиус точки. (Хоть евклидово определение точки и звучит как "нечто, не имеющее ни длины, ни ширины", таких точек

в реальности вы не найдете.) Для этого создадим новый объект `point2`, унаследованный от `point`, в который добавим свойство `radius`.

```
function point2(xx, yy, ccolor, rradius) {
    this.__proto__.x = xx;
    this.__proto__.y = yy;
    this.__proto__.color = ccolor;
    this.radius = rradius;
}
```

```
point2.prototype = new point();
```

Обратите внимание, что мы использовали свойство `__proto__`, чтобы добраться до свойств объекта-родителя. Если бы мы указали просто

```
this.x = xx;
this.y = yy;
this.color = ccolor;
```

то создали бы свойства `x`, `y` и `color` в объекте `point2`, что не входит в нашу задачу.

Самое последнее выражение собственно и задает предка объекта `point2`. Будьте внимательны, не забудьте его добавить.

Можно сделать немного по-другому:

```
function point2(xx, yy, ccolor, rradius) {
    super(xx, yy, ccolor);
    this.radius = rradius;
}
point2.prototype = new point();
```

Здесь оператор `super` вызывает конструктор класса-родителя.

Теперь вы можете использовать новый объект `point2`:

```
var somePoint, somePoint2;
somePoint = new point(100, 50, "black");
somePoint2 = new point2(200, 400, "blue", 0.5);
somePoint *= 2;
somePoint2.radius = 1;
```

Вы можете впоследствии создать новый объект `point3`, наследующий объект `point2`. Тогда объект `point2` станет потомком для объекта `point` и предком — для `point3`.

Все пользовательские объекты и большинство объектов встроенных наследуют свою функциональность от объекта `Object`. Поэтому все они имеют базовый набор свойств и методов этого, самого примитивного, объекта. Из

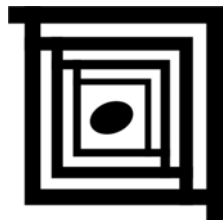
объектов, не наследующих `Object`, можно выделить объект `movieClip`, который будет описан в *главе 22*.

Внешние объекты

Внешние объекты находятся вне ActionScript. Это все объекты, предоставляемые пакетом Flash: клипы, кнопки и др. Внешними они называются потому, что не относятся к языку ActionScript, а находятся как бы вне его. Впрочем, это деление достаточно условное.

Использование внешних объектов будет описано в последующих главах *части 4* этой книги. А полное их описание со всеми свойствами и методами приведено в *приложении 1*.

Глава 22



Создание интерактивных фильмов

Вот мы и подошли к самому главному. Потихоньку, помаленьку, короткими перебежками от панели **Actions** к функциям и массивам, но мы добрались к цели. Долгим было наше путешествие, но даже самая долгая дорога рано или поздно подходит к концу.

Главы 20 и 21 были всего лишь вступлением. Основные вопросы будут рассматриваться в этой главе. Мы воспользуемся уже изученными средствами Flash и языка программирования **ActionScript** для создания интерактивных фильмов. Пока только фильмов — написание Flash-приложений будет описано далее в *главе 23*.

Здесь будут описаны все средства, с помощью которых можно управлять содержимым фильма из сценариев, а также все внешние объекты языка **ActionScript**, используемые для этого. Мы рассмотрим простейшие сценарии, выполняющие типичные задачи. Подобные сценарии вы сможете использовать в своих фильмах и приложениях Flash.

Одним словом, здесь вы научитесь программированию.

Объект *movieClip*

Подавляющее большинство интерактивных элементов в фильмах Flash создаются с помощью клипов, точнее, экземпляров образцов-клипов. (Подробнее о них см. *главу 10*.) Встроенные клипы могут управляться из сценариев, в отличие от графических экземпляров и обычной графики, которая не имеет соответствующих "рычагов управления". Кроме клипов, управляться из сценариев могут кнопки и, разумеется, сам фильм.

За доступ к клипу из сценария "отвечает" объект `movieClip`. Данный объект обеспечивает набор свойств и методов, которые и являются "рычагами управления" клипом. Все они будут подробно рассмотрены далее в этой главе.

В отличие от встроенных и пользовательских объектов, экземпляр объекта `movieClip` не нужно создавать с помощью конструктора. Это выполняет сам

Flash. Также он создает экземпляр объекта `movieClip`, "отвечающий" за *основной фильм* (то есть сам документ Flash). Вам остается только использовать нужные свойства и методы для управления соответствующим клипом (или фильмом).

Оператор `typeof` для клипа возвращает значение "clip".

Зачем нужны встроенные клипы

Встроенные клипы могут использоваться разными путями и для разных целей. Сейчас мы их перечислим.

Во-первых, встроенные клипы применяются для создания сложной трансформационной анимации. Как вы помните, Flash может создать только одно движение в слое. Если же вы хотите создать несколько движений или одно сложное движение, вам придется обращаться к покадровой анимации, либо использовать трансформацию формы или встроенные клипы.

Поясним это на примере. Допустим, вы хотите создать фильм, который бы показывал автомобиль, перемещающийся справа налево по рабочему листу. Причем, его колеса должны вращаться. Обычная трансформация движения, однако, реализует только одно движение в слое: либо перемещение самого автомобиля, либо вращение только одного колеса. Что делать?

Можно, конечно, сделать "классическую" покадровую анимацию, рисуя вручную каждый кадр, но это сложно. Можно создать трансформацию формы, но это еще сложнее. Поэтому лучше всего использовать вложенную анимацию. Порядок действий в этом случае такой:

1. Создаем клип, представляющий собой вращающееся колесо. Экспортируем его в формат Shockwave/Flash. (Экспорт был подробно описан в *главе 19*.)
2. Создаем основной фильм и импортируем в него клип, изображающий вращающееся колесо.
3. В основном фильме создаем графический образец или образец-клип, изображающий наш автомобиль. В него на соответствующие места помещаем два экземпляра импортированного клипа-колеса.
4. Помещаем на рабочий лист готовый экземпляр образца-автомобиля. Анимлируем его, т. е. заставляем двигаться по листу. Дело сделано.

Как видите, мы создали довольно сложную анимацию средствами трансформации движения.

Во-вторых, встроенные клипы применяются для создания сложной анимации, управляемой сценариями. Только клипы имеют соответствующие "рычаги управления", с помощью которых вы можете воздействовать на их поведение из сценариев. (Об этом, собственно, уже говорилось, но стоит

повторить еще раз.) Как создается анимация, управляемая сценариями и, возможно, откликающаяся на действия пользователя, мы рассмотрим далее в этой главе.

В-третьих, вы можете использовать встроенные клипы для создания целых Web-сайтов, размещенных во Всемирной Сети или на локальном диске. В самом деле, фильмы Flash можно рассматривать как достойную замену негибкому и порядком устаревшему языку HTML. И такие сайты уже существуют.

Ну, и, в-четвертых, клипы можно использовать для создания элементов управления Flash-приложений. Но об этом будет подробно рассказано в *главе 23*.

Использование клипов

Прежде чем приступить к рассмотрению возможностей объекта `movieClip`, нам следует выяснить, как же получить доступ к его экземплярам, т. е. встроенным клипам и основному фильму. Поверьте — это не так уж просто и не всегда очевидно. Поэтому нам обязательно нужно изучить некоторые моменты, связанные с именами и путями доступа к клипам.

Имена и пути доступа к клипам

Поскольку фильм или приложение Flash может содержать в себе очень и очень много встроенных клипов, программисту нужно иметь достаточно гибкие средства по управлению ими. Как вы знаете, эта задача решается посредством вызова свойств и методов объекта `movieClip`. А чтобы вызвать нужное свойство или нужный метод, следует обратиться к требуемому экземпляру этого объекта, представляющему данный клип на рабочем листе. Давайте же выясним, как это делается.

Обращение к экземпляру объекта осуществляется по его имени. Это вы знаете из *главы 21*. Имя клипа задается с помощью редактора свойств, а именно, с помощью поля ввода, расположенного в его верхнем левом углу (см. рис. 14.1). Это вы тоже знаете. А, поскольку Flash автоматически создает экземпляры объекта `movieClip` для всех экземпляров-клипов, использованных вами в фильме, то он присваивает им те же имена, что вы задали для этих клипов. Так что, задав в редакторе свойств имя для встроенного клипа, вы фактически задаете имя соответствующего ему экземпляра объекта `movieClip`.

Имени нужного экземпляра клипа может предшествовать имя внешнего клипа, в который он встроен. Иногда таких имен бывает достаточно много, в этом случае говорят о *пути* доступа к экземпляру клипа.

Предположим, что выполняется сценарий, находящийся в главном фильме или каком-либо встроенном клипе. Если вы хотите вызвать свойство или

метод, находящийся в том же клипе (фильме), вам нужно просто указать его, используя модификатор `this`:

```
stop();
```

Данный метод останавливает текущий клип или фильм.

Если вы хотите вызвать свойство или метод клипа, встроенного в текущий клип (фильм), вам нужно не забыть указать его имя:

```
wheel1.stop()
```

Данное выражение останавливает проигрывание клипа под именем `wheel1`. Это может быть колесо нашего автомобиля, рассмотренного в качестве примера чуть раньше. В самом деле, клип автомобиля содержит клип колеса `wheel1`, а значит, все в порядке.

Но если сценарий, содержащий это выражение, перенести из клипа автомобиля в основной фильм, оно перестанет работать. Основной фильм не содержит клипа `wheel1` — он является "собственностью" клипа автомобиля, и путь доступа к клипу `wheel1` станет некорректным.

А вот это выражение в основном фильме работать будет:

```
car.wheel1.stop();
```

Обратите внимание: сначала мы указали название клипа `car` (автомобиль), где находится клип `wheel1` (колесо), потом — название клипа колеса, а уже после него — нужный нам метод. Это потому, что клип `wheel1` является вложенным объектом клипа `car`. А клип `car`, в свою очередь, — вложенный объект основного фильма.

Если вы хотите обратиться к основному фильму из встроенного клипа, воспользуйтесь модификатором `_root`:

```
_root.stop();
```

А чтобы обратиться к внешнему объекту, воспользуйтесь модификатором `_parent`:

```
_parent.stop();
```

Пути доступа к экземплярам объектов могут быть написаны двумя различными способами. Давайте их рассмотрим, а также опишем случаи, когда нужно использовать то или иное выражение доступа.

Во-первых, вы можете записать абсолютный путь. *Абсолютный* путь всегда пишется, начиная от основного фильма, т. е. самого главного клипа:

```
_root.car.engine.drive();
```

Абсолютный путь очень хорош в тех случаях, когда вы хотите обратиться из одного встроенного клипа к другому, находящемуся очень "далеко" от первого. В такой ситуации абсолютный путь часто бывает самым коротким.

Во-вторых, вы можете задать относительный путь. *Относительный* путь записывается относительно текущего клипа. Так, если нам нужно обратиться

к фаре нашего автомобиля, а мы "находимся" в его колесе, то путь станет таким:

```
if (_parent.headlight.lamp.isInOrder) { . . .
```

Относительные пути применяются, если нужно обратиться к другому клипу, находящемуся не очень "далеко". В противном случае, лучше использовать абсолютный путь. Сравните два приведенных ниже выражения с разными видами путей:

```
_parent._parent._parent.objectInner.prop = 10;
_root.objectInner.prop = 10;
```

Относительные пути также используются при создании программного кода компонентов.

Пути доступа к экземплярам объектов различной степени вложенности могут быть весьма длинными:

```
_parent.engine.electricPart.ignitor.spark();
```

Чтобы сократить их, вы можете использовать уже знакомое вам действие **with**:

```
with (_parent.engine.electricPart) {
    headlightLeft.on();
    headlightRight.on();
    ignitor.spark();
}
```

Для написания путей в обычном режиме панели **Actions** можно использовать специально предоставляемые для этого Flash средства. Так, если вам понадобится задать путь доступа, Flash предоставит вам кнопку, показанную на рис. 22.1. При нажатии этой кнопки на экране появится диалоговое окно **Insert Target Path**, показанное на рис. 22.2.



Рис. 22.1. Кнопка **Insert a Target Path**

Большую часть этого окна занимает иерархический список клипов, содержащихся в основном фильме. Сам основной фильм также там присутствует. Набор переключателей **Mode** позволит вам выбрать способ записи пути: переключатель **Absolute** задает абсолютный путь, а переключатель **Relative** — относительный. А в поле ввода **Target** вы можете указать путь доступа к клипу вручную.

Задайте способ записи пути с помощью переключателей набора **Mode**. Выберите нужный клип в списке или введите путь в поле **Target**. И нажмите кнопку **ОК**. Если вы передумали задавать путь доступа, нажмите кнопку **Cancel**.

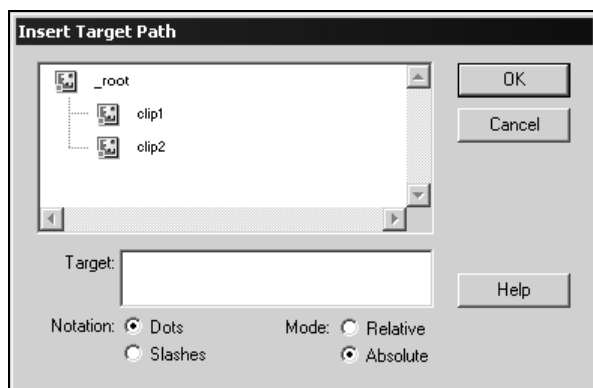


Рис. 22.2. Диалоговое окно **Insert Target Path**

Накладываемые клипы. Уровни наложения

Flash предоставляет разработчикам приложений и интерактивных фильмов еще одну очень интересную и полезную возможность. А именно — накладывать один фильм или клип на другой так, чтобы он перекрывал его, полностью или частично. При этом клипы не встраиваются друг в друга (подобно встроенным клипам), а остаются полностью независимыми. Такие клипы называются *накладываемыми*.

Вспомните — в самом начале этой главы мы говорили, как можно средствами Flash, без привлечения языка HTML строить настоящие Web-сайты. При этом как раз и используются накладываемые клипы. А именно, клип, представляющий Web-страницу, накладывается на "объединяющий" клип, содержащий заголовок сайта, полосу гиперссылок и пр. При этом оба клипа остаются независимыми.

Да, накладываемые клипы не зависят друг от друга, в отличие от встроенных. Вспомните, если вы перемещаете на рабочем листе какой-либо клип, встроенные в него клипы перемещаются вместе с ним. То же самое с изменением размеров (масштабированием), закраской и пр. Накладываемые же клипы вы можете перемещать, масштабировать и закрашивать как угодно — клипы, находящиеся под и над ними, затронуты не будут.

Вы уже знакомы с понятием порядка перекрытия графических фрагментов друг относительно друга (его еще называют *z-координатой*). В случае с накладываемыми клипами имеет место примерно то же самое, только под другим названием — *уровень* клипа.

Отсчет уровней клипов ведется, начиная с самого нижнего клипа (фильма) до самого верхнего. Исходя из этого, клипы с большим номером уровня находятся выше фрагментов с меньшим номером уровня. Однако для обозначения уровней используются не номера, а особые модификаторы, в состав которых входят номера уровней. Они имеют вид `_level<номер уровня>`.

Самый "нижний" клип имеет уровень `_level0`. Более "высокие" клипы имеют имена `_level1`, `_level2` и т. д.

Вы можете использовать эти модификаторы для доступа к накладывающимся клипам:

```
_level0.play();  
_level1._root.innerClip.stop();
```

Как видите, накладывающиеся клипы могут содержать встроенные клипы. Это нормальная ситуация; в самом деле, Flash зачастую не оставляет нам другого выбора, кроме как использовать вложенную анимацию.

Простейшие манипуляции над клипами

Здесь мы рассмотрим самые простые действия, которые вы можете выполнить как с основным фильмом, так и со встроенными клипами. Мы узнаем, что с ними можно проделывать, используя сценарии ActionScript, и как все это может помочь нам в нашей работе. Также мы рассмотрим несколько простейших примеров, которые покажут вам, на что способны язык программирования ActionScript и сам Flash.

Имейте также в виду, что управление основным фильмом осуществляется аналогичными способами. В самом деле, ведь и основной фильм есть не что иное, как экземпляр объекта `movieClip`, т. е. фактически большой клип, в который вложены более мелкие клипы. Конечно, здесь будут иметь место некоторые ограничения, но основные принципы те же самые, что и в случае встроенных клипов. Поэтому в дальнейшем термины "фильм" и "клип" будут означать одно и то же, если не оговорено противоположное.

Управление проигрыванием клипа

Обычно каждый клип (и основной фильм в том числе) начинает проигрываться сразу же после загрузки и проигрывается до последнего кадра. После этого проигрывание начинается сначала, если в настройках экспорта фильма был задан параметр "зацикливания". Однако, используя сценарии, вы можете управлять этими процессами, в частности, останавливать и запускать проигрывание, а также проигрывать фильм, начиная с заданного кадра.

Вы уже знакомы с действием `stop`. Оно останавливает проигрывание текущего фильма. Вы можете включить это действие в сценарий, привязанный к кадру, встроенному клипу или кнопке.

Если вам нужно остановить не *текущий клип* (то есть клип, к которому привязан данный сценарий), а другой, воспользуйтесь методом `stop` объекта `movieClip`.

```
car.stop();  
_parent._parent.stop();  
this.stop();
```

Последнее выражение аналогично простому вызову действия `stop`.

Для запуска проигрывания фильма с того места, где он был остановлен, или с начала, воспользуйтесь действием `play` или одноименным методом объекта `movieClip`.

```
car.play();  
_parent._parent.play();  
this.play();
```

Для перехода на какой-либо кадр текущего клипа вы можете воспользоваться действиями `gotoAndPlay` и `gotoAndStop`. Первое действие выполняет переход на заданный кадр и запускает проигрывание фильма с него. Второе же действие выполняет переход на заданный кадр, но не запускает проигрывание.

Оба этих действия принимают два параметра. Первый параметр задает номер или имя кадра, на который будет выполнен переход. Если значение этого параметра имеет числовой тип, то оно трактуется как номер, если строковый — как имя кадра. Второй, необязательный, параметр задает имя сцены, если он пропущен, то выполняется переход на кадр текущей сцены.

```
gotoAndPlay(10);  
gotoAndStop(x + 1, "Эпизод 1");
```

Вы также можете пользоваться методами `gotoAndPlay` и `gotoAndStop` объекта `movieClip`. С их помощью можно управлять любым встроенным клипом и основным фильмом. Учтите, однако, что эти методы не столь мощны, как одноименные действия. Во-первых, они не могут выполнять переход на кадры другой сцены (и, следовательно, принимают только один параметр). Во-вторых, они могут выполнять переход только по номеру кадра (по имени не могут). Так что для этих целей вам лучше пользоваться действиями `gotoAndPlay` и `gotoAndStop`.

Осталось сказать еще о двух методах объекта `movieClip`. Эти методы не имеют эквивалентных им действий. Метод `prevFrame` перемещает указатель на предыдущий кадр клипа или фильма, а метод `nextFrame` — на следующий. Ни один из этих методов не принимает параметров.

Вы можете использовать эти методы, например, для создания слайд-шоу средствами Flash. Для этого вы помещаете все изображения (слайды) в отдельных кадрах вашего фильма. К первому кадру фильма вы привязываете сценарий, содержащий одно-единственное действие — `stop()`, которое остановит ваш фильм на первом же кадре. Теперь осталось создать кнопки и привязать к ним сценарии, содержащие выражения `this.prevFrame()` и `this.nextFrame()`, которые будут выполнять переход соответственно на предыдущий и последующий кадр фильма. Все — простейшее слайд-шоу готово!

Загрузка и выгрузка клипов

Flash предоставляет вам возможность загружать дополнительные клипы и проигрывать их, используя особые действия языка ActionScript. Причем, вы можете как заменять основной фильм другим, так и помещать вновь загруженные клипы над или под ним, задав соответствующий уровень. Иными словами, вы можете создавать накладывающиеся клипы. Вместо клипов, сохраненных в распространяемом формате Shockwave/Flash, вы можете загружать таким образом графические изображения JPEG.

Для загрузки клипов или изображений JPEG вам следует использовать действие `loadMovie`. Это действие загружает новый клип (или новое изображение) и заменяет им уже существующий встроенный клип основного фильма. Формат его вызова таков:

```
loadMovie("<Интернет-адрес клипа или изображения>",  
⌘<Путь клипа или изображения>);
```

Учтите, что новый клип (или новое изображение) должен находиться в том же интернет-домене, что и основной фильм. Это ограничение предусмотрено из-за соображений безопасности.

Примеры использования действия `loadMovie`:

```
loadMovie("/images/truckWheel.swf", _root.car.wheel2);  
loadMovie(sURL, _root.car);
```

Сначала мы заменяем одно из колес нашего автомобиля на другое, явно позаимствованное у грузовика. А потом заменяем сам автомобиль другим клипом, адрес загрузки которого хранится в переменной `sURL`.

Для загрузки нового клипа или изображения JPEG вы можете также использовать метод `loadMovie` объекта `movieClip`. Этот метод заменяет текущий клип новым.

```
_root.car.wheel2.loadMovie("/images/truckWheel.swf");
```

Действие `loadMovieNum` аналогично `loadMovie`, но загружает клип и делает его накладывающимся. А в качестве второго параметра оно принимает уровень клипа.

```
loadMovieNum("<Интернет-адрес клипа или изображения>", <Уровень клипа>);
```

Пара примеров использования действия `loadMovieNum`:

```
loadMovieNum("/images/banner.swf", 2);  
loadMovieNum("/images/static/motorcycle.jpg", 0);
```

Первое выражение загружает рекламный баннер и помещает его над нашим автомобилем, чтобы вы смогли по достоинству оценить назойливость современной рекламы. Второе же выражение просто заменяет весь наш фильм изображением мотоцикла.

После того, как нужда в загруженном клипе отпадет, самое время его выгрузить. Зачем это нужно? Хотя бы затем, чтобы освободить память компьюте-

ра пользователя для других, более важных задач. Ведь оперативной памяти, как вы прекрасно знаете, слишком много не бывает, лучше не занимать ее понапрасну всяким бараклом. Именно поэтому язык ActionScript предоставляет вам два действия: `unloadMovie` и `unloadMovieNum`. Оба выгружают из памяти ранее загруженный клип: первое — встроенный, а второе — накладывающийся.

Форматы вызова этих действий:

```
unloadMovie(<Путь клипа или изображения>);  
unloadMovieNum(<Уровень клипа>);
```

Примеры:

```
unloadMovie(_root.car.headlight2);  
unloadMovieNum(2);
```

Сначала мы избавляемся от фары нашего автомобиля. А потом с наслаждением "убиваем" назойливо маячащий перед глазами рекламный баннер.

А эти два выражения позволят вам выгрузить сам основной фильм:

```
unloadMovie(_level0._root);  
unloadMovieNum(0);
```

после чего проигрыватель Flash автоматически выгрузит все встроенные и накладывающиеся клипы и завершит свою работу.

Обеспечение безопасности при загрузке клипов

Здесь скажем еще несколько слов о загрузке внешних клипов из сценариев ActionScript.

Дело в том, что Flash позволяет загружать клипы только из того же *поддомена*, откуда загружен основной фильм. Это значит, что вы не сможете загрузить клип с другого сайта, — имейте это в виду, когда будете создавать фильмы, состоящие из множества клипов.

Для выяснения поддомена существуют два простых правила. Их совсем не трудно запомнить.

- ❑ Правило первое: если *доменное имя* (то, что мы набираем в строке адреса Web-обозревателя) сайта содержит один или два компонента, то поддомен равен этому доменному имени. Так, для доменного имени сайта магазина "Озон" **http://ozon.ru** поддомен будет таким же — **http://ozon.ru**.
- ❑ Правило второе: если доменное имя сайта содержит три и более компонента, то для получения поддомена удалите последний — самый левый — компонент. В случае сайта фирмы Macromedia **http://www.macromedia.com** поддоменом будет **http://macromedia.com**. В данном случае вы сможете загружать клипы с интернет-адресов **http://support.macromedia.com** и **http://ftp.macromedia.com**, но ничего не сможете загрузить с сайта Adobe **http://www.adobe.com**.

Зачем все это сделано? Для обеспечения безопасности. Благодаря этому условию сильно ограничиваются возможности создания компьютерных вирусов и вредоносных программ ("троянцев") на Flash. Полученный вами из заслуживающего доверия источника фильм Flash ничего не сможет загрузить с чужого сайта.

Использование обработчиков событий

Здесь мы опишем обработчики событий, перечислим события, которые вы можете обрабатывать, и приведем несколько примеров их обработки.

Вы уже знаете, что такое обработчик событий. Мы достаточно часто упоминали о них ранее и даже рассматривали примеры простейших обработчиков событий. Также мы описали синтаксис обработчика события для клипа и кнопки в *главе 21*.

Итак, для клипа обработчик события имеет вид:

```
onClipEvent(<Событие>) {  
    Код обработчика  
}
```

Для встроенного клипа можно создать обработчики, реагирующие на самые разные события. Все доступные события перечислены в табл. 22.1.

Таблица 22.1. События, принимаемые встроенным клипом

Событие	Описание
data	Наступает при приеме клипом внешних данных (если использовалось действие loadVariables) или загрузке очередного фрагмента клипа (если использовалось действие loadMovie)
enterFrame	Наступает при воспроизведении очередного кадра фильма. Предшествует всем другим событиям
keyDown	Наступает при нажатии любой клавиши на клавиатуре
keyUp	Наступает при отпускании любой клавиши на клавиатуре
load	Наступает сразу после загрузки встроенного клипа и его появления на экране
mouseDown	Наступает при нажатии левой кнопки мыши
mouseMove	Наступает при перемещении курсора мыши над встроенным клипом
mouseUp	Наступает при отпускании левой кнопки мыши
unload	Наступает после того, как клип пропадет с экрана, при воспроизведении следующего кадра (в котором уже нет этого клипа). Предшествует всем другим событиям этого кадра

Здесь нужно дать некоторые пояснения.

Событие `enterFrame` происходит при воспроизведении очередного кадра основного фильма. Это значит, что обработчик данного события будет выполняться постоянно с частотой, равной частоте кадров вашего фильма, пока фильм не закончится. Это можно использовать для выполнения различных задач, например, для отслеживания курсора мыши.

С помощью событий `load` и `unload` можно реализовать запуск или останов воспроизведения других клипов. Например, при загрузке какого-либо клипа воспроизведение другого клипа может приостанавливаться, а после выгрузки — продолжаться. Также вы можете создать сценарий, загружающий после выгрузки одного клипа другой и запускающий его воспроизведение.

События `mouseDown` и `mouseUp` можно, по идее, применить для создания на рабочем листе подобия кнопок. Однако, если привязать к обычным встроенным клипам обработчики этих событий, то они будут срабатывать не совсем корректно. А именно, при щелчке по встроенному клипу будут срабатывать обработчики соответствующего события во всех других клипах. Вероятно, так было предусмотрено специально, но не всегда это бывает полезным. Поэтому лучше для таких целей создавать полноценные кнопки Flash. Создание таких кнопок будет описано в *главе 23*.

Если же кнопки нужны вам прямо сейчас, вы можете поступить следующим образом. Во-первых, можно создать обычный образец, выбрав в диалоговом окне **Create New Symbol** (на рис. 10.2 показано аналогичное диалоговое окно **Convert to Symbol**) переключатель **Button**, и ни на что больше не обращая внимание. Во-вторых, можно преобразовать клип в кнопку, выбрав в раскрывающемся списке в левом верхнем углу редактора свойств пункт **Button**.

Для кнопки обработчик события имеет вид:

```
on(<Событие>) {  
    Код обработчика  
}
```

А все доступные для кнопки события перечислены в табл. 22.2.

Таблица 22.2. События, принимаемые кнопкой

Событие	Описание
<code>press</code>	Наступает при нажатии левой кнопки мыши, когда курсор мыши находится на кнопке
<code>release</code>	Наступает при отпускании левой кнопки мыши, когда курсор мыши находится на кнопке
<code>releaseOutside</code>	Наступает при отпускании левой кнопки мыши, когда курсор мыши находится вне кнопки (когда кнопка была нажата, курсор мыши находился на кнопке)

Таблица 22.2 (окончание)

Событие	Описание
rollOut	Наступает, когда курсор мыши "уходит" с кнопки
rollOver	Наступает, когда курсор мыши "заходит" на кнопку
dragOut	Наступает, когда курсор мыши "уходит" с кнопки, причем левая кнопка мыши нажата
dragOver	Наступает, когда курсор мыши "заходит" на кнопку, причем левая кнопка мыши нажата
keyPress("<Код>")	Наступает при нажатии клавиши с заданным в качестве параметра кодом

Вы можете присваивать встроенным клипам обработчики событий кнопок. Эти обработчики будут срабатывать при наступлении "кнопочных" событий. Для этого просто запишите нужный обработчик, используя действие `on`, а не `onClipEvent`.

Примеры обработчиков событий приведены в конце этой главы.

Управление экземплярами-клипами

А теперь рассмотрим средства, предоставляемые Flash для управления клипами, находящимися на рабочем листе.

Создание и удаление экземпляров-клипов

Вы можете создавать клипы из имеющихся в библиотеке, дублировать их, удалять дубли и даже создавать новые клипы.

Проще всего продублировать уже имеющийся на рабочем листе экземпляр-клип вместе с его графикой, анимацией, переменными и сценариями. Для этого используйте действие `duplicateMovieClip`. Формат его вызова таков:

```
duplicateMovieClip(<Путь клипа>, "<Имя нового клипа>",
    &<Уровень клипа>);
```

Первым параметром задается путь исходного клипа, который вы хотите скопировать. Вторым параметром задается имя нового экземпляра клипа; оно должно иметь строковый тип и быть уникальным. Последним, третьим, параметром задается уровень нового клипа по отношению к исходному.

Пример:

```
duplicateMovieClip(_root.car.wheel2, "wheel5", 0);
```

Мы только что приделали к нашему автомобилю пятое колесо. А вдруг пригодится!

Имейте в виду: если вы удалите исходный клип, то будут также удалены все скопированные с него клипы. Также учтите, что содержимое переменных исходного клипа не копируется в новый клип.

Чтобы удалить ненужную копию клипа, воспользуйтесь действием `removeMovieClip`:

```
removeMovieClip(<Путь скопированного клипа>);
```

Например:

```
removeMovieClip(_root.car.wheel5);
```

Пятое колесо автомобилю все равно без нужды, так что удалим его.

Вы также можете пользоваться методами `duplicateMovieClip` и `removeMovieClip` объекта `movieClip`. Форматы их вызова:

```
<Исходный клип>.duplicateMovieClip("<Имя нового клипа>",
```

```
❏ <Уровень клипа>);
```

```
<Скопированный клип>.removeMovieClip();
```

Есть еще один способ поместить клип на рабочий лист. А именно: создать новый клип, основанный на образце-клипе, который вы создали ранее и поместили в библиотеку. Для этого в объекте `movieClip` предусмотрен метод `attachMovie`.

```
<Клип-родитель>.attachMovie("<Имя разделяемого образца>",
```

```
❏ "<Имя нового клипа>", <Уровень клипа>);
```

Метод `attachMovie` вызывается для того клипа, в который вы хотите встроить новый клип. Второй и третий параметры вам уже знакомы. А вот о первом параметре нужно поговорить подробнее.

Дело в том, что для успешного использования в методе `attachMovie` образец-клип нужно сделать разделяемым. Вы, конечно, помните, как это делается (это было описано в *главе 10*). Сначала нужно выделить в окне библиотеки нужный образец, потом выбрать пункт **Linkage** контекстного или дополнительного меню и задать нужные параметры в диалоговом окне **Linkage Properties** (см. рис. 10.27). Вам потребуется ввести имя разделяемого образца в поле **Identifier** и включить флажок **Export for Runtime Sharing**. (И, конечно, нажать кнопку **ОК**, но это очевидно.)

Однако, в нашем случае все немного по-другому. Прежде всего, чтобы сделать образец разделяемым, нужно включить другой флажок — **Export for ActionScript**. После этого наш образец станет доступным для использования в сценариях. Дадим таким образцам название *сценарных*.

Далее нам нужно решить, когда будут загружаться сценарные образцы. По умолчанию все они загружаются перед тем, как проигрыватель Flash выведет первый кадр фильма. Если таких образцов слишком много, или они очень велики, пауза перед выводом первого кадра может затянуться. Вы можете заставить Flash загружать такие образцы только тогда, когда они действи-

тельно понадобятся. Для этого отключите в окне **Linkage Properties** (см. рис. 10.27) флажок **Export in First Frame**.

Однако здесь нас подстерегает другая опасность. Если вы не использовали где-либо в фильме сценарный образец, то Flash не перенесет его в фильм при экспорте, справедливо полагая, что он здесь лишний. Что вы там написали в сценарии, его не касается. Так вот, чтобы Flash все же сохранил сценарный образец в готовом фильме, вам придется все-таки поместить его экземпляр на рабочий лист, задав для него либо очень маленькие размеры, либо очень большую прозрачность.

После этого вы можете использовать готовый сценарный образец:

```
_root.car.attachMovie("headlight", "headlight3", 0);
```

Это выражение добавит нашему автомобилю третью фару. Вот она-то нам точно пригодится!

Когда отпадет нужда во встроенном клипе, созданном из сценарного образца, вы сможете выгрузить его из памяти, использовав уже знакомый вам метод `removeMovieClip` или одноименное действие. Можно также воспользоваться действием или методом `unloadMovie`.

Ну, и последняя интересная возможность, которую мы здесь рассмотрим, пока что вам не пригодится. Однако она будет очень полезна в будущем, поэтому мы все же упомянем о ней. Flash позволяет вам поместить на рабочий лист новый, "пустой" встроенный клип. Это выполняется вызовом метода `createEmptyMovieClip` объекта `movieClip`.

```
<Клип-родитель>.createEmptyMovieClip("<Имя нового клипа>",  
    &lt;Уровень клипа>);
```

Метод `createEmptyMovieClip` вызывается для того клипа, в который вы хотите встроить новый клип. Оба его параметра вам уже знакомы.

Пример использования этого метода:

```
_root.createEmptyMovieClip("tractor", 0);
```

Это выражение создает компанию нашему автомобилю — трактор. Конечно, никакого трактора там пока нарисовано не будет, но лиха беда начало...

Изменение параметров встроенных клипов

Неужели вы думали, что Flash может только создавать, дублировать и удалять встроенные клипы? Отнюдь! Сейчас мы рассмотрим свойства и методы объекта `movieClip`, с помощью которых вы можете изменять различные параметры встроенных клипов.

Первым делом мы познакомимся со свойствами `_x` и `_y`. Они предоставляют доступ к горизонтальной и вертикальной координате точки фиксации клипа. Причем координаты эти измеряются не относительно рабочего листа, а относительно внешнего клипа. Запомните это.

Свойства `_width` и `_height` содержат соответственно ширину и высоту клипа.

Свойства `_xscale` и `_yscale` задают масштабирование клипа соответственно по горизонтали и вертикали. Так, если задать для свойства `_xscale` значение 50, то клип сожмется по горизонтали вдвое.

Свойство `_alpha` позволяет получить или задать прозрачность клипа. Доступны любые целые значения от 0 (полная прозрачность) до 100 (полная непрозрачность).

Свойство `_rotation` задает угол поворота клипа в градусах.

```
onClipEvent(enterFrame) {  
    ++this._rotation;  
    --this._alpha;  
}
```

Этот обработчик события заставляет встроенный клип плавно поворачиваться и одновременно плавно исчезать. Таким образом, вы можете создавать настоящую анимацию только средствами `ActionScript`.

Свойство `_visible` позволяет сделать клип невидимым. Оно имеет логический тип: значение `true` делает клип видимым, а значение `false` — невидимым.

```
onClipEvent(mouseDown) {  
    this._visible = ~this._visible;  
}
```

Кроме перечисленных выше, объект `movieClip` поддерживает набор особых свойств и методов, доступных только для чтения. Эти свойства и методы возвращают различную служебную информацию, зачастую не относящуюся к самому клипу. Ниже перечислены некоторые из них.

Свойства `_xmouse` и `_ymouse` возвращают соответственно горизонтальную и вертикальную координаты курсора мыши относительно точки фиксации клипа. Пользуясь этими свойствами и обработчиком события `mouseMove`, вы можете контролировать перемещение мыши.

```
onClipEvent(enterFrame) {  
    myCursor._x = _root._xmouse;  
    myCursor._y = _root._ymouse;  
}
```

Приведенный выше обработчик события `enterFrame` заставляет встроенный клип перемещаться вслед за курсором мыши. Фактически встроенный клип ведет себя при этом как курсор мыши.

Свойство `_currentframe` возвращает номер кадра, на котором в данный момент стоит указатель (фактически, номер проигрываемого в данный момент кадра). А общее количество кадров клипа возвращает свойство `_totalframes`.

Свойство `_framesloaded` возвращает количество уже загруженных кадров клипа. Аналогично, методы `getBytesLoaded` и `getBytesTotal` возвращают количество загруженных байтов клипа и полный размер клипа в байтах. Используя их, вы можете создавать индикаторы загрузки клипа или фильма.

```
with (_root) {  
    loadPercent = _framesloaded / _totalframes * 100;  
    loadPercent2 = getBytesLoaded / getBytesTotal * 100;  
}
```

Свойство `_url` возвращает интернет-адрес, с которого был загружен клип или фильм. Иногда это тоже может пригодиться.

Drag'n'drop

Термином "*drag'n'drop*" ("тащи и бросай") обозначают целый набор операций, связанных с перетаскиванием каких-либо объектов операционной системы (файлов, папок, ярлыков), системных или прикладных программ. Так, во Flash вы используете операции перетаскивания для изменения местоположения графических фрагментов и указателя кадра. Как правило, в различных программах (в том числе, в самой операционной системе) операции "drag'n'drop" выполняют сходные функции.

Вы можете сделать один из встроенных клипов вашего фильма или приложения Flash перетаскиваемым. Это выполняется с помощью действия `startDrag`.

```
startDrag(<Путь клипа>, [<За центр>, <X1>, <Y1>, <X2>, <Y2>]);
```

В качестве единственного обязательного параметра передается путь к клипу, который вы хотите сделать перетаскиваемым.

Действие `startDrag` может принимать еще пять параметров. Второй параметр имеет логический тип и позволяет задать точку, к которой будет автоматически "приклеиваться" курсор мыши: точку, по которой пользователь щелкнул мышью (значение `false`) или центр клипа (значение `true`). Параметры `x1` и `y1` задают горизонтальную и вертикальную координаты левого верхнего угла воображаемого прямоугольника, внутри которого можно будет перетаскивать этот клип, и за пределы которого он не сможет выйти. А за координаты правого нижнего угла "отвечают" параметры `x2` и `y2`. Эти координаты задаются относительно внешнего клипа.

```
startDrag(_root.car);
```

Можно также использовать метод `startDrag` объекта `movieClip`:

```
<Путь клипа>.startDrag([<За центр>, [<X1>, <Y1>, <X2>, <Y2>]]);  
_root.car.startDrag(true);
```

После вызова этого действия или метода клип станет перетаскиваемым. При этом пользователь сможет "ухватить" его мышью и перетащить на новое место.

Вы также можете задать границы области, в которой можно будет "таскать" клип:

```
_root.car.startDrag(true, 100, 100, 500, 500);
```

Чтобы отменить возможность перетаскивания клипа, либо вызовите действие или метод `startDrag` для другого клипа (только один клип в данный момент времени может быть перетаскиваемым), либо вызовите действие или метод `stopDrag`. Это действие (метод) не принимает параметров.

```
stopDrag();
```

```
_root.car.stopDrag();
```

Для запуска процесса перетаскивания клипа и обработки его "отпускания" вы можете использовать обработчики событий `press` и `release`. А поскольку это не события встроенного клипа, а события кнопки, вам нужно будет предварительно преобразовать ваш клип в кнопку.

```
on(press) {  
    this.startDrag(true);  
}  
on(release) {  
    this._x = _root._xmouse;  
    this._y = _root._ymouse;  
    this.stopDrag();  
}
```

Это простейшие обработчики перетаскивания клипа. Первый просто запускает операцию перетаскивания клипа. Второй же завершает эту операцию и одновременно помещает клип на то место, куда он был "брошен" пользователем. Конечно, вы можете создать и более сложные обработчики.

Объект `movieClip` предоставляет доступное только для чтения свойство `_droptarget`. Это свойство возвращает "цель" операции "drag'n'drop", т. е. клип, в который был "брошен" текущий клип. Вы можете проверить значение, возвращаемое этим свойством, и в зависимости от этого завершить или не завершить операцию перетаскивания.

Здесь имеет место небольшое затруднение. Дело в том, что свойство `_droptarget` возвращает результат в особом формате, применяемом в старых версиях Flash. Это так называемая "запись со слэшем", когда для разделения имени экземпляра объекта и свойства или метода применяется косая черта (/), а не точка. Для того чтобы преобразовать "запись со слэшем" в "запись с точкой", нужно использовать функцию `eval`.

```
on(release) {  
    if (eval(this._droptarget) != _root.forbiddenArea) {  
        this._x = _root._xmouse;  
        this._y = _root._ymouse;
```

```

        this.stopDrag();
    }
}

```

Вышеуказанный обработчик запрещает пользователю "бросать" перетаскиваемый элемент в клип под названием `forbiddenArea`. Вы также можете, наоборот, разрешить "бросать" перетаскиваемый элемент только в одно определенное место.

Создание фигурного курсора мыши

Очень многие приложения (не только созданные во Flash, но и обычные, работающие под Windows) "щеголяют" фигурными курсорами. Надо сказать, что зачастую такие курсоры действительно нужны и полезны: они помогают пользователю работать с программой. Но иногда фигурные курсоры делают только ради красоты. Это не совсем правильный подход: пользователь привык к хорошо знакомому ему курсору-стрелочке, и любые нововведения, если они ему не помогают, будут излишними. А наша главная задача: чтобы пользователю было удобно.

Что такое "хорошо" и что такое "плохо", мы выяснили. Осталось узнать совсем немного: как средствами Flash создать фигурный курсор мыши. А создается он так же, как и уже знакомые вам перетаскиваемые клипы.

Создайте новый образец-клип и нарисуйте в нем желаемый курсор. Поместите экземпляр этого клипа на рабочий лист и назовите его `cursor`.

Теперь присвойте этому клипу такой обработчик события `load`:

```

onClipEvent(load) {
    this.startDrag(true);
}

```

Этот обработчик будет сразу же при окончании загрузки клипа делать его перетаскиваемым. Запустите просмотр только что созданного фильма и убедитесь в этом сами.

Казалось бы, все замечательно. Одно плохо: "родной" курсор мыши, рисуемый самой Windows, портит всю картину. Его нужно как-то скрыть. Для этого воспользуйтесь методом `hide` пока еще не изученного объекта `Mouse` (он будет рассмотрен далее в этой главе). Этот метод скрывает "родной" курсор мыши.

Новый обработчик будет выглядеть так:

```

onClipEvent(load) {
    Mouse.hide();
    this.startDrag(true);
}

```

Вот теперь все работает, как надо.

Вы можете задать область, в которой будет действовать созданный вами курсор мыши. Для этого просто измените второе выражение приведенного выше обработчика:

```
this.startDrag(true, 200, 100, 600, 400);
```

После этого ваш фигурный курсор будет "заперт" в этих границах.

Выявление совпадений

Во многих приложениях, например играх, нужно знать, находится ли заданная точка внутри какого-либо графического элемента. Эта задача — выявление совпадений — решается в разных случаях по-разному. И, как правило, ее зачастую весьма непростое решение ложится на плечи программиста.

В нашем случае, однако, все намного проще. Разработчики Flash предусмотрели особый метод объекта `movieClip` — `hitTest`. С его помощью вы можете проверять, находится ли заданная точка внутри клипа, или касаются ли друг друга два отдельных клипа.

Метод `hitTest` имеет два формата вызова. Первый из этих форматов позволяет выяснить совпадение точки и клипа:

```
<Клип>.hitTest(<X>, <Y>, <Проверять контур>);
```

Как вы уже поняли, первые два параметра задают соответственно горизонтальную и вертикальную координату нужной вам точки. Третий параметр имеет логический тип и позволяет задать, что считать границами клипа: его контур (значение `true`) или окружающий его прямоугольник (значение `false`).

Если заданная точка находится внутри клипа, метод `hitTest` возвращает `true`. В противном случае возвращается `false`. Вы можете использовать это значение в условном выражении или еще где-либо.

Давайте проверим, как работает этот метод. Создадим новый клип, представляющий собой сложную фигуру из множества кривых. Поместим его на рабочий лист и присвоим ему имя `hit`. После этого присвоим этому клипу обработчик события `enterFrame`:

```
onClipEvent(enterFrame) {  
    if (this.hitTest(_root._xmouse, _root._ymouse, true)) {  
        this._alpha = 50;  
    } else {  
        this._alpha = 100;  
    }  
}
```

Этот сценарий постоянно будет вызываться и проверять, находится ли курсор мыши над нашей фигурой. Если это так, то фигура делается полупрозрачной, в противном случае — полностью непрозрачной.

Заметьте, что метод `hitTest` проверяет только "попадание" в саму фигуру, если его третий параметр равен `true`. Если же его приравнять `false`, то этот метод станет проверять попадание в воображаемый прямоугольник, описанный вокруг фигуры. Это более грубая проверка, но, вероятно, она тоже иногда нужна, если разработчики Flash включили ее в свое творение.

Второй формат метода `hitTest` позволит вам проверить совпадение двух различных клипов, т. е. накладываются они или перекрываются:

```
<Клип 1>.hitTest(<Клип 2>);
```

В смысле передаваемых параметров здесь все ясно. И возвращаемое значение то же самое.

Имейте в виду, что метод `hitTest`, записанный во втором формате, проверяет "попадание" не фигуры в фигуру, а воображаемого прямоугольника, описанного вокруг первой фигуры, в воображаемый прямоугольник, описанный вокруг второй фигуры. Следовательно, этот метод ведет себя так, словно несуществующий третий его параметр равен `false`.

Теперь осталось проверить работу метода. Создайте новый образец-клип со сложной фигурой, поместите его экземпляр на рабочий лист, сделайте его поменьше и назовите `cursor`. К первому (и единственному) кадру фильма привяжите следующий небольшой сценарий:

```
_root.cursor.startDrag(true);
```

Этот сценарий заставляет клип `cursor` двигаться за мышью.

Немного изменим обработчик события `enterFrame` для клипа `hit`:

```
onClipEvent(enterFrame) {  
    if (this.hitTest(_root.cursor)) {  
        this._alpha = 50;  
    } else {  
        this._alpha = 100;  
    }  
}
```

Как видите, здесь было сделано одно-единственное изменение (выделено полужирным шрифтом). Мы поменяли формат вызова метода `hitTest`. Теперь можно проверить, как все это работает.

Работа с графикой

Хоть Flash и предоставляет исключительно мощные средства рисования графики и создания анимации, иногда нужно кое-что дорисовать в процессе показа фильма. Для этого существует целый набор методов и свойств объекта `movieClip`, которые служат для рисования различных линий и заливок. Графика, нарисованная с их помощью, называется *созданной программно* или просто программной.

Вы уже знаете о маскирующих слоях, изображениях-масках и маскируемых изображениях. Вы знаете, как их создавать и использовать. Но Flash предоставляет вам кое-что еще. А именно: операцию преобразования любого встроенного клипа (в том числе, и нарисованного вами) в маску. Представляете, какие это открывает возможности!

Рисование

Вы можете смешивать в одном клипе (фильме) графику, нарисованную "вручную" в среде Flash и созданную программно. Однако здесь нужно запомнить один момент: все, что вы нарисовали в среде Flash, выводится поверх нарисованного из сценария. Поэтому, если клип содержит много графики, нарисованной "вручную", то она перекроет графику, созданную программно. Идеальный путь решения этой проблемы — создание специальных "пустых" клипов, предназначенных для рисования программной графики, либо в среде Flash, либо вызвав описанный выше метод `createEmptyMovieClip` объекта `movieClip`.

При программном рисовании графики используется концепция *пера*. (Не путать с инструментом "перо".) Перо — это единственный инструмент, которым вы можете рисовать линии программно. Ни "прямоугольник", ни "карандаш", ни прочие знакомые вам инструменты здесь не доступны.

Перо можно перемещать на любое место клипа, используя метод `moveTo`, при этом ничего рисоваться не будет — вы просто ставите перо в нужную точку. В качестве параметров этого метода вы задаете новые горизонтальную и вертикальную координаты пера. Эти координаты отсчитываются относительно точки фиксации клипа.

```
<Клип>.moveTo(<X>, <Y>);
```

Чтобы нарисовать прямую линию, используйте метод `lineTo`. Параметры этого метода аналогичны предыдущему.

```
<Клип>.lineTo(<X>, <Y>);
```

Пример:

```
with (_root.paintBox) {  
    moveTo(100, 200);  
    lineTo(300, 200);  
    lineTo(200, 100);  
    lineTo(100, 200);  
}
```

Этот сценарий нарисует в клипе `paintBox` треугольник.

Рисование кривых выполняется чуть сложнее. Для этого вам нужно использовать метод `curveTo`, принимающий четыре параметра:

```
<Клип>.curveTo(<AX>, <AY>, <X>, <Y>);
```

Параметры x и y задают соответственно горизонтальную и вертикальную координаты конечной точки кривой, в которой остановится перо. Параметры AX и AY задают горизонтальную и вертикальную координаты точки искривления. Все эти координаты отсчитываются опять же от относительно точки фиксации клипа.

Пример:

```
with (_root.paintBox) {  
    moveTo(300, 300);  
    curveTo(400, 300, 400, 200);  
    curveTo(400, 100, 300, 100);  
    curveTo(200, 100, 200, 200);  
    curveTo(200, 300, 300, 300);  
}
```

Этот сценарий нарисует окружность или нечто на нее похожее. Других способов нарисовать программно окружность нет.

Для рисования линии будут использованы текущие параметры ее стиля: толщина, цвет и прозрачность. Поскольку неизвестно, какие параметры стиля линии были заданы в настоящий момент, лучше задать их перед любыми операциями рисования. Это выполняется с помощью метода `lineStyle`.

```
<Клип>.lineStyle([<Толщина линии>, [<Цвет линии>,  
    ↪ [<Прозрачность линии>]]]);
```

Толщина линии задается в пикселах, от 0 до 255. Если она не задана, то линия нарисована не будет. Цвет линии задается в формате `0xRRGGBB`, где `RR` — доля красного цвета, `GG` — зеленого и `BB` — синего соответственно. Пример задания цвета: `0x22FF44`. Прозрачность линии может принимать значения от 0 (полная прозрачность) до 100 (полная непрозрачность).

Учтите: если в методе `lineStyle` не был задан ни один параметр (а такое возможно), то ни одна линия, скорее всего, вообще не будет нарисована. Поэтому лучше задать хотя бы один параметр, а самое правильное — все.

```
with (_root.paintBox) {  
    lineStyle(2, 0x000000);  
    moveTo(100, 200);  
    lineTo(300, 200);  
    lineTo(200, 100);  
    lineTo(100, 200);  
}
```

Именно так должен выглядеть нормально работающий сценарий рисования треугольника. Прежде, чем нарисовать что-либо, мы с помощью метода `lineStyle` задаем стиль линии: толщина в 2 пиксела и черный цвет.

А теперь давайте посмотрим на только что нарисованный треугольник и подумаем: чего же в нем не хватает? В *главе 5* мы выяснили, что Flash автоматически создает заливку, если мы нарисовали замкнутую фигуру, не имеет значения, каким инструментом мы при этом пользовались. Здесь же заливки нет. Почему?

Вероятно, разработчики Flash решили, что, если вы пользуетесь сценариями ActionScript, значит, вы достаточно квалифицированный специалист по Flash, и вам не нужна особая помощь. Поэтому вам придется самостоятельно создавать заливку для нарисованных программно фигур.

Для создания заливки служат два метода. Первый из них — `beginFill` — указывает Flash, что с этого момента нужно отслеживать все операции программного рисования и при создании замкнутой фигуры сделать заливку. Второй — `endFill` — прекращает отслеживание операций рисования и собственно рисует заливку (если, конечно, ее можно нарисовать).

Формат вызова метода `beginFill`:

```
<Клип>.beginFill([<Цвет линии>, [<Прозрачность заливки>]]);
```

Цвет линии задается в формате RGB, а прозрачность — значением от 0 до 100. Опять же, если не задан ни один параметр, то заливка, скорее всего, не будет нарисована.

Метод `endFill` не принимает ни одного параметра.

Нарисуем теперь треугольник с заливкой:

```
with (_root.paintBox) {  
    lineStyle(2, 0x000000);  
    beginFill(0x222233);  
    moveTo(100, 200);  
    lineTo(300, 200);  
    lineTo(200, 100);  
    lineTo(100, 200);  
    endFill();  
}
```

С помощью метода `beginFill` создается обычная, сплошная заливка. Если же вы хотите создать градиентную заливку, то нужно будет воспользоваться методом `beginGradientFill` в паре с методом `endFill`. Полное описание этого метода приведено в *приложении 1*.

Чтобы удалить все, что вы нарисовали, вызовите метод `clear`. Он удалит всю созданную программно графику и сбросит заданный вами стиль линии.

```
_root.paintBox.clear();
```

Пример использования операций рисования

Для того чтобы вы не боялись экспериментировать, приведем небольшой пример рисования изображения в процессе проигрывания фильма. А именно, нарисуем график функции $f(x) = x^2$. Это, скорее, курьез, чем реально необходимый пример, но он поможет нам лучше изучить операции рисования.

Итак, создайте новый документ Flash. И привяжите к первому кадру фильма следующий сценарий (мы рассмотрим его построчно):

```
dx = 400;
```

Это выражение задает величину отступа от левого края листа. Именно в данной точке будет начало координат нашего графика.

```
x = y = 0;
```

Это временные переменные.

```
lineStyle(1, 0x000000);  
for(x = -20; x < 21; x++) {  
    y = x * x;  
    beginFill(0x000000);  
    moveTo(dx + x - 1, y - 1);  
    lineTo(dx + x + 1, y - 1);  
    lineTo(dx + x + 1, y + 1);  
    lineTo(dx + x - 1, y + 1);  
    lineTo(dx + x - 1, y - 1);  
    endFill();  
}
```

А этот код служит, собственно, для рисования графика.

Мы использовали цикл со счетчиком, чтобы вычислить значения функции для аргумента в пределах от -20 до 20. Каждая точка, рисуемая нами на листе, представляет собой небольшой квадрат, заполненный черной заливкой. Вот и все!

Правда, график наш рисуется вверх ногами. Чтобы исправить этот огрех, добавим еще одну переменную:

```
dy = 400;
```

задающую смещение начала координат графика по вертикали, и немного изменим код нашего цикла:

```
for(x = -20; x < 21; x++) {  
    y = x * x;  
    beginFill(0x000000);  
    moveTo(dx + x - 1, dy - y + 1);  
    lineTo(dx + x + 1, dy - y + 1);  
    lineTo(dx + x + 1, dy - y - 1);
```

```
lineTo(dx + x - 1, dy - y - 1);  
lineTo(dx + x - 1, dy - y + 1);  
endFill();  
}
```

Теперь наш график рисуется нормально.

Работа с масками

Чтобы преобразовать встроенный клип в маску, используйте метод `setMask`:

```
<Маскируемый клип>.setMask(<Клип-маска>);
```

Не забудьте только, что и маска, и маскируемое изображение (то есть, маскируемый клип) должны находиться в разных слоях.

Вы можете проверить действие этого метода. Создайте новый документ Flash, содержащий два слоя, поместите в нижний слой клип, содержащий любое растровое изображение, а в верхний — экземпляр образца-клипа, с любой простой геометрической фигурой. Назовите растровое изображение `masked`, а клип — `mask`. После этого присвойте первому (и единственному) кадру фильма следующий небольшой сценарий:

```
masked.setMask(mask);  
mask.startDrag(true);
```

Первое выражение этого сценария присваивает растровому изображению `masked` маску — клип `mask`. Второе выражение заставляет клип `mask` (то есть, маску) перемещаться за мышью.

Теперь запустите созданный фильм на проигрывание. "Захватите" клип-маску мышью и попробуйте перемещать его с места на место. Выглядит это весьма забавно.

Чтобы убрать маску, выполните метод `setMask` с параметром `null`:

```
masked.setMask(null);
```

Не забывайте также, что вы можете управлять маской из сценария. Можно, например перемещать ее или изменять размеры.

Объект Key

Объект `Key` служит для обработки нажатия клавиш клавиатуры. Этот объект создается самим Flash, его единственный экземпляр носит имя `Key`.

Свойства и методы объекта Key

Два важнейших метода, которые обязательно вам понадобятся, если вы захотите в своем сценарии обрабатывать нажатия клавиш, — это `getAscii` и `getCode`. Первый метод возвращает код символа ASCII, который присвоен

этой клавише. Второй метод возвращает так называемый *виртуальный код клавиши*, низкоуровневый код, позволяющий узнать, какая клавиша на клавиатуре была нажата. Ни один из этих методов не принимает параметров.

```
code1 = Key.getAscii();
code2 = Key.getCode();
```

Код символа ASCII зависит от текущей кодировки. Коды символов кодировки Windows 1251 приведены в *приложении 1*. Там же приведены и виртуальные коды клавиш.

Еще один полезнейший метод — `isDown`. Он принимает единственный параметр — виртуальный код клавиши — и возвращает `true`, если эта клавиша нажата.

```
if (Key.isDown(Key.ENTER)) { . . . }
```

Приведенное выше выражение проверяет, была ли нажата клавиша <Enter>.

Метод `isToggled` возвращает `true`, если была включена одна из клавиш-переключателей: <NumLock> или <CapsLock>. Он принимает единственный параметр — числовое значение 20 (виртуальный код клавиши <CapsLock>) или 144 (<NumLock>).

```
if (Key.isToggled(20)) { s = s.toLowerCase; }
```

Кроме вышеперечисленных методов, объект `Key` поддерживает ряд свойств, доступных только для чтения и возвращающих значения виртуальных кодов различных *служебных клавиш*. Все эти свойства перечислены в табл. 22.3.

Таблица 22.3. Свойства объекта `Key`, возвращающие коды служебных клавиш

Свойство	Клавиша	Виртуальный код клавиши
BACKSPACE	<BackSpace>	8
CAPSLock	<CapsLock>	20
CONTROL	<Ctrl>	17
DELETEKEY		46
DOWN	<Стрелка вниз>	40
END	<End>	35
ENTER	<Enter>	13
ESCAPE	<Esc>	27
HOME	<Home>	36
INSERT	<Ins>	45
LEFT	<Стрелка влево>	37
PGDN	<PgDn>	34

Таблица 22.3 (окончание)

Свойство	Клавиша	Виртуальный код клавиши
PGUP	<PgUp>	33
RIGHT	<Стрелка вправо>	39
SHIFT	<Shift>	16
SPACE	<Пробел>	32
TAB	<Tab>	9
UP	<Стрелка вверх>	38

Обработка нажатий клавиш

Рассмотрим простейший пример интерактивного фильма, обрабатывающего нажатия клавиш клавиатуры.

Создадим новый документ Flash. В нем создадим образец-клип, содержащий простую геометрическую фигуру, например, круг. Поместим экземпляр этого образца на рабочий лист и назовем его `circle`. Этот экземпляр мы заставим перемещаться по листу в ответ на нажатия клавиш-стрелок.

Теперь выделим наш экземпляр и привяжем к нему вот такой, довольно сложный сценарий. Для удобства рассмотрения разобьем его на две части.

```
onClipEvent(load) {  
    d = 1;  
}
```

Обработчик события `load` помещает значение в переменную `d` сразу после загрузки клипа. Эта переменная задает величину перемещения нашего экземпляра при нажатии любой клавиши-стрелки. Вы можете увеличить это число, чтобы заставить клип двигаться быстрее.

```
onClipEvent(keyDown) {  
    switch (Key.getCode()) {  
        case Key.UP :  
            _root.circle._y -= d;  
            break;  
        case Key.DOWN :  
            _root.circle._y += d;  
            break;  
        case Key.LEFT :  
            _root.circle._x -= d;  
            break;
```

```
case Key.RIGHT :  
    _root.circle._x += d;  
    break;  
}  
}
```

А этот обработчик события `keyDown`, несмотря на его кажущуюся сложность, весьма прост. Он сравнивает значение виртуального кода нажатой клавиши с кодами клавиш-стрелок и, если одна из них нажата, изменяет соответствующую координату клипа `circle`.

Теперь проверим созданный фильм. Как видите, все работает!

Конечно, этот пример очень прост. Но, руководствуясь им, вы можете создавать и несравнимо более сложные вещи. Например, как насчет игры "15", реализованной на Flash? Или хорошо вам известного "Сапера"?

Использование объектов-перехватчиков

Только что мы рассмотрели первый способ обработки нажатий клавиш — использование обработчика события, привязанного к клипу. Но Flash предоставляет вам еще один способ сделать это, а именно — использовать *объект-перехватчик*, особый объект, который вы создаете сами и привязываете к объекту `Key`. (В терминологии Flash объект-перехватчик называется "listener", то есть, "слушатель".) Этот второй способ сложнее первого, но имеет ряд преимуществ, которые мы здесь рассмотрим.

Что такое объект-перехватчик? Ничего сложного, это просто пользовательский объект, который должен удовлетворять особым условиям. А именно: иметь два метода, `onKeyUp` и `onKeyDown`. При нажатии любой клавиши на клавиатуре вызывается метод `onKeyDown`, при ее отпускании — метод `onKeyUp`.

Давайте создадим экземпляр простейшего объекта-перехватчика (предположим, что функции обработки нажатия клавиш `fOnKeyDown` и `fOnKeyUp` уже объявлены). Проще всего создать его как экземпляр объекта `Object`:

```
myListener = new Object();  
myListener.onKeyDown = fOnKeyDown;  
myListener.onKeyUp = fOnKeyUp;
```

Для привязки экземпляра объекта-перехватчика используем метод `addListener` объекта `Key`. Этот метод принимает единственный параметр — экземпляр объекта-перехватчика.

```
Key.addListener(myListener);
```

Чтобы удалить ненужный объект-перехватчик, воспользуемся методом `removeListener`:

```
Key.removeListener(myListener);
```

Всегда удаляйте ненужные объекты-перехватчики, чтобы освободить оперативную память компьютера для более полезных вещей. Впрочем, если объект-перехватчик будет использоваться все время, пока проигрывается фильм, вы можете его и не удалять сами — это сделает проигрыватель Flash, когда завершит воспроизведение фильма.

Какие же преимущества сулит использование объектов-перехватчиков? Их два, и мы их сейчас рассмотрим.

- ❑ Обработка всех нажатий клавиш "в одном месте", а именно, в одной или двух функциях-методах объекта. Иногда без этого не обойтись.
- ❑ Возможность создания нескольких объектов-перехватчиков, работающих независимо друг от друга. Это может понадобиться для создания сложных Flash-приложений, например, игр.

Напоследок рассмотрим пример применения объектов-перехватчиков. Ранее мы создали фильм, реагирующий на нажатие клавиш: в ответ на них по листу перемещался небольшой клип. Теперь перепишем этот пример с использованием объектов-перехватчиков.

```
onClipEvent(load) {  
    function fOnKeyDown() {  
        switch (Key.getCode()) {  
            case Key.UP :  
                _root.circle._y -= d;  
                break;  
            case Key.DOWN :  
                _root.circle._y += d;  
                break;  
            case Key.LEFT :  
                _root.circle._x -= d;  
                break;  
            case Key.RIGHT :  
                _root.circle._x += d;  
                break;  
        }  
    }  
    d = 1;  
    myListener = new Object();  
    myListener.onKeyDown = fOnKeyDown;  
    Key.addListener(myListener);  
}  
onClipEvent(unload) {
```

```
Key.removeListener(myListener);  
}
```

Попробуйте разобраться в этом сценарии сами. Единственное замечание: нам пришлось поместить объявление функции `fOnKeyDown` в обработчик события, т. к. сценарий, привязанный к клипу, может содержать только обработчики событий.

Объект *Mouse*

Объект `Mouse` служит для доступа к некоторым свойствам мыши. Единственный экземпляр этого объекта под именем `Mouse` создается самим Flash.

Свойств у объекта `Mouse` нет совсем. Да и методов немного.

Прежде всего, этот объект предоставляет методы `hide` и `show`. Первый метод скрывает курсор мыши, а второй — вновь выводит его на экран. Ни один из них не принимает параметров. Используйте эти методы, если вы хотите создать свой собственный курсор мыши. (Как это сделать, было описано ранее в этой главе.)

Объект `Mouse` поддерживает использование объектов-перехватчиков. Для добавления и удаления их служат хорошо вам знакомые методы `addListener` и `removeListener`. Объект-перехватчик должен иметь следующие методы:

- ❑ `onMouseDown` вызывается при нажатии левой кнопки мыши;
- ❑ `onMouseUp` вызывается при отпускании левой кнопки мыши;
- ❑ `onMouseMove` вызывается при любом перемещении мыши.

Объект *Color*

Объект `Color` служит для представления цвета какого-либо встроенного клипа или самого фильма. Также с его помощью вы можете задавать различные преобразования цвета этого клипа (фильма).

Экземпляр объекта `Color` создается с помощью конструктора, имеющего следующий формат:

```
<переменная> = new Color(<Клип>);
```

Например:

```
circleColor = new Color(circle);
```

Метод `getRGB` возвращает цвет клипа в формате RGB. А метод `setRGB` позволяет его задать.

```
color1 = circleColor.getRGB();  
carColor.setRGB(0x00FF00);
```

С помощью второго выражения мы "покрасили" наш многострадальный автомобиль в зеленый цвет.

Чтобы задать для цвета какие-либо преобразования, вам нужно воспользоваться вспомогательным объектом. Он представляет собой обычный экземпляр объекта `Object`, имеющий, однако, некоторые особые свойства.

```
myTransform = new Object();
```

Теперь вспомогательному объекту следует присвоить несколько свойств, значения которых и зададут необходимое преобразование цвета. Все эти свойства перечислены в табл. 22.4.

Таблица 22.4. Свойства вспомогательного объекта цветовых преобразований

Свойство	Описание
ra	Процентное изменение красной составляющей, от -100 до 100
rb	Изменение красной составляющей, от -255 до 255
ga	Процентное изменение зеленой составляющей, от -100 до 100
gb	Изменение зеленой составляющей, от -255 до 255
ba	Процентное изменение синей составляющей, от -100 до 100
bb	Изменение синей составляющей, от -255 до 255
aa	Процентное изменение прозрачности, от -100 до 100
ab	Изменение прозрачности, от -255 до 255

Пользоваться вспомогательным объектом очень просто. Предположим, что нам нужно убавить интенсивность синей составляющей цвета наполовину и добавить 20 единиц красной составляющей. Для этого напишем два следующих выражения:

```
myTransform.ba = -50;
myTransform.rb = 20;
```

Если вы обратитесь к *главе 10* и посмотрите на рис. 10.8, то увидите, что вышеперечисленные свойства аналогичны полям ввода диалогового окна **Advanced Effect**.

Чтобы применить заданные вами цветовые преобразования, воспользуйтесь методом `setTransform`:

```
circleColor.setTransform(myTransform);
```

Как видите, этот метод принимает единственный параметр — ссылку на экземпляр вспомогательного объекта. А метод `getTransform` возвращает ссылку на сам экземпляр вспомогательного объекта:

```
othTransform = circleColor.getTransform();
```

Вы можете использовать этот метод, чтобы получить вспомогательный объект, прочитать значения его свойств, изменить их и снова присвоить цвету:

```
othTransform.aa /= 2;  
othTransform.ab = 10;  
circleColor.setTransform(othTransform);
```

Объект *Sound*

Объект *Sound* служит для управления проигрыванием звука. Вы можете использовать экземпляры этого объекта для управления звуковым сопровождением отдельного клипа или всего фильма.

Экземпляры объекта *Sound* создаются с помощью конструктора, имеющего следующий формат:

```
<переменная> = new Sound([<Клип>]);
```

Единственный параметр конструктора задает клип, звуковым сопровождением которого вы хотите управлять. Если параметр не указан, создается экземпляр объекта, управляющий звуковым сопровождением всего фильма.

```
mySound = new Sound(car);  
globalSound = new Sound();
```

Свойства и методы объекта *Sound*

Объект *Sound* имеет много свойств и методов. Некоторые из них мы здесь рассмотрим.

Метод *start* позволит вам запустить проигрывание звука. Он имеет следующий формат:

```
<Экземпляр>.start([<Отметка>, <Количество повторов>]);
```

Этот метод может принимать два необязательных параметра. Первый параметр определяет, с какой отметки будет проигран звук; отметка задается как количество секунд, которые Flash отсчитает с момента начала звука. Второй параметр определяет количество повторов звука. Если ни один параметр не задан, звук будет проигран один раз с начала.

```
mySound.start(10, 2);
```

Метод *stop* немедленно останавливает проигрывание звука. Формат его вызова таков:

```
<Экземпляр>.stop([<Имя звука>]);
```

В качестве единственного параметра этого метода может быть передано имя звука, который нужно остановить, оно передается в строковом виде. Если метод был вызван без параметра, останавливается проигрывание всех звуков.

```
mySound.stop("beep");
```

Не принимающий параметров метод `getVolume` возвращает значение громкости звука. А метод `setVolume` позволяет задать новое значение громкости звука, переданное в качестве его единственного параметра. Доступны значения от 0 (звук не слышен) до 100 (полная громкость; значение по умолчанию).

```
mySound.setVolume(mySound.getVolume() + 10);
```

Пара методов `getPan` и `setPan` ведут себя так же и позволяют задать панорамирование звука в пространстве. Здесь доступны значения от -100 (звук в левом канале) до 100 (звук в правом канале); значение по умолчанию — 0 (звук посередине).

```
mySound.setPan(50);
```

Более сложные преобразования над звуком вы также можете выполнять при помощи вспомогательного объекта. Он представляет собой обычный экземпляр объекта `Object`, но с особыми свойствами, которые перечислены в табл. 22.5.

Таблица 22.5. Свойства вспомогательного объекта преобразований звука

Свойство	Описание
ll	Задаёт, какая часть "содержимого" левого канала будет проигрываться в левом канале в процентах, от 0 до 100
lr	Задаёт, какая часть "содержимого" левого канала будет проигрываться в правом канале в процентах, от 0 до 100
rl	Задаёт, какая часть "содержимого" правого канала будет проигрываться в левом канале в процентах, от 0 до 100
rr	Задаёт, какая часть "содержимого" правого канала будет проигрываться в правом канале в процентах, от 0 до 100

Несколько примеров использования вспомогательного объекта звуковых преобразований:

```
myTransform = new Object();  
with (myTransform) {  
    ll = 100;  
    lr = 0;  
    rl = 0;  
    rr = 100;  
}
```

После выполнения этого сценария звук будет проигрываться как обычно: "содержимое" левого канала — в левом канале, "содержимое" правого — в правом.

```
with (myTransform) {  
    ll = 0;  
    lr = 100;  
    rl = 100;  
    rr = 0;  
}
```

Этот сценарий поменяет каналы местами.

```
with (myTransform) {  
    ll = 50;  
    lr = 50;  
    rl = 50;  
    rr = 50;  
}
```

А этот сценарий сделает звук монофоническим.

Для применения звуковых преобразований выполните метод `setTransform`:

```
mySound.setTransform(myTransform);
```

А метод `getTransform` позволит вам получить ссылку на экземпляр вспомогательного объекта.

Объект `Sound` поддерживает два свойства, доступных только для чтения. Свойство `position` возвращает текущую проигрываемую позицию звука в миллисекундах. А свойство `duration` возвращает продолжительность звука также в миллисекундах.

Создание органов управления звуком

Пользуясь описанными выше методами объекта `Sound`, вы можете создавать в своих фильмах и приложениях Flash органы управления громкостью и панорамированием звука. Для этого вам понадобится совсем немного терпения и всего лишь несколько строк на языке `ActionScript`.

Рассмотрим небольшой пример создания органов управления звуком. Пусть это будут знакомые вам по приложениям Windows движковые регуляторы; один — вертикальный — будет управлять громкостью, а другой — горизонтальный — панорамированием.

Сначала выберем, что мы будем проигрывать. У автора нашелся на диске аудиофайл с композицией "Stargazers" группы "Nightwish". Если же вы не любите финский *progressive metal*, поищите что-нибудь другое.

Импортируем звуковой файл во Flash. Открываем окно библиотеки и перетаскиваем импортированный звук на рабочий лист. Теперь растянем единственный кадр делений на 500 (а лучше — больше) шкалы, чтобы то, что мы делаем, действительно можно было слушать. Проиграем полученный резуль-

тат, нажав клавишу <Enter>, чтобы удостовериться, что все идет нормально, после чего остановим фильм и переместим указатель на первый кадр. И приступаем к созданию органов управления.

Начнем с регулятора громкости. Нарисуем на рабочем листе небольшую шкалу. После этого создадим маленький клип, представляющий собой круг, — это будет ручка регулятора. Проверим, находится ли точка фиксации образца в центре круга, иначе ручку не удастся "прицепить" к шкале точно за центр. Поместим экземпляр этого клипа на рабочий лист и назовем его `volume`.

В результате этих действий у вас должно получиться что-то, похожее на рис. 22.3. Заметьте вертикальные координаты верхнего и нижнего делений этой шкалы — они нам потом пригодятся. Предположим, у вас они равны соответственно 95 и 247.

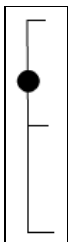


Рис. 22.3. Шкала громкости с регулятором

Теперь выделим первый кадр фильма и присвоим ему совсем небольшой сценарий:

```
clipSound = new Sound();
```

Он создает экземпляр объекта `Sound` для управления звуковым сопровождением фильма.

Затем выделите ручку регулятора и присвойте ей целых три обработчика. Мы рассмотрим их последовательно.

```
on(press) {  
    this.startDrag(false, this._x, 95, this._x, 247);  
}
```

Первый обработчик заставляет ручку регулятора перемещаться вслед за курсором мыши. Обратите внимание на параметры метода `startDrag`: мы задаем пределы перетаскивания клипа, равные нарисованной нами шкале громкости. Если же этого не сделать, то пользователь сможет "вытащить" ручку за пределы шкалы и вдоволь позубоскалить над горе-программистами.

```
on(release) {  
    this.stopDrag();  
}
```

Второй обработчик оставляет ручку регулятора на месте, если пользователь отпустит кнопку мыши.

```
onClipEvent(enterFrame) {
    _root.clipSound.setVolume((247 - this._y) / (247 - 95) * 100);
}
```

А третий обработчик осуществляет изменение громкости звука в зависимости от того, где в данный момент находится ручка регулятора. Конечно, можно поместить этот код в обработчик события `release`, но тогда громкость звука будет изменяться после "отпускания" ручки регулятора. А сейчас громкость изменяется прямо во время перетаскивания ручки.

Обратите особое внимание на выражение, вычисляющее значение параметра метода `setVolume`. Попробуйте самостоятельно в нем разобраться (и держите перед глазами рис. 22.3).

```
onClipEvent(load) {
    this._y = 95;
}
```

Этот обработчик задает сразу же после загрузки ручки ее изначальную позицию — на верхней метке шкалы.

Попробуйте запустить фильм на проигрывание. Потаскайте ручку по шкале и послушайте, что делается с вашим звуком. Работает, не правда ли?

Теперь добавим регулятор панорамирования. Создадим точно такую же шкалу, но горизонтальную. Поместим на нее ручку — экземпляр уже созданного нами образца-клипа — и назовем ее `panning`. То, что должно у нас получиться, показано на рис. 22.4.

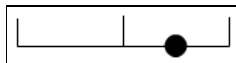


Рис. 22.4. Шкала панорамирования с регулятором

Запомним горизонтальные координаты левого и правого делений этой шкалы. Предположим, что они равны соответственно 258 и 408.

Экземпляр объекта `Sound`, управляющий звуком всего фильма, у нас уже есть. Осталось добавить к ручке регулятора `panning` обработчики событий нажатия и отпускания левой кнопки мыши. Рассмотрим их по отдельности.

```
on(press) {
    this.startDrag(false, 258, this._y, 408, this._y);
}
```

Этот код разрешает ручке перемещаться за курсором мыши. Мы также ограничиваем область ее перемещения, чтобы пользователь не мог "вытащить" ручку за пределы шкалы.

```
on(release) {  
    this.stopDrag();  
}
```

Этот уже знакомый вам обработчик оставляет ручку регулятора на месте, если пользователь отпустит кнопку мыши.

```
onClipEvent(enterFrame) {  
    _root.clipSound.setPan(this._x - (408 - 258) / 2);  
}
```

А этот обработчик реализует собственно изменение панорамирования звука. Обратите внимание на выражение, служащее для вычисления значения этого панорамирования — пожалуй, в таких сценариях это самая сложная часть.

```
onClipEvent(load) {  
    this._x = 258 + (408 - 258) / 2;  
}
```

Осталась мелочь. А именно: в самом начале поместить ручку регулятора в среднее положение шкалы, туда, где она должна находиться по умолчанию.

Теперь наш пример завершен. Попробуйте его запустить. Конечно, он сделан на скорую руку, но ведь не все сразу.

Загрузка и выгрузка звуков

Вы уже знаете, как можно загружать и проигрывать внешние клипы. То же самое можно проделывать со звуками. Имейте только в виду, что известное вам ограничение на загрузку клипов из того же поддомена, из которого был загружен основной фильм, действует и на звуки.

Для загрузки внешнего звука вам следует воспользоваться методом `loadSound` объекта `Sound`:

```
<Звук>.loadSound(<Интернет-адрес или путь звукового файла>,  
    <Потоковый звук>);
```

Как видите, этот метод принимает два параметра. Первый параметр задает интернет-адрес звукового файла или его путь на локальном диске компьютера в строковом виде. Второй параметр — логический — задает, будет ли загружаемый звук потоковым или звуком-сигналом. (О потоковых звуках и сигналах см. главу 17.) Чтобы сделать звук потоковым, передайте в качестве второго параметра `true`.

Ниже приведен пример использования данного метода:

```
newSound = new Sound();  
newSound.loadSound("Nightwish_(Stargazers).mp3", true);  
newSound.play();
```

Так мы можем добраться до (не)любимых "Nightwish". Обратите внимание, что сначала мы создали экземпляр объекта `Sound`, а уже потом загрузили в него звук. И не забудьте запустить его на проигрывание методом `play`.

Вы также можете создать новый экземпляр, основанный на образце-звуче, который был создан ранее и помещен в библиотеку фильма. Для этого объект `Sound` предусматривает метод `attachSound`. Формат его вызова таков:

```
<Звук>.attachSound("<Имя разделяемого образца>");
```

Не забудьте только, что образец-звук должен быть сделан сценарным.

```
newSound2 = new Sound();
```

```
newSound2.attachSound("Star_industry_(Nineties).mp3");
```

```
newSound2.play();
```

А как насчет английского готик-рока?

Объект *Stage*

Объект `Stage` служит для доступа к свойствам окна проигрывателя Flash и некоторым методам для управления им. При этом не важно, запущен ли проигрыватель Flash как отдельная программа или работает в окне другой программы, например, Web-обозревателя.

Единственный экземпляр этого объекта под именем `Stage` создается самим Flash. Следовательно, вам создавать его не нужно.

Прежде всего, объект `Stage` поддерживает доступные только для чтения свойства `height` и `width`. Они возвращают соответственно высоту и ширину окна проигрывателя в пикселах. Вы можете использовать их значения в сценариях.

Свойство `align` позволит вам задать выравнивание фильма в окне проигрывателя. Все поддерживаемые этим свойством значения перечислены в табл. 22.6. Они должны иметь строковый тип.

Таблица 22.6. Значения, поддерживаемые свойством `align` объекта `Stage`

Значение	Выравнивание	
	по горизонтали	по вертикали
T	Верхнее	По центру
B	Нижнее	По центру
L	По центру	Левое
R	По центру	Правое
TL	Верхнее	Левое

Таблица 22.6 (окончание)

Значение	Выравнивание	
	по горизонтали	по вертикали
TR	Верхнее	Правое
BL	Нижнее	Левое
BR	Нижнее	Правое

С помощью свойства `scaleMode` вы можете задать параметры масштабирования изображения Flash, если оно не помещается в окно проигрывателя. Это свойство имеет строковый тип и принимает следующие значения:

- ❑ `"showAll"` — будет показано все изображение, для чего может быть применено масштабирование. Однако пропорции изображения искажены не будут, в результате этого вдоль горизонтальных или вертикальных сторон его могут появиться границы;
- ❑ `"noBorder"` — то же самое, что `"showAll"`, но границы появляться не будут, — проигрыватель Flash обрежет изображение по горизонтали или вертикали, чтобы их избежать;
- ❑ `"exactFit"` — будет показано все изображение, для чего может быть применено масштабирование, в результате которого могут исказиться размеры изображения;
- ❑ `"noScale"` — изображение ни в каком случае не будет масштабироваться, в результате чего может оказаться обрезанным.

Объект `Stage` поддерживает использование объектов-перехватчиков. Объект-перехватчик должен иметь метод `onResize`, который будет вызван при изменении размеров окна проигрывателя Flash. Таким образом, вы можете создавать фильмы, реагирующие на изменение размеров окна проигрывателя.

Объект *System.capabilities*

Объект `System.capabilities` служит для получения параметров компьютерной платформы, на которой работает проигрыватель Flash. Здесь `System` — внешний объект, единственный экземпляр которого под именем `System` создается самим Flash, а `capabilities` — вложенный в него объект.

Объект `System.capabilities` имеет много свойств и ни одного метода. Это значит, что вы можете только прочитать параметры клиентского компьютера, но не изменить их. Так, вы не сможете добавить к нему звуковую карту или сменить разрешение экрана — увы, это не в силах простого программиста.

Здесь мы рассмотрим только немногие свойства этого объекта.

Свойство `hasAudio` возвращает `true`, если компьютер клиента имеет звуковые возможности.

```
if (System.capabilities.hasAudio) {  
    _root.Sound.play();  
}
```

Свойство `hasMP3` возвращает `true`, если компьютер клиента имеет установленный кодек формата MP3.

```
if (System.capabilities.hasMP3) {  
    _root.mp3Sound.play();  
} else {  
    _root.othSound.play();  
}
```

Свойство `os` возвращает название операционной системы вида "Windows 2000".

Свойство `version` возвращает номер версии проигрывателя Flash, используемого клиентом. По умолчанию возвращается 6.0.

Свойства `screenResolution.x` и `screenResolution.y` позволят вам узнать разрешение экрана клиентского компьютера. Первое свойство возвращает разрешение экрана по горизонтали, а второе — по вертикали. (Обратите внимание, что здесь также имеет место вложенный объект `screenResolution`.)

Использование таймеров

Очень часто во многих приложениях нужно выполнять какой-либо код через равные промежутки времени. Это может понадобиться, например, в игровых программах реального времени или для создания анимации с помощью сценариев `ActionScript`. Также это может быть использовано для обновления информации, взятой с удаленного сервера. Для таких задач во Flash предусмотрены так называемые *таймеры*.

Таймер создается с помощью действия `setInterval`. В качестве параметров этого действия передаются функция или метод объекта, которые будут вызываться по истечении интервала времени, и само значение этого интервала. Действие `setInterval` возвращает особый *идентификатор* таймера, который в дальнейшем используется для его уничтожения.

Действие `setInterval` имеет два формата вызова, приведенные ниже.

<Переменная>=`setInterval` (<Функция>, <Значение интервала>

`[, <Список параметров функции, разделенных запятыми>]);`

```
<Переменная>=setInterval(<Экземпляр объекта>, <Метод объекта>,  
    <Значение интервала>  
    [, <Список параметров метода, разделенных запятыми>]);
```

Как видите, оба этих формата отличаются ненамного. Первый формат задает в качестве вызываемого кода функцию, а второй — метод объекта. Величина интервала задается в миллисекундах. Также может присутствовать список параметров вызываемой функции или метода, разделенных запятыми. Он нужен, если функция или метод принимает параметры.

Разумеется, функция или метод должны быть объявлены прежде, чем будут использованы.

```
timerID1 = setInterval(tick, 1000);  
timerID2 = setInterval(car, moveBy, 100, d);
```

Выше приведены два примера вызова действия `setInterval`. Первый пример создает таймер, вызывающий функцию `tick` каждую секунду (1000 миллисекунд). (Подразумевается, что функция `tick` уже объявлена.) Второй пример создает таймер, вызывающий каждые 100 миллисекунд метод `moveBy` экземпляра объекта `car` и передающий ему параметр `d`.

Когда будете использовать таймеры, имейте в виду следующее. Дело в том, что Flash отмеряет интервалы не очень точно. Он пытается привязать каждый интервал к частоте кадров фильма, если, конечно, это возможно. Если интервал меньше частоты кадров, то он обрабатывается с максимальной точностью, и код вызывается вовремя. Если же интервал больше частоты кадров, то код, связанный с таймером, выполняется только тогда, когда Flash воспроизводит очередной кадр фильма. Это сделано для того, чтобы лишний раз не перерисовывать экран, если код что-то изменит в изображении.

Чтобы обновить экран после выполнения связанного с таймером кода, вы можете использовать действие `updateAfterEvent`. Вызывайте его сразу же после выполнения всех выражений, затрагивающих графику, например, изменения местоположения клипа или его цвета.

После того, как нужда в таймере отпадет, его следует уничтожить, чтобы освободить системные ресурсы. Для этого используйте действие `clearInterval`:

```
clearInterval(<Идентификатор таймера>);
```

Например:

```
clearInterval(timerID2);
```

Теперь рассмотрим пример, показывающий, как можно использовать таймер для создания "ленивого" курсора мыши. Обычный курсор мыши нигде надолго не задерживается. Как только вы переместите мышь, он сразу же спешит за ней. "Ленивый" же курсор отличается от обычного тем, что никогда не торопится за мышью. Давайте же сделаем такой курсор.

Прежде всего, создайте новый образец-клип и нарисуйте в нем желаемый курсор. Постарайтесь нарисовать его так, чтобы точка фиксации пришлась как раз на "острие" этого курсора. Поместите экземпляр этого клипа на рабочий лист и назовите его `cursor`.

Теперь напишем код сценария, который будет привязан к первому кадру фильма:

```
function tick() {  
  if (_root.cursor._x != _root._xmouse) {  
    if (_root.cursor._x < _root._xmouse) {  
      if (_root._xmouse - _root.cursor._x < d) {  
        _root.cursor._x = _root._xmouse;  
      } else {  
        _root.cursor._x += d;  
      }  
    } else {  
      if (_root.cursor._x - _root._xmouse < d) {  
        _root.cursor._x = _root._xmouse;  
      } else {  
        _root.cursor._x -= d;  
      }  
    }  
  }  
  
  if (_root.cursor._y != _root._ymouse) {  
    if (_root.cursor._y < _root._ymouse) {  
      if (_root._ymouse - _root.cursor._y < d) {  
        _root.cursor._y = _root._ymouse;  
      } else {  
        _root.cursor._y += d;  
      }  
    } else {  
      if (_root.cursor._y - _root._ymouse < d) {  
        _root.cursor._y = _root._ymouse;  
      } else {  
        _root.cursor._y -= d;  
      }  
    }  
  }  
  
  updateAfterEvent();  
}
```

Эта функция реализует перемещение "ленивого" курсора. В двух словах, она работает следующим образом. Сравниваются координаты, горизонтальная и вертикальная, курсора и мыши и выполняется приращение координат курсора. Заметьте, что координаты мыши измеряются относительно основного фильма.

```
d = 25;
```

Это величина приращения, на которое будут изменяться координаты курсора.

```
ctID = setInterval(tick, 50);
```

Здесь мы создаем таймер, вызывающий функцию `tick` каждые 50 миллисекунд.

```
_root.onUnload = function() {  
    clearInterval(ctID);  
}
```

А здесь мы привязываем к событию `unload` клипа `cursor` обработчик, удаляющий созданный ранее таймер. Удалять за собой затребованные для своих нужд ресурсы, те же таймеры, — хороший стиль программирования.

Если уж зашла речь о таймерах, то вам может пригодиться функция `getTimer`. Она возвращает количество миллисекунд, прошедших с момента, когда фильм начал проигрываться.

Создание библиотек кода

Хороший программист всегда стремится сохранить свои наработки для дальнейшего использования. Он никогда не будет делать одну и ту же работу несколько раз, во всяком случае, постарается не делать. Он имеет свой набор удачных сценариев, которые и применяет в своих разработках. Это справедливо для любого языка программирования, в том числе, для `ActionScript`.

Здесь мы рассмотрим создание таких наборов сценариев, иначе говоря, *библиотек кода* (не путать с библиотекой образцов!).

Библиотека кода — это обычный сценарий, содержащий, как правило, неизменяемый (но, возможно, пополняемый) набор объектов и функций подходящих для различных фильмов и приложений. Вместо того чтобы писать во всех фильмах одни и те же объявления одних и тех же функций и объектов (хотя это тоже вполне допустимо), программист просто копирует в фильм всю библиотеку и без проблем вызывает из нее нужные ему функции и объекты.

Самое лучшее место для размещения библиотеки кода — "пустой" образец клип. Такой образец можно поместить в одну из стандартных библиотек

Flash (о стандартных библиотеках см. *главу 10*). Библиотеки кода могут использоваться не только их создателем, но и его коллегами, и, в принципе, ничто не мешает отдавать или даже продавать их на сторону.

Как создать библиотеку кода во Flash? Очень просто. Прежде всего, отберите самые удачные ваши функции и объявления объектов и поместите их во внешний файл. (Конечно, это не обязательно, но, согласитесь, проще взять и скопировать все "хором", чем рыскать по множеству документов Flash в поисках нужного куска кода.) После этого создайте "пустой" образец-клип и привяжите сохраненный код к его первому кадру.

Здесь нужно сказать кое-что об объявлениях функций. Так как они будут использованы как в основном фильме, так и во встроенных клипах, то должны быть глобальными. Поэтому при их объявлении должен использоваться модификатор `_global`.

```
function _global.doSomething(. . .) {
```

Можно также сделать немного по-другому:

```
_global.doSomething = function(. . .) {
```

После этого объявленную функцию можно вызывать откуда угодно.

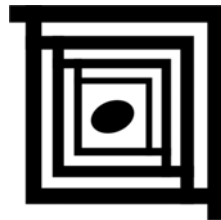
Объекты объявляются точно так же. Вы ведь помните, что объявление объекта — фактически объявление его конструктора.

```
function _global.someObject(. . .) {
```

```
_global.someObject = function(. . .) {
```

Ну и, конечно же, вы можете сделать библиотеку кода разделяемым образцом. Тогда ее сможет использовать кто угодно, где угодно. Как это сделать, было описано в *главе 10*.

Глава 23



Создание приложений Flash

Знаете ли вы, что отличает простой интерактивный фильм от полноценного приложения Flash? Правильно, наличие пользовательского интерфейса!

Пользовательским интерфейсом называется совокупность средств, предназначенных для организации общения пользователя и программы. В случае хорошо нам знакомой операционной системы Windows, пользовательский интерфейс — это набор окон и элементов управления, которые служат для ввода данных в программу и получения от нее результатов. Любое приложение, созданное во Flash или в другом средстве разработки, должно иметь пользовательский интерфейс, иначе оно не сможет получать и выводить данные. Ведь главное предназначение любой программы — обработка данных. А фильм ну что фильм — крутится себе на экране, независимо, смотрят его или нет.

Поэтому, если вы собираетесь делать на Flash нечто большее, чем фильмы с ограниченными возможностями "общения" со зрителем (чем мы занимались в *главе 22*), вам обязательно нужно позаботиться о пользовательском интерфейсе. И не просто слепить пару элементов управления и связать их парой строк кода — вам придется сделать его удобным для пользователя. Иначе пользователь предпочтет вашей программе другую, благо сейчас у него есть широчайшие возможности выбора.

Итак, какие же средства предоставляет Flash создателям приложений? Давайте их перечислим и кратко рассмотрим.

- ❑ *Кнопки.* Это экземпляры образцов-кнопок, впервые упомянутые в *главе 10*. Образцы-кнопки не очень отличаются от образцов-клипов, за несколькими исключениями, обусловленными самой их "интерфейсной" природой. Кнопки позволяют привязать обработчики к событию `press` (щелчок на кнопке) и некоторым другим.
- ❑ *Поля ввода и динамические текстовые блоки.* Первые позволяют принять какие-либо данные от пользователя, а вторые — вывести результаты. Создание полей ввода и динамических текстовых блоков было описано в *главе 7*. Запомните, что обычные, статические текстовые блоки не могут управляться из сценариев.

- ❑ *Полноценные элементы управления.* Собственно, это обычные клипы, созданные самими разработчиками Flash и включенные в его состав. Среди элементов управления вы можете найти флажки, переключатели, списки и пр., что вам привычно по интерфейсу Windows. Но, в отличие от элементов управления Windows, элементы управления Flash могут менять свой внешний вид. Как это сделать, мы расскажем в конце этой главы.
- ❑ *Пользовательские элементы управления.* Это новые элементы управления, не встроенные во Flash, а разработанные вами. Для создания пользовательских элементов управления вы должны использовать встроенные клипы. Как это делается, мы рассмотрим в конце данной главы.

Понятие элемента управления неразрывно связано с понятием компонента. (Поэтому мы будем рассматривать их вместе.) *Компонентом* во Flash называется своего рода "кусочек" графики и кода ActionScript, который вы можете вставлять в свои приложения. Все элементы управления, поставляемые с Flash, реализованы в виде компонентов. Вы также можете писать свои компоненты и потом использовать их где угодно и даже передавать коллегам.

Компоненты — одно из значительных нововведений, появившихся в программировании в последнее время. (Снобы от программирования, правда, так не считают, но нам с ними не по пути.) Они позволяют создавать весьма сложные приложения, просто собирая вместе отдельные их "кирпичики", возможно, написанные другими разработчиками. Количество средств разработки, поддерживающих компонентное программирование, неуклонно растет, и совсем недавно к ним добавился и наш любимый Flash.

Итак, приступим к рассмотрению создания приложений Flash. И начнем с кнопок.

Кнопки

Здесь мы рассмотрим создание и использование кнопок — простейших элементов управления, предоставляемых Flash. Кнопки используются очень часто, и не только в приложениях, но и в обычных интерактивных фильмах Flash, например, для создания Web-сайтов.

Создание кнопок

Чтобы создать образец-кнопку, сделайте следующее. Выберите пункт **New Symbol** в меню **Insert** или нажмите комбинацию клавиш <Ctrl>+<F8>. Если у вас открыто окно библиотеки, вы также можете выбрать пункт **New Symbol** в дополнительном меню этого окна или нажать кнопку, показанную на рис. 10.4, эта кнопка находится в нижнем левом углу окна библиотеки. На экране появится диалоговое окно **Create New Symbol**, показанное на

рис. 10.2. Введите в поле **Name** имя создаваемого образца, включите переключатель **Button** в группе **Behavior** и нажмите кнопку **OK**.

После всего этого, как вы знаете, Flash откроет вновь созданный, пока еще пустой образец в режиме правки. Вам останется нарисовать вашу кнопку.

Но не спешите это делать. Посмотрим внимательно на временную шкалу (рис. 23.1). Прекрасно видно, что кнопка — это на самом деле клип, состоящий из четырех кадров, точнее, не кадров, а делений временной шкалы. И это весьма странные кадры: все они имеют подписи: Up, Over, Down и Hit.

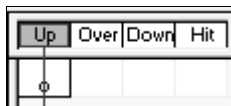


Рис. 23.1. Четыре кадра, из которых состоит кнопка

Дело в том, что эти кадры содержат изображения, которые будут высвечиваться Flash на рабочем листе в различных случаях. Давайте их перечислим:

- ☐ изображение, находящееся в кадре Up, высвечивается, когда кнопка не нажата, и курсор мыши не расположен над ней;
- ☐ изображение, находящееся в кадре Over, высвечивается, когда курсор мыши расположен над кнопкой, но сама кнопка не нажата;
- ☐ изображение, находящееся в кадре Down, высвечивается в тот момент, когда пользователь нажимает (щелкает) кнопку;
- ☐ последнее изображение (кадр Hit) задает область, которая будет откликаться на щелчки мыши ("горячую" область кнопки). Это изображение никогда не отображается Flash.

Выделите кадр Up и нарисуйте изображение кнопки. Пусть она будет овальной. После этого выделите деление временной шкалы Over и поместите туда новый ключевой кадр, выбрав пункт **Keyframe** меню **Insert** или пункт **Insert Keyframe** контекстного меню. После этого Flash автоматически скопирует нарисованное вами в первом кадре изображение в новый кадр кнопки, вам останется только его подправить. Если вы хотите полностью изменить изображение, создаваемое в очередном кадре кнопки, то нужно будет выбрать пункт **Blank Keyframe** меню **Insert** или пункт **Blank Insert Keyframe** контекстного меню.

Нарисуйте содержимое остальных двух кадров кнопки. В принципе, вам не обязательно создавать все четыре изображения. Вы можете нарисовать единственное изображение в кадре Up, так часто и поступают. После этого Flash автоматически перенесет это изображение в остальные три кадра. Конечно, такая кнопка получится "неживой", т. е. не будет реагировать на щелчок и движение мыши.

Вот и все. Теперь вернитесь в режим правки фильма и откройте окно библиотеки. Найдите в списке образцов только что созданную вами кнопку и

выделите ее. Заметьте, что вы можете "проиграть" кнопку, как клип, щелкнув находящуюся в правом верхнем углу панели предварительного просмотра образцов кнопку воспроизведения (см. рис. 10.17). Для воспроизведения можно также выбрать пункт **Play** контекстного или дополнительного меню окна библиотеки, а для останова — **Stop**.

Теперь проверим нашу кнопку в действии. Поместите экземпляр кнопки на рабочий лист. Щелкните по нему мышью, чтобы его выделить. Измените размеры кнопки. Можете задать для нее какое-либо преобразование, например, повернуть на 90° или сделать полупрозрачной.

Собственно проверить кнопку вы можете двумя путями. Во-первых, можно просто запустить проигрывание фильма. Во-вторых, можно воспользоваться еще одной интересной особенностью Flash. Для этого включите пункт-переключатель **Enable Simple Buttons** в меню **Control** или нажмите комбинацию клавиш <Ctrl>+<Alt>+. После этого кнопка станет работать прямо на рабочем листе Flash, без необходимости запуска проигрывания фильма.

Попробуйте поместить курсор мыши над кнопкой и посмотрите, как она изменится. Щелкните по кнопке. Как видите, Flash прекрасно выполняет свои обязанности.

Вы можете привязывать звуки к кнопкам. Причем, привязать свой звук можно к каждому состоянию кнопки, т. е. когда курсор мыши укажет на кнопку, будет проигран один звук, а при щелчке по кнопке — другой звук. Сейчас мы расскажем, как это делается.

Прежде всего, требуемые звуки вам необходимо импортировать. Как это делается, было подробно рассказано в *главе 17*. Поместите их прямо в библиотеку и дайте им "говорящие" имена, например, `buttonUp` и `buttonOver`. Если хотите, задайте для них параметры экспорта.

Теперь откройте нужный образец-кнопку в режиме правки. Добавьте новый слой для звуков и выделите его. После этого выделите конкретное деление на шкале анимации, в зависимости от того, к какому состоянию кнопки вы хотите привязать звук. Так, если вы хотите привязать звук к состоянию `Over` (когда курсор мыши указывает на кнопку), вам следует выделить деление `Over`. Создайте здесь новый пустой ключевой кадр. А после этого вам останется только "бросить" нужный звук на рабочий лист. Результат всего этого показан на рис. 23.2.

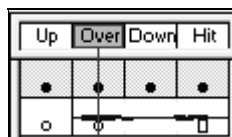


Рис. 23.2. Звук, привязанный к состоянию `Over` кнопки

Чтобы привязать звуки сразу к двум состояниям кнопки (Over или Down; к состоянию Up звук привязывать не имеет смысла), вам будет лучше всего создать для каждого звука отдельный слой. Таким образом, вы сможете лучше управлять привязанными звуками.

Кнопки Flash имеют еще одну интересную особенность. Дело в том, что "горячая" область кнопки совсем не обязательно должна совпадать с самой кнопкой. Кнопка может срабатывать, если пользователь щелкнет совсем по другому месту на рабочем листе. Это позволяет создавать весьма забавные и полезные вещи, вроде подсказок, "всплывающих" при наведении на какое-либо место клипа, и пр.

Объект *Button*

Объект `Button` "отвечает" за доступ к кнопке из сценариев ActionScript. Этот объект предоставляет набор свойств и методов, с помощью которых и осуществляется управление кнопкой.

Экземпляры объекта `Button` создаются самим Flash для каждой кнопки, для которой вы задали имя. Вам самим создавать их не нужно.

Во многом объект `Button` похож на объект `movieClip`. Эти два объекта имеют весьма схожий набор свойств и методов и зачастую ведут себя одинаково. В частности, объект `Button` поддерживает свойства `_x`, `_y`, `_xmouse`, `_ymouse`, `_height`, `_width`, `_alpha` и `_rotation`, уже знакомые вам по объекту `movieClip`. Однако объект `Button` не поддерживает методы `play`, `stop`, `gotoAndPlay`, `gotoAndStop` и подобные им, т. к. назначение его все же иное, чем у объекта `movieClip`. Так что вы можете рассматривать кнопку как сильно специализированный, "урезанный" вариант клипа.

Ниже будут рассмотрены свойства, которые обязательно пригодятся вам в работе с экземплярами объекта `Button`. (Многие из них поддерживаются и объектом `movieClip`.)

Свойство `enabled` позволяет разрешить или запретить доступ пользователя к кнопке. Оно имеет логический тип: значение `true` разрешает доступ к кнопке, а значение `false` — запрещает.

Свойство `_visible` позволяет сделать кнопку видимой или невидимой. Оно имеет логический тип: значение `true` делает кнопку видимой, а значение `false` — невидимой.

Свойство `useHandCursor` позволяет вам сменить курсор мыши, отображаемый, если мышь поместить над кнопкой. Если задано значение `true`, то отображается курсор в виде "указующего перста", как над гиперссылкой. Если же задано значение `false`, то отображается обычная стрелка.

Свойство `tabIndex` задает *порядок обхода* элементов управления при последовательных нажатиях клавиши `<Tab>`. (При нажатии комбинации клавиш

<Shift>+<Tab> обход выполняется в обратном направлении.) Значением этого свойства может быть любое целое неотрицательное число, оно задает порядок в очереди элементов управления. Так, сначала фокус ввода переместится на элемент управления со значением порядка обхода 0, потом — со значением 1 и т. д.

Если ни для одного элемента управления в приложении не задан порядок в очереди обхода (то есть, свойство `tabIndex` равно `undefined`), Flash будет применять порядок обхода по умолчанию. Если же вы собираетесь задать порядок обхода, имейте в виду, что значение `undefined` меньше любого числового. Поэтому элементы управления, для которых свойство `tabIndex` не было задано, будут в очереди обхода первыми.

Свойство `tabEnabled` позволяет убрать кнопку (и вообще любой элемент управления) из порядка обхода, сделать его недоступным для выбора с клавиатуры, но все же доступным для выбора мышью. Это свойство имеет логический тип: значение `true` или `undefined` делает элемент управления доступным для выбора с клавиатуры, а значение `false` — недоступным.

Объект `Button` поддерживает большое количество событий, которые вы можете использовать для написания обработчиков. Все эти события были перечислены в табл. 22.2.

Ниже приведены два примера обработчиков событий нажатия кнопки. (Эти обработчики привязаны к двум разным кнопкам.) Первый обработчик загружает в клип `screen` некий фильм Flash из файла `movie.swf` и запускает его проигрывание. Второй же обработчик останавливает этот фильм.

```
on(press) {  
    _root.screen.loadMovie("movie.swf");  
    _root.screen.play()  
}  
  
on(press) {  
    _root.screen.stop();  
}
```

Пользуясь методом `attachMovie` объекта `movieClip`, вы можете создавать новые кнопки в фильме или клипе:

```
on(release) {  
    _root.attachMovie("button", "button2", 0);  
}
```

Поля ввода и динамические текстовые блоки

Поля ввода и динамические текстовые блоки, описанные нами в главе 7, служат соответственно для ввода и вывода текста. С помощью текстовых

полей ваше приложение может принять какие-либо данные от пользователя, например, его имя и пароль для входа на сервер почты. А с помощью динамического текстового блока приложение может вывести пользователю результат обработки данных — тот же текст письма.

Конечно, создание почтового сервера — непростая задача. Но поля ввода и динамические текстовые блоки могут пригодиться вам и в других случаях. Поэтому мы их сейчас и рассмотрим.

Повторим еще раз: статические текстовые блоки не могут управляться из сценариев.

Объект *TextField* и его использование

И поля ввода, и динамические текстовые блоки представляются в ActionScript одинаково — как экземпляры объекта `TextField`. Объект `TextField` предоставляет набор свойств и методов, с помощью которых и осуществляется управление полем ввода или динамическим текстовым блоком.

Экземпляры объекта `TextField` создаются самим Flash для каждого поля ввода или динамического текстового блока, для которого вы задали имя. Самостоятельно создавать их не нужно.

Объект `TextField` имеет множество свойств и методов. Мы опишем только некоторые из них; полное описание этого объекта приведено в *приложении 1*.

Свойство `type` служит для задания типа экземпляра объекта `TextField`: будет это поле ввода или динамический текстовый блок. Это свойство имеет строковый тип: значение `"input"` превращает экземпляр в поле ввода, а значение `"dynamic"` — в динамический текстовый блок. Причем это свойство доступно как для чтения, так и для записи, а это значит, что вы можете превращать поля ввода в динамические текстовые блоки и наоборот.

Получить или задать текст, находящийся в поле ввода или динамическом текстовом блоке, вы можете, обратившись к свойству `text`, имеющему строковый тип. Учтите, что это свойство возвращает текст без тегов HTML, если таковые в нем присутствовали.

Из *главы 7* вы знаете, что поле ввода или динамический текстовый блок могут быть связаны с переменной, из которой они будут брать и в которую помещать свое содержимое. Впоследствии вы можете получать и задавать содержимое поля ввода или текстового блока, просто обратившись к этой переменной. Сама переменная задается с помощью свойства `variable`, имеющего строковый тип.

```
txtName.text = "Vasya Pupkin";  
name = "Vasya Pupkin";
```

Если к полю ввода `txtName` была привязана переменная `name`, то оба приведенных выше выражения будут выполнять одно и то же действие.

Объект `TextField` поддерживает свойства `_x`, `_y`, `_xmouse`, `_ymouse`, `_height`, `_width`, `_alpha` и `_rotation`, знакомые вам по объекту `movieClip`. Кроме того, он поддерживает свойства `enabled`, `_visible`, `tabEnabled` и `tabIndex`.

Свойство `html` позволяет разрешить или запретить вводить в поле ввода или динамический текстовый блок текст в формате HTML. Оно имеет логический тип: значение `true` разрешает вводить HTML-текст, а значение `false` — запрещает. Если ввод HTML-текста разрешен, то вы можете воспользоваться свойством `htmlText`, чтобы задать или исходный код HTML.

```
txtName.html = true;  
txtName.htmlText = "<B>Vlad</B> <I>Dronov</I>";
```

Свойство `maxChars` позволяет задать максимальное количество символов, которое может быть введено в поле ввода пользователем. (Количество символов, помещаемое в это поле ввода в сценарии, в любом случае не ограничено.) Это свойство принимает неотрицательные числовые значения; значение `null` отменяет ограничение.

Доступное только для чтения свойство `length` возвращает количество символов, введенное в поле ввода.

Свойство `multiline` позволяет сделать поле ввода или динамический текстовый блок многострочным. Для этого передайте этому свойству значение `true`. Значение `false` вновь делает поле ввода или динамический текстовый блок однострочным.

Свойство `password` позволяет превратить обычное поле ввода в поле ввода пароля. Для этого передайте этому свойству значение `true`. Напомним, что поле ввода пароля отличается от обычного поля ввода тем, что вместо любых символов появляются звездочки. Значение `false` вновь делает поле ввода обычным.

Свойство `selectable` позволяет сделать поле ввода или динамический текстовый блок выделяемым. При этом пользователь сможет выделять текст, находящийся в данном поле или текстовом блоке, и копировать его в буфер обмена Windows. Это свойство имеет логический тип: значение `true` разрешает пользователю выделять текст, а значение `false` — запрещает.

Если вы хотите, чтобы текст в поле ввода или динамическом текстовом блоке переносился по строкам, присвойте значение `true` свойству `wordWrap`. Значение `false` отменяет автоматический перенос строк текста.

Объект `TextField` поддерживает также несколько свойств, задающих форматирование помещенного в него текста. Все эти свойства перечислены в табл. 23.1.

Таблица 23.1. Свойства форматирования текста объекта *TextField*

Свойство	Описание
<code>background</code>	Если равно <code>true</code> , то поле ввода или динамический текстовый блок будет иметь непрозрачный фон. Если равно <code>false</code> , то фона нет (или есть прозрачный фон)
<code>backgroundColor</code>	Цвет непрозрачного фона
<code>border</code>	Если равно <code>true</code> , то поле ввода или динамический текстовый блок будет иметь рамку. Если равно <code>false</code> , то рамки нет
<code>borderColor</code>	Цвет рамки
<code>textColor</code>	Цвет текста

Объект `TextField` также поддерживает четыре события, перечисленные в табл. 23.2.

Таблица 23.2. События, поддерживаемые объектом *TextField*

Свойство	Описание
<code>onChanged</code>	Вызывается, если содержимое поля ввода было изменено пользователем
<code>onKillFocus</code>	Вызывается, если поле ввода или динамический текстовый блок потерял фокус ввода. В обработчик этого события передается единственный параметр — ссылка на элемент управления, принявший фокус
<code>onScroller</code>	Вызывается при прокрутке содержимого поля ввода или динамического текстового блока
<code>onSetFocus</code>	Вызывается, если поле ввода или динамический текстовый блок получил фокус ввода. В обработчик этого события передается единственный параметр — ссылка на элемент управления, до этого момента имевший фокус ввода

Заметьте, что обычный синтаксис обработчика события вы использовать не сможете. То есть, код вида:

```
on(changed) {
    needToSave = true;
}
```

работать не будет. Вам придется в первом кадре фильма (это самое лучшее место для подобного рода дел) создать функцию-обработчик события:

```
function txtNameChanged () {
    needToSave = true;
}
```

и присвоить ее свойству `onChanged` соответствующего экземпляра объекта `textField`:

```
_root.txtName.onChanged = txtNameChanged;
```

Впрочем, можно сделать намного проще и компактнее, совместив оба этих фрагмента кода:

```
_root.txtName.onChanged = function() { needToSave = true; };
```

События `onChanged` и `onScroller` могут быть обработаны с помощью объекта-перехватчика.

```
myListener = new Object();
```

```
myListener.onChanged = function() { needToSave = true; };
```

Для привязки только что созданного экземпляра объекта-перехватчика воспользуемся хорошо вам знакомым методом `addListener`. Это опять же лучше всего выполнять в первом кадре фильма.

```
_root.txtName.addListener(myListener);
```

Для удаления объекта-перехватчика воспользуйтесь методом `removeListener`.

```
_root.txtName.removeListener(myListener);
```

Воспользуйтесь этим методом, если вы сами хотите удалить ненужный объект-перехватчик, созданный вами для временных нужд. Если же вы создали объект-перехватчик, который собираетесь использовать постоянно, то можете его не удалять. Когда проигрыватель Flash завершится, он сам очистит всю занимаемую им память.

Вы можете использовать метод `createTextField` объекта `movieClip` для создания поля ввода или динамического текстового блока из сценария. Формат этого метода таков:

```
<Клип>.createTextField(<Имя экземпляра>, <Уровень экземпляра>, <X>, <Y>,  
    <Ширина>, <Высота>);
```

Этот метод принимает шесть обязательных параметров. Первый параметр задает имя создаваемого поля ввода или динамического текстового блока. Второй параметр задает уровень, аналогично уровню клипа. Третий и четвертый параметры задают соответственно горизонтальную и вертикальную координаты верхнего левого угла создаваемого поля ввода (динамического текстового блока) относительно клипа, в котором оно создается. Последние два параметра задают размеры — ширину и высоту — поля ввода (динамического текстового блока). И координаты, и размеры задаются в пикселах.

Учтите, что изначально создается динамический текстовый блок, т. е. свойство `type` соответствующего экземпляра объекта `TextField` равно `"dynamic"`. Чтобы преобразовать этот экземпляр в поле ввода, присвойте свойству `type` строку `"input"`.

```
_root.createTextField("txtName", 0, 100, 100, 200, 50);  
_root.txtName.type = "input";  
_root.txtName.variable = "Name";
```

Для удаления поля ввода или динамического текстового блока, созданного с помощью предыдущего метода, используйте метод `removeTextField` объекта `TextField`. Имейте только в виду, что поле ввода (динамический текстовый блок), созданное в среде Flash, удалить таким способом не удастся. Формат записи метода следующий:

```
_root.txtName.removeTextField();
```

Пример приложения Flash, использующего поля ввода

Теперь самое время создать небольшое приложение Flash, которое использовало бы поля ввода. Давайте сделаем приложение, которое будет получать имя и фамилию пользователя, а потом, в ответ на нажатие кнопки, выводить приветствие.

Создайте новый документ Flash. Поместите на рабочий лист два поля ввода под именами `txtName1` и `txtName2`. Как это сделать, было описано в главе 7. Поместите левее этих полей ввода текстовые подписи. Создайте кнопку, поместите ее экземпляр на рабочий лист и назовите его `ok`. Сохраните готовый документ. У вас должно получиться то, что показано на рис. 23.3.

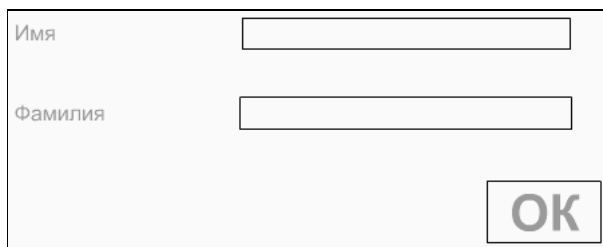


Рис. 23.3. Первый кадр приложения Flash, использующего поля ввода

Теперь создайте второй, пустой ключевой кадр. Поместите в него единственный динамический текстовый блок и дайте ему имя `txtOutput`. В него наш сценарий выведет приветственный текст.

Присвойте первому кадру фильма следующий простой сценарий:

```
_root.stop();
```

Это выражение остановит проигрывание фильма на первом же кадре, где помещаются наши поля ввода.

Осталось присвоить кнопке `ok` обработчик события `release`:

```
on(release) {  
    with (_root) {  
        nextFrame();  
        txtOutput.text = "Здравствуйте, " + txtName1.text + " " +  
            txtName2.text + "!";  
    }  
}
```

Метод `nextFrame`, как вы знаете, перемещает указатель на следующий кадр фильма, после чего проигрыватель Flash покажет содержимое этого кадра, где находится динамический текстовый блок с текстом приветствия. А текст этот формируется вторым выражением сценария.

Запускаем проигрывание фильма, вводим нужные значения в поля ввода, нажимаем кнопку и получаем только надпись "Здравствуйте!". Такое впечатление, что Flash не может получить доступ к полям ввода `txtName1` и `txtName2`, чтобы извлечь введенный в них текст. Но это, собственно, и правильно — ведь этих полей нет во втором кадре. А выражение, формирующее приветственный текст, выполняется как раз тогда, когда проигрыватель Flash показывает второй кадр нашего фильма. (Это, кстати, урок на будущее.)

Что делать? Выход очевиден: привяжем к полям ввода переменные и впоследствии обратимся к ним за введенными данными. К полю `txtName1` привяжем переменную `name1`, а к полю `txtName2` — переменную `name2`. Теперь осталось немного изменить код сценария, привязанного к кнопке:

```
on(release) {  
    with (_root) {  
        nextFrame();  
        txtOutput.text = "Здравствуйте, " + name1 + " " + name2 + "!";  
    }  
}
```

и все будет прекрасно работать.

Форматирование текста

Как вы помните из *главы 7*, имеется возможность форматировать текст, помещаемый в поля ввода и динамические текстовые блоки. Вы можете выделять текст различными шрифтами, цветом, устанавливать различные виды выравнивания для абзацев, задавать отступы текста и красной строки. Все это можно выполнять как в среде Flash, так и с помощью сценариев.

Управление форматированием текста осуществляется с помощью объекта `TextFormat`. Он содержит ряд свойств, с помощью которых и задаются параметры форматирования текста.

Прежде всего, нужно создать экземпляр объекта `TextFormat`.

```
myFormat = new TextFormat();
```

После этого вы можете задать требуемый формат с помощью соответствующих свойств. Полностью все эти свойства перечислены в *приложении 1*, здесь же мы приведем только некоторые из них.

Свойство `bold` позволит вам сделать шрифт текста полужирным. Это свойство имеет логический тип: значение `true` делает шрифт полужирным, а `false` — обычным.

Свойство `italic` позволит вам сделать шрифт текста курсивным. Это свойство также имеет логический тип: значение `true` делает шрифт курсивным, а `false` — обычным.

Аналогичное свойство `underline` задает для текста подчеркивание. Для этого присвойте этому свойству значение `true`. Значение `false` убирает подчеркивание текста.

Чтобы задать для текста новый шрифт, присвойте его имя свойству `font`:

```
myFormat.font = "Verdana";
```

А свойство `size` позволяет задать размер (кегель) шрифта в пунктах:

```
myFormat.size = 48;
```

Сменить цвет текста вы можете, присвоив нужное значение свойству `color`:

```
myFormat.color = 0x002277;
```

Чтобы задать выравнивание текста в абзаце, вам нужно воспользоваться свойством `align`. Вы можете присвоить ему три строковых значения: `"left"` (выравнивание по левому краю), `"center"` (по центру) и `"right"` (по правому краю).

А свойство `url` позволит вам превратить фрагмент текста в гиперссылку:

```
myFormat.url = "http://www.marcomedia.com";
```

Чтобы выполнить обратное преобразование — из гиперссылки в обычный текст — присвойте этому свойству пустую строку:

```
myFormat.url = "";
```

Хорошо, мы задали нужные параметры форматирования текста. Но как теперь применить их к этому тексту?

Очень просто! Объект `TextField` предоставляет для этого два метода. Мы рассмотрим их последовательно.

Метод `setTextFormat` позволяет применить заданный формат к уже существующему в поле ввода или динамическом текстовом блоке тексту. Этот метод имеет три формата вызова, которые мы сейчас перечислим.

Первый, "глобальный" формат самый простой:

```
<Текстовое поле>.setTextFormat(<Экземпляр объекта TextFormat>);
```

Этот формат вызова применяет заданные параметры форматирования ко всему тексту в поле ввода или динамическом текстовом блоке. Экземпляр объекта `TextFormat`, задающий установленное форматирование, передается в метод единственным параметром.

```
txtNotes.setTextFormat(myFormat);
```

Второй формат вызова:

```
<Текстовое поле>.setTextFormat(<Номер символа>,  
    &<Экземпляр объекта TextFormat>);
```

Этот формат присваивает параметры форматирования символу текста, номер которого передан в качестве первого параметра. Учтите, что нумерация символов текста начинается с нуля. Вторым параметром передается нужный экземпляр объекта `TextFormat`.

```
txtNotes.setTextFormat(0, firstLetterFormat);
```

Третий формат вызова:

```
<Текстовое поле>.setTextFormat(<Номер первого символа>,  
    &<Номер последнего символа>,  
    &<Экземпляр объекта TextFormat>);
```

Этот формат присваивает параметры форматирования целой последовательности символов текста. Номер первого символа последовательности передается первым параметром метода, номер последнего символа последовательности — вторым параметром. Третьим параметром передается нужный экземпляр объекта `TextFormat`.

```
txtNotes.setTextFormat(10, 30, nameFormat);
```

Второй метод — `setNewTextFormat` — позволяет задать формат текста, который будет введен пользователем позднее. Формат его вызова очень прост:

```
<Текстовое поле>.setNewTextFormat(&Экземпляр объекта TextFormat);
```

Экземпляр объекта `TextFormat`, задающий форматирование, передается в метод единственным параметром.

Кроме того, объект `TextField` предоставляет два парных метода, возвращающих ссылку на экземпляр объекта `TextFormat`, описывающий форматирование текста. Их также два: `getTextFormat`, возвращающий параметры форматирования уже существующего текста, и `getNewTextFormat`, возвращающий параметры форматирования текста, который будет введен пользователем потом. Описание этих методов см. в *приложении 1*.

Объект *Selection*

Flash также предоставляет возможность управления текстовым курсором и выделением текста в полях ввода и динамических текстовых блоках. Для этого предназначен объект `Selection`, единственный экземпляр которого по имени `Selection` создается самим Flash.

С помощью объекта `Selection` вы можете получить позицию текстового курсора в поле ввода. Для этого вам нужно воспользоваться методом `getCaretIndex`. Если же ни одно поле ввода не имеет фокуса, возвращается `-1`.

Методы `getBeginIndex` и `getEndIndex` возвращают номера соответственно начального и конечного символа выделенного фрагмента текста. Если ничего не выделено, опять же возвращается `-1`.

Вы можете выделить нужный фрагмент текста, воспользовавшись методом `setSelection`. Первым параметром этого метода должен быть номер первого символа выделяемого фрагмента, а вторым — номер последнего символа. Запомните, что нумерация символов текста начинается с нуля.

```
Selection.setSelection(10, 30);
```

Вы также можете просто поставить текстовый курсор в требуемую позицию, передав методу `setSelection` номер нужного символа и в первом, и во втором параметре:

```
Selection.setSelection(10, 10);
```

Метод `getFocus` возвращает имя переменной, привязанной к полю ввода, имеющему в данный момент фокус ввода. Если ни одно поле ввода не имеет фокуса, возвращается `null`.

Вы можете дать фокус вводу какому-либо полю. Для этого передайте имя переменной, привязанной к этому полю, в качестве параметра методу `setFocus`:

```
Selection.setFocus("Name");
```

```
Selection.setFocus("_root.id");
```

Точно таким же образом вы можете дать фокус вводу кнопке, передав этому методу путь кнопки:

```
Selection.setFocus("_root.btnOK");
```

Чтобы снять фокус ввода со всех элементов управления, передайте методу `setFocus` значение `null`.

Вы можете обрабатывать событие `onSetFocus`, наступающее, когда поле ввода или динамический текстовый блок получает фокус ввода. Для этого вам будет необходимо использовать объект-перехватчик.

Элементы управления

Все, что мы проходили ранее, суть простейшие элементы пользовательского интерфейса, предоставляемые Flash разработчикам. С их помощью вы можете создавать только самые простые приложения, которые и приложениями-то назвать можно с трудом: формы ввода, элементы оформления Web-страниц и пр. Конечно, никто не спорит, что и формы ввода данных, и элементы оформления Web-страниц нужны и важны. Но для их создания достаточно зачастую хватит обычных кнопок, можно обойтись даже без полей ввода.

Высший пилотаж Flash-программирования — это создание полноценных приложений, сходных с приложениями Windows. Это могут быть игры, утилиты, калькуляторы, даже текстовые редакторы. Для создания пользовательского интерфейса таких приложений применяются настоящие элементы управления: флажки, переключатели, списки, меню и т. п.

Что такое элементы управления Flash? Ничего особенного — обычные клипы, только ведущие себя особым образом, благодаря соответствующим сценариям ActionScript. Элементы управления имеют набор особых свойств и методов, позволяющих задавать и получать их состояние и управлять ими. Задавать начальные значения свойств вы можете как в среде Flash, так и в сценарии.

Вот о таких элементах управления мы и поговорим. Сначала мы рассмотрим *встроенные элементы управления*, которые поставляются с самим Flash, а потом поговорим о создании пользовательских элементов управления.

Компоненты и работа с ними

Но прежде, чем мы начнем говорить об элементах управления, нам нужно выяснить еще кое-что. А именно, все, связанное с компонентами.

Понятие о компонентах

Flash MX привносит в создание приложений кое-что новое, а именно, компоненты. Компонентом называется специальный клип, имеющий набор свойств и методов и ведущий себя особым образом. Такие компоненты представляют собой отдельные, независимые графические элементы и фрагменты кода ActionScript, которые можно использовать в любом Flash-приложении. Таким образом, компоненты — это особый вид пользовательских объектов.

Типичный пример компонента — любой встроенный элемент управления Flash. Но компоненты не обязательно должны представлять элементы пользовательского интерфейса. Они могут быть обычными графическими элементами, которые используются в фильмах, или вообще быть невидимыми и служить только для целей программирования. Примерами такого рода компонентов могут быть фигурный курсор мыши и невидимый компонент, отслеживающий нажатия клавиш клавиатуры и перемещающий заданный клип по рабочему листу.

Как правило, любой компонент делается независимым от его окружения. Это значит, что вы можете поместить в свой фильм или приложение некоторый компонент, и он будет работать нормально все независимо от других компонентов фильма или приложения. Если же один компонент требует наличия на листе других компонентов, то об этом особо указывается в его описании.

Работа с компонентами в среде Flash

Все доступные во Flash компоненты отображаются в панели **Components**. Чтобы вызвать ее на экран, включите пункт-выключатель **Components** в меню **Window** или нажмите комбинацию клавиш <Ctrl>+<F7>. Сама панель показана на рис. 23.4.

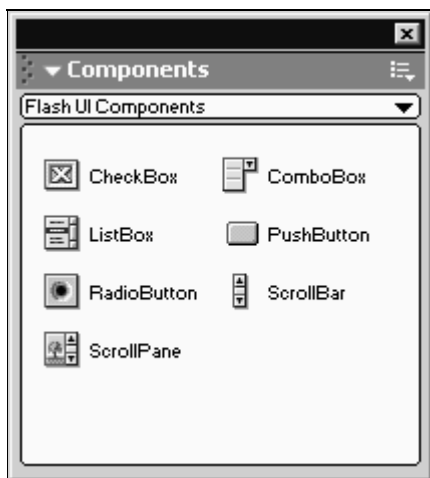


Рис. 23.4. Панель **Components**

Как видите, большую часть этой панели занимает список компонентов. На самом деле, в панели показывается только одна страница этого списка. Чтобы увидеть содержимое других страниц, откройте меню, щелкнув по маленькой стрелке в правом верхнем углу окна, и выберите в нем нужный пункт-переключатель (рис. 23.5). Изначально, однако, список компонентов состоит всего из одной страницы — **Flash UI Components**, в которой перечислены встроенные элементы управления Flash.



Рис. 23.5. Меню страниц списка компонентов

По умолчанию, в списке компонентов показываются их графические обозначения и текстовые подписи. Вы можете скрыть подписи, оставив только графические обозначения. Для этого отключите пункт-выключатель **Show Description** в контекстном или дополнительном меню панели.

Чтобы добавить компонент на рабочий лист, сделайте одно из двух: либо перетащите его из панели **Components** на лист, либо дважды щелкните по нему. После этого выбранный вами компонент появится на рабочем листе (рис. 23.6).

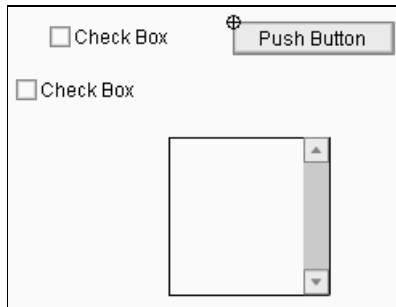


Рис. 23.6. Несколько компонентов на рабочем листе

По умолчанию все компоненты будут выглядеть так же, как во время проигрывания фильма. Вы можете заставить их выглядеть "проще", более схематично, но не получите от этого никаких особых преимуществ. Для этого отключите пункт-выключатель **Enable Live Preview** в меню **Control**.

Вы уже знаете, что все компоненты — на самом деле образцы-клипы. Поэтому, помещение компонента на рабочий лист — это создание экземпляра соответствующего образца-клипа. Из этого следует, что все компоненты должны присутствовать и в окне библиотеки. Если вы откроете это окно, то увидите их (рис. 23.7). Причем для удобства поиска все они будут помещены в папку **Flash UI Components**.



Рис. 23.7. Компоненты в окне библиотеки

Кроме самих компонентов, в окне библиотеки, в той же папке присутствует множество других образцов-клипов. Это различные вспомогательные графические элементы, в частности, различные фрагменты компонентов. Не удаляйте их, иначе компоненты станут неработоспособными.

Вы можете перемещать выделенные компоненты по листу, менять их размеры, вращать и искажать, как и обычные графические фрагменты. Имейте только в виду, что эти операции применимы не для всех компонентов. Так, флажки можно только перемещать, но нельзя изменять их размеры.

Теперь нужно задать параметры созданных вами компонентов. Для этого можно воспользоваться либо редактором свойств, либо панелью **Component Parameters**. Но, прежде всего, вам нужно выделить нужный компонент.

Взгляните на редактор свойств (рис. 23.8). Кроме поля ввода имени компонента и набора полей, служащих для задания его местоположения и размеров, вы увидите список доступных параметров и их значений. Вы можете выделить любой параметр и задать нужное значение с помощью поля ввода или списка, которые появятся правее в этой же строке. После этого нажмите клавишу <Enter>, чтобы сохранить введенное значение, и <Esc>, чтобы отменить его.

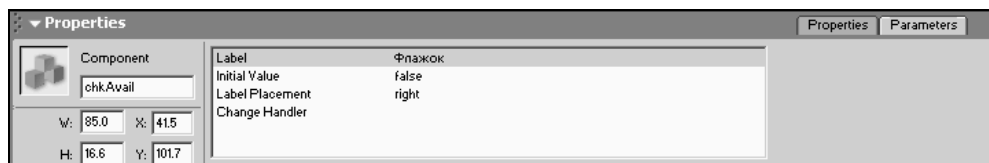


Рис. 23.8. Вид редактора свойств при выделенном компоненте

Если вы хотите получить доступ к обычному редактору свойств, предоставляющему средства для задания преобразований цвета компонента, щелкните кнопку **Properties**, расположенную в верхней части редактора. Вернуться к заданию параметров компонента вы можете, щелкнув кнопку **Parameters**, расположенную там же.

Кроме того, вы можете задать параметры компонента в панели **Component Parameters**. Чтобы вызвать ее на экран, включите пункт-выключатель **Components Parameters** в меню **Window** или нажмите комбинацию клавиш <Alt>+<F7>. Сама эта панель показана на рис. 23.9.

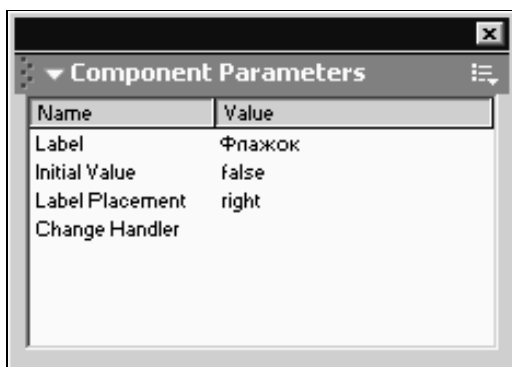


Рис. 23.9. Панель **Components Parameters**

На наш взгляд, пользоваться панелью **Components Parameters** проще, чем редактором свойств. Во-первых, список параметров, находящийся в ней, состоит из двух колонок: `Name` (имя свойства) и `Value` (значение свойства), так что имя и значение свойства разделены. Во-вторых, вы можете вывести на экран справку по выделенному компоненту, выбрав пункт **Help on Component** в дополнительном меню этой панели.

Удалить компонент, помещенный на рабочий лист, очень просто. Выделите его и нажмите клавишу ``. Однако, если это последний компонент такого типа (например, последний список), вам следует также удалить соответствующие ему образцы-клипы из библиотеки. Если вы это не сделаете, в вашем фильме останется бесполезный графический мусор, который будет только занимать место на диске.

Как вы знаете, все образцы-клипы, соответствующие добавленным вами на рабочий лист компонентам, находятся в папке `Flash UI Components`. Поэтому вы легко сможете найти и удалить ненужный образец. Какой образец какому компоненту соответствует, будет описано далее в этой главе. Так, для списков это будет образец `ListBox`, а для флажков — образец `CheckBox`.

Кроме основных образцов, для компонента в библиотеке могут присутствовать и дополнительные, обычно представляющие какие-либо его графические фрагменты. Эти вспомогательные компоненты вы можете найти в папках (имеются в виду папки библиотеки) `Component Skins` и `Core Assets — Developer Only\Other Assets`, расположенных в папке `Flash UI Components`. Вы сразу увидите их: они расположены в папках, названия которых содержат название типа соответствующего компонента (`ListBox`, `CheckBox` и т. п.).

Встроенные элементы управления Flash

Итак, с компонентами покончено. Конечно, мы и в дальнейшем будем упоминать этот термин в связи с элементами управления. А в самом конце этой главы мы рассмотрим примеры создания собственных компонентов: видимого элемента управления и невидимого служебного компонента.

А сейчас мы опишем элементы управления, поставляемые вместе с Flash. Напомним, что все они реализованы в виде компонентов.

Флажок (*CheckBox*)

Компонент `CheckBox` позволит вам создать флажок, имеющий два состояния: "включено" и "выключено". Внешний вид этого компонента схож с внешним видом стандартного флажка Windows.

Компонент `CheckBox` в среде Flash предоставляет разработчику набор параметров, перечисленных в табл. 23.3.

Таблица 23.3. Параметры, отображаемые элементом управления
CheckBox в среде Flash

Параметр	Описание
Change Handler	Имя функции-обработчика события, происходящего при щелчке мышью на флажке. Функция-обработчик должна быть объявлена в том же клипе, в котором находится элемент управления. Не обязателен. Задается в строковом виде
Initial Value	Начальное состояние флажка: включен (true) или отключен (false). Значение по умолчанию — false (флажок отключен)
Label	Текстовая подпись
Label Placement	Местонахождение текстовой подписи: левее (left) или правее (right) флажка. Значение по умолчанию — right (подпись находится правее флажка)

Флажок Flash возвращает значение `true`, если он включен, и `false`, если выключен.

Фактически элемент управления `CheckBox` состоит из двух компонентов: собственно флажка и текстовой надписи `Label`. Поэтому, если вы удаляете соответствующий флажку образец-клип из библиотеки, не забудьте удалить также и образец, представляющий текстовую надпись.

Вы можете задать ширину, но не высоту флажка. Высота его всегда остается постоянной и определяется размером шрифта текстовой надписи.

Раскрывающийся список (*ComboBox*)

Компонент `ComboBox` позволит вам создать раскрывающийся список, предоставляющий пользователю возможность выбора одного пункта. Причем пользователь также сможет ввести в список новое значение с клавиатуры. Внешний вид этого компонента схож с внешним видом стандартного раскрывающегося списка Windows.

Компонент `ComboBox` в среде Flash предоставляет разработчику набор параметров, перечисленных в табл. 23.4.

Таблица 23.4. Параметры, отображаемые элементом управления
ComboBox в среде Flash

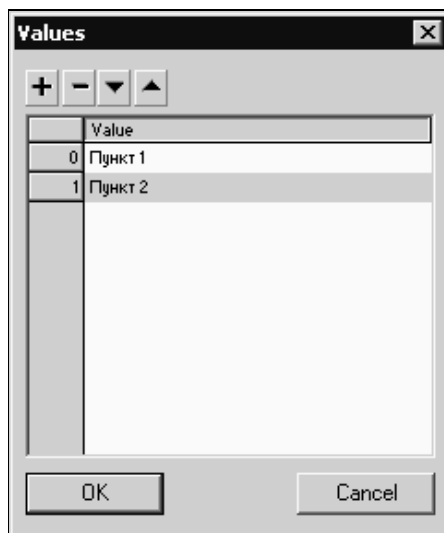
Параметр	Описание
Change Handler	Имя функции-обработчика события, происходящего при выборе пункта в списке или вводе в него значения (если это разрешено). Функция-обработчик должна быть объявлена в том же клипе, в котором находится элемент управления. Не обязателен. Задается в строковом виде

Таблица 23.4 (окончание)

Параметр	Описание
Data	Массив строк, представляющий значения, привязанные к пунктам списка. Не обязателен
Editable	Задаёт, может ли пользователь вводить в раскрывающийся список значение с клавиатуры. Пункт true разрешает ввод, пункт false запрещает. Значение по умолчанию — false
Labels	Массив строк, представляющий названия пунктов списка
Row Count	Количество пунктов, одновременно отображаемых в списке. Если количество пунктов в списке превышает это значение, появляется полоса прокрутки. Значение по умолчанию — 8

Раскрывающийся список Flash работает следующим образом. Для создания пунктов используются строки из массива `Labels`, в то время как массив `Data` хранится в памяти и нигде не показывается. Когда пользователь выбирает какой-либо пункт списка, элемент управления возвращает соответствующую ему строку из массива `Data`. Если массив `Data` не задан, возвращается название выбранного пункта (значение массива `Label`). Если пользователь вводит значение с клавиатуры, возвращается именно оно.

Для ввода массивов строк Flash предоставляет диалоговое окно **Values** (рис. 23.10). Чтобы вызвать его на экран, нажмите кнопку с изображением лупы в списке параметров редактора свойств.

Рис. 23.10. Диалоговое окно **Values**

Большую часть этого окна занимает список строк, составляющих массив. Как видите, все строки пронумерованы, и нумерация их (как это принято в массивах ActionScript) начинается с нуля. Вы можете выделить требуемую строку и произвести над ней какие-либо действия.

Чтобы ввести в список новую строку, нажмите кнопку с изображением знака "плюс". В список будет добавлена новая строка. Щелкните по ней, и вместо нее появится небольшое поле ввода. Введите в него нужный текст и нажмите клавишу <Enter>.

Чтобы удалить строку, выделите ее и нажмите кнопку с изображением знака "минус".

Чтобы изменить строку, выделите ее. После этого вместо нее появится небольшое поле ввода, содержащее текст строки. Введите в него новый текст и нажмите клавишу <Enter> для его сохранения или клавишу <Esc> для возврата к старому тексту.

Введя массив строк, нажмите кнопку **ОК**, чтобы сохранить его и закрыть окно **Values**. Если вы не хотите сохранять введенный массив, нажмите кнопку **Cancel**.

Фактически элемент управления `ComboBox` состоит из двух компонентов: собственно раскрывающегося списка и полосы прокрутки `ScrollBar`. Поэтому, если вы удаляете соответствующий списку образец-клип из библиотеки, не забудьте удалить также и образец, представляющий полосу прокрутки.

Вы можете задать ширину, но не высоту раскрывающегося списка. Высота его всегда остается постоянной и определяется размером шрифта, которым набраны пункты списка.

Обычный список (*ListBox*)

Компонент `ListBox` позволит вам создать обычный список, предоставляющий пользователю возможность выбора одного или нескольких пунктов. Внешний вид этого компонента схож с внешним видом стандартного списка Windows.

Компонент `ListBox` в среде Flash предоставляет разработчику набор параметров, перечисленных в табл. 23.5.

Таблица 23.5. *Параметры, отображаемые элементом управления `ListBox` в среде Flash*

Параметр	Описание
Change Handler	Имя функции-обработчика события, происходящего при выборе пункта в списке. Функция-обработчик должна быть объявлена в том же клипе, в котором находится элемент управления. Не обязателен. Задается в строковом виде

Таблица 23.5 (окончание)

Параметр	Описание
Data	Массив строк, представляющий значения, привязанные к пунктам списка. Не обязателен
Labels	Массив строк, представляющий названия пунктов списка
Select Multiple	Задаёт, может ли пользователь выбирать в списке сразу несколько значений. Пункт true разрешает это, пункт false запрещает. Значение по умолчанию — false

Список Flash работает следующим образом. Для создания пунктов используются строки из массива `Labels`, в то время как массив `Data` хранится в памяти и нигде не показывается. Когда пользователь выбирает какой-либо пункт списка, элемент управления возвращает соответствующую ему строку из массива `Data`. Если массив `Data` не задан, возвращается название выбранного пункта (значение массива `Label`). Если пользователь выбрал сразу несколько пунктов, возвращается массив, состоящий из всех выбранных пунктов.

Для ввода массивов строк Flash предоставляет диалоговое окно **Values** (см. рис. 23.10).

Фактически элемент управления `ListBox` состоит из двух компонентов: собственно списка и полосы прокрутки `ScrollBar`. Поэтому, если вы удаляете соответствующий списку образец-клип из библиотеки, не забудьте удалить также и образец, представляющий полосу прокрутки.

Когда вы изменяете высоту списка, Flash автоматически подгоняет ее так, чтобы в списке отображалось целое число пунктов.

Кнопка (*PushButton*)

Компонент `PushButton` позволит вам создать обычную командную кнопку. В общем, она сходна с уже рассмотренными нами кнопками Flash, но выглядит как стандартная кнопка Windows.

Компонент `PushButton` в среде Flash предоставляет разработчику набор параметров, перечисленных в табл. 23.6.

Таблица 23.6. Параметры, отображаемые элементом управления *PushButton* в среде Flash

Параметр	Описание
Change Handler	Имя функции-обработчика события, происходящего при выборе пункта в списке. Функция-обработчик должна быть объявлена в том же клипе, в котором находится элемент управления. Задается в строковом виде

Таблица 23.6 (окончание)

Параметр	Описание
Label	Название кнопки

Фактически элемент управления `PushButton` состоит из двух компонентов: собственно кнопки и текстовой надписи `Label`. Поэтому, если вы удаляете соответствующий кнопке образец-клип из библиотеки, не забудьте удалить также и образец, представляющий текстовую надпись.

Переключатель (*RadioButton*)

Компонент `RadioButton` позволит вам создать переключатель. Внешний вид этого компонента схож с внешним видом стандартного переключателя Windows. Компонент `RadioButton` в среде Flash предоставляет разработчику набор параметров, перечисленных в табл. 23.7.

Таблица 23.7. Параметры, отображаемые элементом управления *RadioButton* в среде Flash

Параметр	Описание
Change Handler	Имя функции-обработчика события, происходящего при выборе переключателя. Функция-обработчик должна быть объявлена в том же клипе, в котором находится элемент управления. Не обязателен. Задается в строковом виде
Data	Значение, привязанное к переключателю. Не обязателен
Group Name	Имя группы, в которую входит переключатель
Initial State	Начальное состояние переключателя: включен (true) или отключен (false). Только один переключатель в группе может быть включен. Значение по умолчанию — false (переключатель отключен)
Label	Название переключателя
Label Placement	Местонахождение текстовой подписи: правее (left) или левее (right) переключателя. Значение по умолчанию — right (подпись находится правее флажка)

В одном переключателе толку немного — переключатели должны составлять группы. Переключатели, входящие в одну группу, должны иметь одинаковое значение параметра `Group Name`. Только один переключатель может быть включен. Хорошим тоном считается выделение группы переключателей, например, линиями или прямоугольником (рис. 23.11).

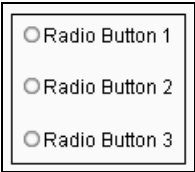


Рис. 23.11. Группа переключателей, выделенная прямоугольником

Группа переключателей возвращает значение параметра `Data` включенного переключателя. Если параметр `Data` не задан, возвращается название переключателя `Label`.

Фактически элемент управления `RadioButton` состоит из двух компонентов: собственно переключателя и текстовой надписи `Label`. Поэтому, если вы удаляете соответствующий переключателю образец-клип из библиотеки, не забудьте удалить также и образец, представляющий текстовую надпись.

Вы можете задать ширину, но не высоту переключателя. Высота его всегда остается постоянной и определяется размером шрифта текстовой надписи.

Полоса прокрутки (*ScrollBar*)

Компонент `ScrollBar` позволит вам создать обычную полосу прокрутки. Она может быть использована как отдельный элемент управления, предназначенный для ввода данных, так и в составе более сложных элементов управления. Внешний вид этого компонента схож с внешним видом стандартной полосы прокрутки `Windows`.

Компонент `ScrollBar` имеет одну особенность, обусловленную деталями реализации. После того, как вы добавили первую полосу прокрутки на рабочий лист (а также один из содержащих ее компонентов — `ComboBox`, `ListBox` или `ScrollPane`), добавлять новые полосы прокрутки вы можете только перетаскиванием соответствующего ей образца из окна библиотеки. Вы не должны добавлять на лист новые полосы прокрутки перетаскиванием из панели **Components**.

Компонент `ScrollBar` в среде `Flash` предоставляет разработчику набор параметров, перечисленных в табл. 23.8.

Таблица 23.8. *Параметры, отображаемые элементом управления `ScrollBar` в среде `Flash`*

Параметр	Описание
Horizontal	Задаёт, будет ли полоса прокрутки горизонтальная (true) или вертикальная (false)
Target Text Field	Поле ввода или динамическое текстовое поле, к которому будет привязана полоса прокрутки

Полоса прокрутки возвращает целое значение, показывающее позицию указателя.

Вы можете "прицепить" полосу прокрутки к полю ввода или динамическому текстовому блоку (рис. 23.12). Для этого перетащите полосу прокрутки из панели **Components** или окна библиотеки в поле ввода или динамический текстовый блок, к нужной его стороне, и "бросьте" ее там. Flash сделает за вас все остальное. Учтите только, что поле ввода или динамический текстовый блок, к которому вы привязываете полосу прокрутки, должен иметь имя, чтобы Flash смог подставить его в параметр **Target Text Field**.



Рис. 23.12. Динамический текстовый блок с привязанными к нему полосами прокрутки

Если вы измените размеры поля ввода или динамического текстового блока с привязанными полосами прокрутки, то вам нужно будет изменить размеры и полос прокрутки. Если вы удалите поле ввода или динамический текстовый блок, то вам также придется позаботиться и об удалении полос прокрутки. Flash многое делает сам, но не может сделать за вас все.

Если вы собираетесь полностью удалить полосу прокрутки из библиотеки, то сначала удостоверьтесь, что там нет ни одного использующего ее элемента управления (ComboBox, ListBox или ScrollPane).

Панель с прокруткой (*ScrollPane*)

Компонент `ScrollPane` позволит вам создать панель, в которую вы можете поместить любой клип, причем, если этот клип не будет помещаться в панель, то в ней автоматически появятся полосы прокрутки. Такая панель может быть использована для помещения больших клипов на небольшом пространстве. Панель с прокруткой показана на рис. 23.13, в нее загружена кар-

тинка с кадром из "Властелина колец" в формате JPEG, которая после импорта была преобразована в образец-клип.



Рис. 23.13. Панель с прокруткой

Компонент `ScrollPane` в среде Flash предоставляет разработчику набор параметров, перечисленных в табл. 23.9.

Таблица 23.9. Параметры, отображаемые элементом управления *ScrollPane* в среде Flash

Параметр	Описание
Drag Content	Задаёт, будет ли пользователь иметь возможность перетаскивать содержимое внутри панели (true) или нет (false). Значение по умолчанию — false
Horizontal Scroll	Задаёт, будет ли горизонтальная полоса прокрутки видна все время (true), не видна никогда (false) или будет появляться только тогда, когда в ней появится необходимость (auto). Значение по умолчанию — auto
Scroll Content	Имя сценарного образца-клипа, который станет содержимым панели с прокруткой
Vertical Scroll	Задаёт, будет ли вертикальная полоса прокрутки видна все время (true), не видна никогда (false) или будет появляться только тогда, когда в ней появится необходимость (auto). Значение по умолчанию — auto

Фактически элемент управления `ScrollPane` содержит два компонента: собственно панель и полосу прокрутки `ScrollBar`. Поэтому, если вы удаляете соответствующий панели образец-клип из библиотеки, не забудьте удалить также и образец, представляющий полосу прокрутки.

Программирование пользовательского интерфейса

Итак, мы изучили встроенные элементы управления. Теперь выясним, как связать друг с другом эти разрозненные фрагменты графики и кода.

Работа с элементами управления из сценариев

Поскольку все элементы управления являются компонентами, а все компоненты являются образцами-клипами, вы можете добавлять их на рабочий лист программно, используя метод `attachMovie` объекта `movieClip`:

```
_root.attachMovie("FCheckBoxSymbol", "chkFlag", 0);
```

Благо, все компоненты уже преобразованы в сценарные образцы.

Для удаления ненужного элемента вы должны будете использовать метод или действие `removeMovieClip`:

```
_root.chkFlag.removeMovieClip();
```

Для управления элементом интерфейса вы можете использовать все свойства объекта `movieClip`, перечисленные в *главе 22* и *приложении 1*. Кроме того, все элементы управления поддерживают свои собственные методы, которые мы сейчас вкратце рассмотрим. Полное описание этих методов вы можете найти в *приложении 1*.

Так, для задания размера любого элемента управления вы можете использовать метод `setSize`. Этот метод может иметь два формата вызова, в зависимости от того, к какому компоненту он применяется.

```
<Компонент>.setSize(<Ширина>);
```

Этот метод меняет ширину тех компонентов, чья высота задается размером шрифта текстовой надписи. Это компоненты `CheckBox`, `ComboBox` и `RadioButton`. Значение ширины задается в пикселах.

```
<Компонент>.setSize(<Ширина>, <Высота>);
```

А этот метод применяется для всех остальных компонентов, у которых можно менять оба размера.

Для того чтобы получить значение элемента управления, воспользуйтесь методом `getValue`. В зависимости от типа компонента, этот метод возвращает разные значения. Так, для флажка `CheckBox` этот метод вернет `true`, если флажок включен, и `false`, если он выключен. А для раскрывающегося списка `ComboBox` он вернет значение, привязанное к выбранному пункту списка или, если это значение не задано, само название пункта.

Метод `setValue` осуществляет обратное действие — задает значение элемента управления.

```
_root.chkAddMeIntoMailList.setValue(true);
```

Вам может также пригодиться метод `getSelectedIndex`, поддерживаемый компонентами `ComboBox` и `ListBox` и возвращающий номер выбранного в списке пункта. А метод `setSelectedIndex`, поддерживаемый этими же

компонентами, выбирает в списке пункт с номером, переданным ему в качестве параметра.

С помощью метода `setEnabled` вы можете разрешить или запретить доступ пользователю к элементу управления. Формат вызова этого метода прост: он принимает один-единственный логический параметр, `true` (доступ разрешен) или `false` (доступ запрещен). А метод `setEnabled` возвращает текущее состояние "доступности" элемента управления.

Компоненты `CheckBox`, `PushButton` и `RadioButton` поддерживают методы для получения и задания текстовой надписи. Для получения текстовой надписи служит метод `getLabel`, а для задания — `setLabel`. Текст новой надписи передается в метод `setLabel` единственным параметром.

Списки `ComboBox` и `ListBox` поддерживают методы, предназначенные для добавления новых и удаления ненужных пунктов. Для этого служат несколько методов, которые мы рассмотрим ниже.

Метод `addItem` добавляет новый пункт в конец списка.

```
<Список>.addItem(<Название>[, <Значение>]);
```

Эти параметры вам уже знакомы. Название пункта отображается в списке, а Значение возвращается при выборе этого пункта. Если Значение не задано, возвращается Название.

Метод `addItemAt` добавляет новый пункт в заданную позицию списка.

```
<Список>.addItemAt(<Номер>, <Название>[, <Значение>]);
```

Здесь добавляется еще один параметр — Номер списка. Не забывайте, что пункты списка, как и элементы массива, нумеруются с нуля.

Для удаления какого-либо пункта списка используйте метод `removeItemAt`.

```
<Список>.removeItemAt(<Номер>);
```

Для удаления всех элементов списка вызовите метод `removeAll`, не принимающий параметров.

Метод `replaceItemAt` позволяет заменить один пункт списка другим.

```
<Список>.replaceItemAt(<Номер>, <Название>[, <Значение>]);
```

Вы должны передать этому методу Номер заменяемого пункта списка, его новое Название и, что не обязательно, новое Значение.

Для заполнения списка пунктами вы можете также использовать особые объекты, называемые *объектами-поставщиками данных*. С помощью таких объектов можно задать массив пунктов для списка целиком и потом привязать его к нужному списку одним-единственным методом `setDataProvider`.

Существует две разновидности таких объектов: простая и сложная. Мы рассмотрим их по очереди.

Простой объект-поставщик данных — это обычный массив, содержащий строки, которые станут названиями пунктов списка.

```
list = new Array();  
list[0] = "Flash";  
list[1] = "Dreamweaver";  
list[2] = "Fireworks";
```

Используя массив строк, вы можете задать только названия пунктов меню, но не значения. Надо сказать, часто хватает и этого. Но если вы хотите задать и названия, и значения, вам понадобится создать массив объектов.

```
list2 = new Array();  
list2[0] = {label:"Flash", data:1};  
list2[1] = {label:"Dreamweaver", data:2};  
list2[2] = {label:"Fireworks", data:3};
```

Обратите внимание: мы присваиваем каждому элементу массива экземпляр объекта `Object`, содержащий свойства `label` (название пункта) и `data` (значение пункта).

Сложный объект-поставщик данных — это экземпляр объекта `DataProviderClass`. `DataProviderClass` — пользовательский объект, созданный программистами, разрабатывавшими встроенные компоненты Flash, на языке `ActionScript`. Реализован этот объект в образце `DataProvider`, находящимся в папке `Core Assets - Developer Only/ FUIComponent Class Tree` библиотеки. Вы можете дважды щелкнуть по значку этого образца и просмотреть сценарий, привязанный к его первому кадру, — там находится весь нужный вам код. Он довольно сложен, так что будьте внимательны.

Преимущества использования экземпляра объекта `DataProviderClass` в качестве поставщика данных очевидны. Вы можете пользоваться методами этого объекта для добавления, замены и удаления пунктов списка, а также сортировки массива пунктов. Все методы объекта описаны в *приложении 1*.

```
objList = new DataProviderClass();  
objList.addItem({label:"Flash", data:1});  
objList.addItem({label:"Dreamweaver", data:2});  
objList.addItem({label:"Fireworks", data:3});
```

Привязать объект-поставщик данных к списку можно методом `setDataProvider`:

```
lstMacromediaProducts.setDataProvider(list2);  
lstMacromediaProducts.setDataProvider(objList);
```

Как видите, формат вызова данного метода не зависит от того, простой или сложный объект-поставщик данных мы используем.

Написание обработчиков событий

Все элементы управления предоставляют для обработки только одно событие — изменение состояния. Оно наступает, когда пользователь включает

или отключает флажок, выбирает пункт списка или нажимает кнопку. Функция, обрабатывающая это событие, может выполнять самые разные действия: изменять доступность других элементов управления, изменять их значения или запускать какие-либо действия.

Существует два пути, чтобы присвоить этому событию обработчик. Во-первых, вы можете присвоить имя функции-обработчика параметру **Change Handler** (у кнопки `PushButton` этот параметр называется **Click Handler**) в среде `Flash`. Сделать это можно либо в редакторе свойств, либо в панели **Component Parameters**. Во-вторых, вы можете сделать это в сценарии, воспользовавшись методом `setChangeHandler` (у кнопки `PushButton` — `setClickHandler`).

```
<Компонент>.set<Change|Click>Handler(<Функция>[, <Путь функции>]);
```

Первым параметром этого метода передается имя функции-обработчика события в строковом виде. Вторым параметром может передаваться путь к этой функции, точнее, к клипу, где она объявлена. Если второй параметр пропущен, подразумевается, что эта функция объявлена в текущем клипе.

Функция-обработчик события должна принимать один параметр — ссылку на элемент управления, в котором произошло это событие. Это значит, что вы можете написать одну-единственную функцию, которая будет обрабатывать события, происходящие сразу в нескольких элементах управления. Чтобы выяснить, где именно произошло событие, вы можете проверять свойство `_name` объекта `movieClip`, возвращающее имя клипа:

```
function onClick(control) {  
    if (control._name == "chkSendMeEMail") {  
        _root.txtEAddress.setEnabled(_root.chkSendMaEMail.getValue());  
    } else {  
        if (control._name == "chkICQ") {  
            _root.txtICQ.setEnabled(_root.chkICQ.getValue());  
        }  
    }  
}  
  
_root.chkSendMeEMail.setChangeHandler("onClick");  
_root.chkICQ.setChangeHandler("onClick");
```

Пример приложения Flash, использующего элементы управления

Давайте напишем простейшее приложение, которое будет собирать некоторые данные о пользователе и обрабатывать их. Сделаем так, чтобы оно выдавало пользователю приветствие, основываясь на этих данных. Короче говоря, это будет слегка усложненная версия приложения, уже рассмотренного нами в этой главе.

Создадим новый документ Flash и поместим на рабочий лист три поля ввода, раскрывающийся список, флажок и кнопку. Поместим также текстовые надписи, поясняющие назначение этих элементов управления. То, что у нас должно получиться, показано на рис. 23.14.

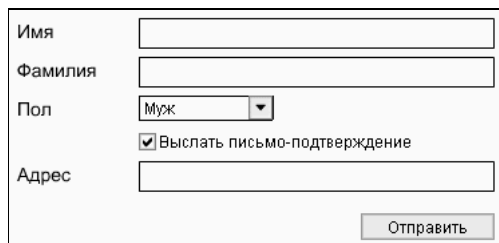
Скриншот простейшего приложения, состоящего из нескольких элементов управления, расположенных в форме. Вверху — поле ввода с меткой «Имя». Ниже — поле ввода с меткой «Фамилия». Затем — раскрывающийся список с меткой «Пол», в котором выбран вариант «Муж». Под ним — флажок с меткой «Выслать письмо-подтверждение», который отмечен галочкой. Внизу — поле ввода с меткой «Адрес». В нижнем правом углу находится кнопка с надписью «Отправить».

Рис. 23.14. Простейшее приложение, использующее элементы управления

Дадим элементам управления следующие имена:

- первое поле ввода — `txtName1`;
- второе поле ввода — `txtName2`;
- раскрывающийся список — `cboGender`;
- флажок — `chkSendMail`;
- третье поле ввода — `txtAddress`;
- кнопка — `btnOK`.

Теперь назначим раскрывающемуся списку такие значения параметров:

- **Row Count** — 2;
- **Labels** — «Муж» и «Жен»;
- **Data** — «m» и «f».

После этого назначим значения параметров для флажка:

- **Label** — «Выслать письмо-подтверждение»;
- **Initial Value** — `true`;
- **Change Handler** — «`onSendMailChange`».

И, напоследок, назначим значения параметров для кнопки:

- **Label** — «Отправить»;
- **Change Handler** — «`onOKClick`».

Закончив с формой для ввода данных, приступим к созданию приветствия. Приветствие будет находиться во втором кадре фильма. Создадим его и поместим на рабочий лист динамический текстовый блок по имени `txtOutput`. Зададим ему такие размеры, чтобы он занимал большую часть рабочего листа.

Теперь можно заняться сценариями. Все сценарии, используемые в этом приложении, будут привязаны к первому кадру фильма. Мы рассмотрим их по очереди.

Прежде всего, нам нужно определиться, где хранить данные, введенные пользователем в форму. Конечно, их можно поместить в переменные, но в этом случае данные не будут представлять собой единого целого. Давайте создадим для их хранения особый объект, который будет являться экземпляром объекта `Object`.

```
dataStorage = new Object();
```

Это выражение создает экземпляр объекта `Object`, который и послужит хранилищем наших данных.

Теперь напомним функцию `onSendMailChange`, которая будет обрабатывать включение или отключение флажка `chkSendMail`. Если флажок отключен, поле ввода `txtAddress` будет запрещено — ведь если пользователь не хочет получать от нас письмо, то и незачем спрашивать у него адрес.

```
function onSendMailChange(sender) {  
    _root.txtAddress.setEnabled(!_root.chkSendMail.getValue());  
}
```

Самая сложная функция — `onOKClick`. В ее обязанности входит поместить данные из элементов управления в объект `dataStorage`, перейти на второй кадр, сформировать текст приветствия и поместить его в динамический текстовый блок `txtOutput`.

```
function onOKClick(sender) {  
    with (dataStorage) {  
        name1 = _root.txtName1.text;  
        name2 = _root.txtName2.text;  
        gender = _root.cboGender.getValue();  
        sendMail = _root.chkSendMail.getValue();  
        address = _root.txtAddress.text;  
    }  
    _root.nextFrame();  
    with (dataStorage) {  
        txtOutput.text = "Здравствуйте, уважаем";  
        if (gender == "м") {  
            txtOutput.text += "ый ";  
        } else {  
            txtOutput.text += "ая ";  
        }  
        txtOutput.text += name1 + " " + name2 + "!";  
        if (sendMail) {
```

```
txtOutput.text += " На адрес '" + address +  
"'" + " вам отправлено письмо.";  
}  
}  
}
```

Осталась мелочь. А именно, остановить фильм на первом кадре.

```
_root.stop();
```

Теперь попробуйте запустить созданное приложение. Оно должно работать. А если не работает, значит, вы допустили какую-то ошибку. Тщательно проверьте код сценария и значения параметров всех элементов управления. И, самое главное, проверьте, задали ли вы имена элементам управления — это самая распространенная ошибка начинающих Flash-программистов.

Тонкая настройка компонентов

Как вы уже, наверно, заметили, набор параметров компонентов, которые вы можете менять, не очень велик. В него входят геометрические размеры компонента, текстовая надпись, список пунктов для списков, начальное состояние для флажков и переключателей и, разумеется, имя. Вы не можете менять ни цвет, ни параметры текста. Также нельзя изменить различные детали компонента, например, форму флажка или кнопки.

На самом деле, все это возможно. Но для этого вам придется узнать кое-что о стилях и "шкурах"; стили позволят вам изменить цвет, параметры текста и некоторые другие параметры компонентов, а "шкуры" — их внешний вид.

Настройка стилей компонентов

Стиль компонента — это особый объект, экземпляр объекта `FStyleFormat`, который служит для задания различных параметров компонента, недоступных из среды Flash. В основном, это цветовые настройки компонента, такие как цвет точки переключателя или цвет текста кнопки. Также, используя стиль, вы можете задать тип шрифта, его размер, величины отступов и некоторые другие параметры.

Создать экземпляр объекта `FStyleFormat` можно только в том случае, если на рабочий лист помещен хотя бы один компонент. Создается он очень просто, как и все экземпляры всех объектов.

```
myStyle = new FStyleFormat();
```

Полностью свойства этого объекта описаны в *приложении 1*. Здесь мы приведем описание только некоторых из них.

Итак, изменить цвет галочки флажка и точки переключателя вы можете с помощью свойств `check` и `radioDot` соответственно. Присвойте им нужное значение цвета.

Чтобы изменить цвет фона компонента, присвойте новый цвет свойству `background`. А с помощью свойства `face` вы можете изменить главный цвет компонента.

```
myStyle.background = 0x000000;  
myStyle.face = 0xFFFFFFFF;  
myStyle.check = 0xFFFFFFFF;  
myStyle.radioDot = 0xFFFFFFFF;
```

Приведенный выше фрагмент кода позволяет сделать элементы управления "негативными", т. е. белыми на черном фоне.

Для задания параметров текста служат особые свойства, часть которых перечислена ниже:

- ☐ `textFont` — задает тип шрифта для отображения текста;
- ☐ `textSize` — кегль шрифта в пунктах;
- ☐ `textColor` — цвет текста;
- ☐ `textAlign` — выравнивание текста;
- ☐ `textBold` — "жирность" шрифта;
- ☐ `textItalic` — курсивное начертание шрифта.

Например:

```
myStyle.textFont = "Verdana";  
myStyle.textSize = 14;  
myStyle.textItalic = true;
```

После задания всех параметров нового стиля вам нужно привязать его к элементам управления. Для этого воспользуйтесь уже знакомым методом `addListener`, который здесь выполняет не совсем типичную для него роль.

```
<Стиль>.addListener(<Список компонентов, разделенных запятыми>;  
myStyle.addListener(cboGender, btnOK);
```

Чтобы удалить привязку элемента управления к стилю, воспользуйтесь методом `removeListener`.

```
myStyle.removeListener(btnOK);
```

После этого вы можете изменять свойства объекта стиля, как вам угодно. Чтобы перенести эти изменения на привязанные к стилю элементы управления, вызовите метод `applyChanges`.

```
myStyle.applyChanged();
```

Чтобы удалить привязку элемента управления к стилю, воспользуйтесь методом `removeListener`.

```
myStyle.removeListener(btnOK);
```

Если вы хотите одновременно изменить параметры всех элементов управления, присутствующих на рабочем листе, то вам не нужно создавать новый

стиль. Для этого Flash предусматривает глобальный стиль `globalStyleFormat`, являющийся экземпляром объекта `FStyleFormat`, созданным самим Flash.

```
globalStyleFormat.textName = "Verdana";
```

```
globalStyleFormat.txtSize = 14;
```

```
globalStyleFormat.applyChanges();
```

И, наконец, если вы хотите изменить параметры только одного элемента управления, то воспользуйтесь методом `setStyleProperty`.

```
<Компонент>.setStyleProperty(<Имя свойства>, <Значение свойства>);
```

Пример использования этого свойства:

```
chkSendMail.setStyleProperty("check", 0x888888);
```

Изменение "шкур" компонентов

При создании встроенных элементов управления разработчики Flash использовали очень интересную технику — наложение "шкур" (или "скинов" от английского *skin* — шкура). В двух словах, *"шкура"* компонента — это набор графических изображений, каждое из которых используется в качестве одного из фрагментов элемента управления: галочки флажка, точки переключателя или стрелки раскрывающегося списка. Когда элемент управления собирается отобразить себя на листе, он загружает все изображения, составляющие "шкуру", и выводит их в нужных местах рабочего листа.

Создавать элементы управления, основанные на "шкурах", сложнее, но это позволяет менять их внешний вид заменой соответствующих изображений. Таким образом, вы можете заменить пресловутую галочку флажка, отредактировав соответствующее ей изображение или создав другое и привязав его к "шкуре".

Изображения, используемые для отображения элементов управления Flash, представляют собой обычные образцы-клипы. Каждый такой образец соответствует какому-либо фрагменту элемента управления. Поскольку таких изображений зачастую очень много, вы можете настроить внешний вид элементов управления очень точно. А как править образцы, вы узнали еще в *главе 10*.

Элементы управления Flash имеют еще одну интересную особенность. В качестве элементов "шкур" они используют не простые изображения, а сложные, представляющие собой готовые фрагменты, уже "собранные" из более простых фрагментов. Так, кнопка включает в себя фрагмент в виде прямоугольника (образец `fpb_disabled`), состоящий, в свою очередь, из пяти других, более простых фрагментов.

Но как найти элементы "шкур", соответствующие нужному элементу управления?

Рассмотрим структуру папок библиотеки, в которых хранятся элементы управления и их "шкуры". Все интересующие нас образцы выделим полужирным шрифтом.

Flash UI Components

  <Компонент>

   Component Skins

     <Компонент> Skins

       <Сложные элементы "шкур" компонента>

       Global Skins

         <Сложные элементы "шкур", используемых несколькими компонентами>

           Core Assets – Developer Only

             <Набор образцов и библиотек, используемых несколькими компонентами>

             Other Assets

               <Компонент> Assets

                 <Простые элементы "шкур">

Как видите, все используемые в "шкурах" элементы разбросаны по разным папкам. Это не отнюдь делает их поиск удобнее. Но что поделаешь — так распорядились разработчики фирмы Macromedia. Вероятно, так им было удобнее.

Найти нужный элемент "шкур" можно одним способом: просматривая все элементы "шкур", относящиеся к нужному элементу управления. Если вы владеете английским в достаточной степени, то можете искать элементы "шкур" по названиям.

Найдя требуемый элемент "шкур", исправьте его или перерисуйте заново. Когда будете его перерисовывать, постарайтесь не смещать изображение относительно точки фиксации образца, иначе "шкура" не будет правильно работать.

Чтобы увидеть все сделанные изменения, вам придется запустить фильм на проигрывание — в режиме создания фильма Flash их не покажет. Ну и, конечно, на рабочем листе должен присутствовать элемент управления, "шкуру" которого вы меняли.

Если вы добавили в сложную "шкуру" какой-либо новый фрагмент, то сможете заставить его менять цвет, когда разработчик, использующий ваш компонент, меняет значение одного из свойств объекта `FStyleFormat`. Для этого вам будет нужно привязать этому фрагменту нужное свойство. Такая операция выполняется вызовом метода `registerSkinElement` данного компонента.

```
<Компонент>.registerSkinElement(<Имя фрагмента>, <Свойство>);
```

Здесь все достаточно просто. Первым параметром этого метода передается имя фрагмента, к которому вы хотите привязать свойство объекта `FStyleFormat`. Имя самого этого свойства в виде строки передаете вторым параметром.

Откройте для правки какой-нибудь сложный элемент "шкур", например, кнопку открытия раскрывающегося списка `fsb_downArrow`. Выделите пер-

вый кадр слоя README и откройте панель **Actions**. Вы увидите список выражений вида:

```
component.registerSkinElement(arrow_mc, "arrow");  
component.registerSkinElement(face_mc, "face");  
component.registerSkinElement(shadow_mc, "shadow");
```

Эти выражения привязывают некоторое свойство объекта `FStyleFormat` к одному из фрагментов этой части компонента. Так, первое выражение привязывает свойство `arrow`, "отвечающее" за цвет стрелки, к фрагменту `arrow_mc`, являющемуся экземпляром образца `fc_b_arrow`. Просто добавьте в конце этого списка свое выражение такого же вида, и новый фрагмент "шкуры" станет "отзываться" на изменение стиля.

Кстати, вам следует соблюдать соглашения об именовании образцов и экземпляров, составляющих "шкуру". Так, имя образца должно начинаться с `fc_b_` (например, `fc_b_roundbutton`), а имя экземпляра должно заканчиваться `_mc` (`roundbutton_mc`). Следование этому простому правилу позволит соблюсти элементарный порядок в библиотеке.

Если вы хотите вернуть измененную вами "шкуру" какого-либо элемента управления к исходному состоянию, просто добавьте на рабочий лист еще один элемент управления того же типа. При этом Flash выдаст вам предупреждение, говорящее о том, что измененные вами образцы будут перезаписаны, включите переключатель **Replace Existing Items (Not Undoable)**. Если же вы все-таки хотите сохранить измененную "шкуру" и использовать ее в других элементах управления, включите переключатель **Use Existing Component**. После этого нажмите кнопку **ОК**, чтобы закрыть окно предупреждения.

Создание пользовательского элемента управления

Итак, с встроенными элементами управления мы покончили. Теперь выясним, как создаются пользовательские элементы управления.

Зачем создавать свои собственные элементы управления и компоненты вообще? Затем, зачем программисты всего мира пишут все новые и новые программы, несмотря на то, что их уже написано превеликое множество. Создателями всего нового движет неудовлетворенность текущим положением вещей, в том числе, встроенными компонентами Flash.

Вот и нами тоже движет такая неудовлетворенность. В *главе 22* мы создали регулятор громкости звука. Давайте превратим его в компонент, в пользовательский элемент управления, который может использоваться в любом приложении. И назовем этот компонент `SoundVolume`.

Пусть наш компонент имеет два параметра. Первый параметр, `Movie Clip` будет принимать имя клипа, громкость звука которого изменяет наш ком-

понт-регулятор. Если значение этого параметра равно пустой строке "", то регулятор будет менять громкость звукового сопровождения основного фильма. Второй параметр `Handler Size` будет задавать размер ручки регулятора.

Создадим новый документ Flash. И распишем все действия по созданию нового компонента шаг за шагом.

Создание образца-клипа

Это самый первый шаг по созданию нового компонента.

Но сначала нужно создать структуру папок библиотеки. Распределение различных образцов, составляющих компонент, по разным папкам уменьшит вероятность того, что какой-то образец совпадет по имени с другим, созданным разработчиком, использующим ваш компонент. (О папках и их преимуществах подробно было описано в *главе 10*.)

Не стоит нарушать традиции фирмы Macromedia. Поместим наш новый компонент, его основной образец, в папку `Flash UI Components`. А все вспомогательные образцы, используемые в компоненте, поместим в папку `Core Assets - Developer Only/Other Assets/SoundVolume Assets`. Теперь создадим все положенные папки и приступим к созданию образцов.

Прежде всего, создадим образец-клип, представляющий ручку регулятора. Назовем этот образец `fcv_volume`, как рекомендует Macromedia. Нарисуем в нем небольшой круг — и больше ничего. Проверим, находится ли точка фиксации образца точно в центре круга. Закроем полученный образец и поместим его в папку `Core Assets - Developer Only/Other Assets/SoundVolume Assets`, вложенную в папку `Flash UI Components`.

Теперь создадим еще один образец-клип, который станет основным образцом нашего компонента. Назовем его `SoundVolume`. Нарисуем в нем шкалу и поместим на эту шкалу только что нарисованный нами вспомогательный клип-ручку. Назовем экземпляр вспомогательного клипа-ручки `volume_mc` и преобразуем его в кнопку, т. к. нам понадобится обрабатывать некоторые "кнопочные" события. Переместим все изображение так, чтобы точка фиксации приходилась на место пересечения среднего деления шкалы и самой шкалы, и поместим туда ручку регулятора. Назовем единственный пока слой клипа `scale`. Закроем готовый клип и поместим его в папку `Flash UI Components`.

Результат наших трудов показан на рис. 23.15. Сохраним его на диске.

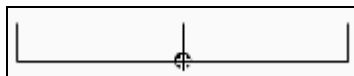


Рис. 23.15. Внешний вид компонента `SoundVolume`

Написание сценариев ActionScript

После создания всей графики можно приступить и к написанию сценариев.

Снова откроем образец-клип `SoundVolume`. Создадим новый слой под именем `actions`. К первому его кадру привяжем такой несложный сценарий (рассмотрим его по строкам):

```
clipSound = new Sound();
```

Здесь мы создаем экземпляр объекта `Sound`, который будет управлять звуковым сопровождением всего фильма.

```
volume_mc._width = volume_mc._height = 10;
```

Задаем значения по умолчанию для ширины и высоты ручки регулятора.

```
volume_mc._x = this._width;
```

Устанавливаем ручку на значение, соответствующее максимальной громкости.

```
this.onEnterFrame = function() {  
    clipSound.setVolume(volume_mc._x / this._width * 100);  
}
```

Ну, а здесь мы, собственно, и выполняем изменение громкости фильма.

Следующий сценарий привяжем к ручке регулятора `volume_mc`. Рассмотрим его (сценарий, а не регулятор) по частям.

```
on(press) {  
    this.startDrag(false, 0, this._x, _parent._width, this._x);  
}
```

Этот обработчик разрешает ручке перемещаться, если пользователь "ухватится" за нее мышью. Обратите внимание на параметры метода `startDrag` — мы разрешаем ручке двигаться только по шкале.

```
on(release) {  
    this.stopDrag();  
}
```

Этот обработчик завершает процесс перемещения, если пользователь отпустит кнопку мыши.

Собственно, весь этот код вам уже знаком. Мы рассматривали его в *главе 22*, когда впервые создавали регулятор громкости. Правда, тогда он не был компонентом...

Вы уже можете проверить созданный нами компонент в работе. (Точнее, компонентом его пока что можно назвать с большой натяжкой, т. к. он еще не имеет ни параметров, ни иконки, ни описания.) Закройте все образцы, которые вы открыли для правки, поместите на рабочий лист экземпляр образца `SoundVolume`, назовите его как-нибудь и запустите фильм на проигрывание. Попробуйте перемещать ручку регулятора — она движется! Для очи-

стки совести можете привязать к фильму какой-нибудь звук и произвести окончательные испытания компонента.

Создание параметров компонента

Графика и код готовы и работают. Теперь создадим параметры компонента.

Как вы помните, наш компонент будет иметь два параметра:

- ❑ **Movie Clip** — задает клип, громкость звукового сопровождения которого будет регулировать наш компонент;
- ❑ **Handler Size** — задает размер ручки регулятора.

Но задавать параметры еще рано. Нужно изменить код компонента так, чтобы он использовал их значения. Тот код, что мы написали ранее, этого не делает.

К первому кадру слоя `actions` образца-клипа `SoundVolume` у нас был привязан сценарий. Этот сценарий содержал выражение

```
clipSound = new Sound();
```

создающее экземпляр объекта `Sound`, управляющего звуком, и

```
volume_mc._width = volume_mc._height = 10;
```

задающее размер по умолчанию ручки регулятора.

Заменим эти выражения следующими:

```
if (movieClipName == "") {  
    clipSound = new Sound();  
} else {  
    clipSound = new Sound(movieClipName);  
}  
  
volume_mc._width = volume_mc._height = handlerSize;
```

Здесь переменная `movieClipName` задает имя клипа, а переменная `handlerSize` — размер ручки регулятора.

Теперь добавим еще два метода, позволяющие управлять размером ручки регулятора из сценариев. Это методы `getHandlerSize` и `setHandlerSize`, первый из которых возвращает размер ручки, а второй — задает его. Поместим их объявление в сценарий, привязанный к первому кадру слоя `actions`. Описывать их код мы не будем — он и так достаточно прост.

```
function getHandlerSize() {  
    return handlerSize;  
}  
  
function setHandlerSize(hs) {  
    handlerSize = hs;  
    volume_mc._width = volume_mc._height = handlerSize;  
}
```

Вот теперь мы готовы к созданию параметров.

Для создания параметров мы будем использовать диалоговое окно **Component Definition**. Чтобы вывести его на экран, выделите в списке окна библиотеки образец, являющийся нужным компонентом, и выберите пункт **Component Definition** в контекстном или дополнительном меню. Само диалоговое окно **Component Definition** показано на рис. 23.16.

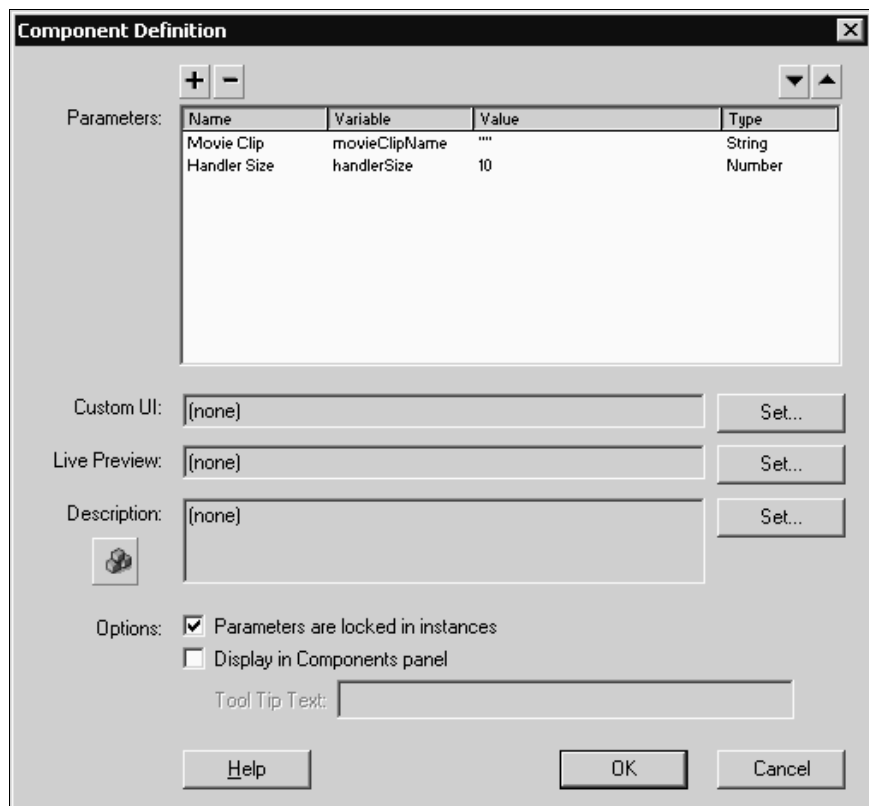


Рис. 23.16. Диалоговое окно **Component Definition**

В данный момент нас интересует большой список **Parameters**, расположенный в верхней половине этого диалогового окна. Он представляет собой таблицу из четырех колонок:

- ☐ **Name** — имя параметра, которое будет отображаться в редакторе свойств и панели **Component Parameters**;
- ☐ **Variable** — имя переменной, с которой связан этот параметр;
- ☐ **Value** — значение этого параметра по умолчанию;
- ☐ **Type** — тип данных параметра.

Имя параметра не обязательно должно удовлетворять правилам именования переменных. Так, например, оно может содержать пробелы. Собственно, вы уже это выяснили, когда работали со встроенными компонентами — многие из их параметров имеют в своих именах пробелы.

Тип данных параметра выбирается в раскрывающемся списке, который содержит следующие пункты:

- ☐ **Default** — Flash сам пытается определить тип данных. При этом значения `true` и `false` относятся к логическим, числа — к числовым, остальные — к строковым;
- ☐ **Array** — массив;
- ☐ **Object** — экземпляр объекта;
- ☐ **List** — массив строк;
- ☐ **String** — строковый;
- ☐ **Number** — числовой;
- ☐ **Boolean** — логический;
- ☐ **Font Name** — имя шрифта;
- ☐ **Color** — цвет.

Лучше всегда самим выбирать нужный тип данных, чем позволять это сделать Flash. Хотя Flash и делает это в большинстве случаев довольно логично, но хороший программист все же старается держать контроль над типами данных в своих руках.

Чтобы добавить в список новый параметр, щелкните кнопку с изображением знака "плюс". В списке **Parameters** появится новая строка. Щелкните по этой строке в колонке **Name** списка. После этого там появится небольшое поле ввода, в которое вы сможете ввести имя параметра. Введя его, нажмите клавишу <Enter> для сохранения или клавишу <Esc> для отмены ввода. Так же введите значения в колонки **Variable** и **Value**.

Чтобы задать тип параметра, щелкните по строке в колонке **Type**. После этого там появится раскрывающийся список, в котором вы сможете выбрать нужный параметр. Имейте в виду, что после выбора типа параметра Flash сбросит значение, введенное вами в колонке **Value**. Так что, наверно, сначала будет лучше выбрать тип данных параметра, а потом уже задать его значение по умолчанию.

Чтобы удалить ненужный параметр, выделите его в списке **Parameters** и нажмите кнопку с изображением знака "минус". Параметры удаляются без предупреждения, так что будьте внимательны.

Кнопки с изображением стрелок позволяют вам перемещать параметры в списке вверх и вниз, изменяя порядок их следования.

Закончив ввод параметров, нажмите кнопку **ОК**, чтобы сохранить их и закрыть окно **Component Definition**. Если же вы не хотите сохранять введенные параметры, нажмите кнопку **Cancel**.

Обратите внимание, что после того, как вы введете для образца-клипа хоть один параметр, его условное изображение в списке окна библиотеки меняется (рис. 23.17). После этого образец становится *образцом-компонентом* или просто компонентом.




Name	Kind	U
 Flash UI Components	Folder	
 Core Assets - Dev...	Folder	
 SoundVolume	Component	

Рис. 23.17. Образец-компонент в окне библиотеки (выделен)

Создание значка компонента

А теперь мы зададим для компонента значок. Этот значок будет появляться в списке образцов окна библиотеки и в панели **Components**.

Выбрать значок, который представит наш компонент в окне библиотеки, сравнительно несложно. Откройте диалоговое окно **Component Definition**. Под текстовой надписью **Description** отыщите небольшую кнопку с картинкой и нажмите ее. На экране появится довольно большое меню (рис. 23.18).

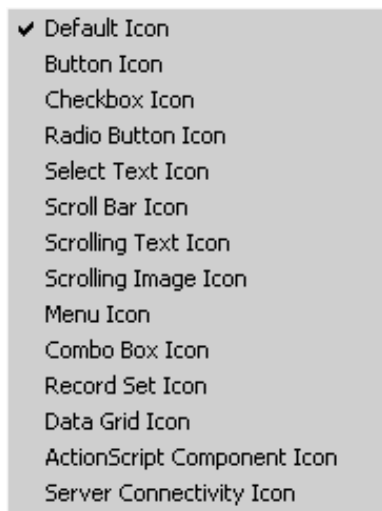


Рис. 23.18. Меню значков окна **Component Definition**

Здесь, правда, не присутствуют сами значки, так что придется по очереди выбирать каждый пункт-переключатель, пока не подберете подходящий. Выбор значков довольно мал, поэтому зачастую лучше всего будет оставить значок по умолчанию (пункт **Default Icon**). Или создать свой собственный значок.

А свой собственный значок создать несколько сложнее. Вам придется нарисовать его самим, воспользовавшись программой растровой графики, например, Microsoft Paint, поставляемым в составе Windows. Но, поскольку вы уже достаточно хорошо владеете Flash, то проще сделать это именно в нем. Имейте в виду, что значок должен иметь размеры 20×20 пикселей. Сохраните полученное изображение в формате BMP, GIF, JPEG или PNG и импортируйте его в библиотеку, содержащую наш компонент. После этого создайте в папке Flash UI Components библиотеки папку FCustomIcons, поместите туда импортированный значок и дайте ему то же имя, что и компоненту.

Теперь вам останется только закрыть окно библиотеки и открыть его снова. И полюбоваться на результат, показанный на рис. 23.19.

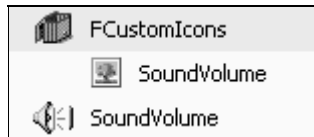


Рис. 23.19. Значок компонента (вверху — папка FCustomIcons и импортированное растровое изображение компонента SoundVolume)

Создание описания компонента

Теперь мы добавим текстовое описание компонента, которое будет отображаться в панелях **Actions** и **Reference**.

Такое описание может быть создано в двух форматах: обычном текстовом и XML. Описание в обычном текстовом формате создать проще всего, но описание в формате XML выглядит лучше и профессиональнее. Здесь мы создадим оба описания, сначала в обычном текстовом формате, потом в формате XML.

Создать текстовое описание компонента очень просто. Вызовите на экран диалоговое окно **Component Definition**. Щелкните кнопку **Set**, расположенную правее области редактирования **Description**. В появившемся на экране диалоговом окне **Component Description** (рис. 23.20) включите переключатель **Description is plain text** и введите текст описания в область редактирования **Description**. Закончив ввод, нажмите кнопку **OK**, чтобы сохранить текст описания, или **Cancel**, чтобы отменить его. После этого вам останется закрыть диалоговое окно, нажав кнопку **OK** или **Cancel**.

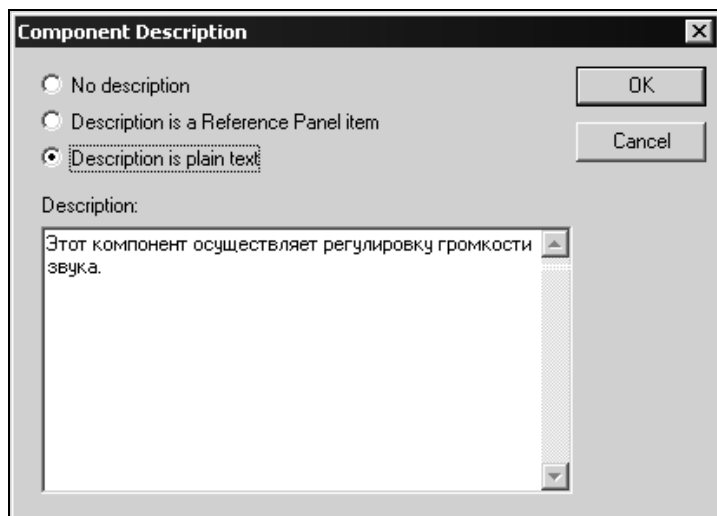


Рис. 23.20. Диалоговое окно **Component Description**
(включен переключатель **Description is plain text**)

После того, как вы закончите создание компонента, экспортируете его и подключите к среде Flash, введенное вами описание будет присутствовать в иерархическом списке панели **Reference**. Вам будет нужно только выбрать в ветви **Flash UI Components** иерархического списка, расположенного в левой части этой панели, созданный вами компонент, и в правой части появится его описание.

Если вы захотите убрать описание компонента, то включите переключатель **No description** в диалоговом окне **Component Description**. И, разумеется, не забудьте нажать кнопку **OK**.

Создав простое текстовое описание, вы добились того, что ваш компонент будет присутствовать в списке панели **Reference**. Если вы хотите, чтобы в иерархическом списке панели **Actions** отображался список методов вашего компонента, то нужно создать описание в формате XML. Сейчас мы рассмотрим, как оно делается.

Но сначала несколько слов о языке XML. XML — это универсальный язык описания данных в текстовом виде. Его особенностью является то, что данные форматируются с использованием тегов, как и в HTML. Однако набор тегов языка XML не фиксирован жестко, как в HTML, а задается самим разработчиком, что позволяет кодировать с его помощью любые данные. Что мы и увидим далее.

Внимание!

Имейте в виду, что язык XML очень требователен к правильности написания кода. Если многие программы Web-обозревателей снисходительно прощают

некоторые ошибки кодирования HTML, то в случае с XML такие "номера" не пройдут. Также XML чувствителен к регистру символов, т. е. `<STRING>` и `<string>` — разные теги.

Описание в формате XML создается в текстовом редакторе и сохраняется в обычном текстовом файле с расширением `.xml`. Этот файл помещается в подкаталог `Application Data/Macromedia/Flash MX/Configuration/Action Panels/Custom Actions`. Операционные системы Windows 95/98/Me "прячут" этот подкаталог в папке `Windows`, а операционные системы Windows — в папке пользовательского профиля.

Список методов (а также свойств, событий и внутренних объектов, но мы будем говорить только о методах) компонента и их описания помещаются внутрь парного тега `<customactions></customactions>`. Список методов заключается внутрь парного тега `<actionspanel>...</actionspanel>`, а их описания — внутрь тега `<reference>...</reference>`; все эти теги вкладываются внутрь тега `<customactions></customactions>`. Как видите, теги имеют достаточно "прозрачные" имена, говорящие, в какой панели будет отображаться их содержимое: **Actions** или **Reference**.

Написание текстов для панели **Reference** мы рассмотрим чуть ниже. Сейчас же давайте сосредоточимся на панели **Actions**.

Вы, конечно же, помните, что список в левой части панели **Actions** имеет иерархическую структуру, т. е. состоит из множества ветвей, в которые вложены пункты. Для создания ветви служит парный тег `<folder>...</folder>`. Его формат таков:

```
<folder name="<Название ветви>" tiptext="<Текст подсказки>">
    . . .
</folder>
```

Название ветви, задаваемое атрибутом `name`, появится в иерархическом списке панели **Actions**, а текст подсказки, заданный атрибутом `tiptext`, в строке подсказки этой панели, но не в панели **Reference**. Текст для панели **Reference** мы зададим потом.

Теги `<folder>` можно вкладывать друг в друга, создавая вложенные ветви и подветви.

Для создания пункта иерархического списка вам нужно использовать тег `<string>`. Этот тег, можно сказать, одинарный, но т. к. в языке XML, по правилам, не должно быть одинарных тегов, в его конце просто добавляется символ косой черты — `<string/>`. Формат этого тега таков:

```
<string name="<Название пункта>" tiptext="<Текст подсказки>"
    ⚡text="<Текст операции>"/>
```

Здесь нужно поговорить о тексте операции, задаваемом атрибутом `text`. Этот атрибут представляет собой формат вызова метода. В случае метода

`getHandlerSize` он будет таков: `"getHandlerSize()"`. Если же метод имеет параметры, как, например, `setHandlerSize`, то эти параметры перечисляются внутри скобок и выделяются знаками процента: `"setHandlerSize(% handlerSize %)"`. Встретив в тексте такое определение, Flash автоматически высветит подсказку по коду.

Теперь поговорим о теге `<reference>`, с помощью которого кодируются тексты для панели **Reference**. Как вы помните, эти теги также вкладываются в тег `<customactions></customactions>`. Формат тега `<reference>` таков:

```
<reference path="<Путь к ветви или пункту списка>">  
    . . . Текст описания в формате HTML  
</reference>
```

Атрибут `path` задает путь к ветви или пункту иерархического списка, описание для которого вы хотите создать. Этот путь имеет вид "Ветвь 1/Ветвь 2/Пункт".

Сам текст описания вводится в формате HTML. В нем вы можете использовать все поддерживаемые Flash HTML-теги:

- ☐ `<A>...` (гиперссылка);
- ☐ `...` (полужирный шрифт);
- ☐ `...` (задание цвета шрифта);
- ☐ `...` (задание шрифта);
- ☐ `...` (задание размера шрифта);
- ☐ `<I>...</I>` (курсив);
- ☐ `<P>...</P>` (отдельный абзац текста в многострочном поле ввода);
- ☐ `<U>...</U>` (подчеркивание).

Для выделения заголовка вы можете использовать стиль `titleStyle`.

А теперь приведем пример XML-файла, описывающего наш компонент `SoundVolume`. Мы рассмотрим его по частям.

```
<customactions>
```

Главный тег нашего файла. Все описание компонента помещается в него.

```
<actionspanel>
```

Этим тегом начинается описание содержимого панели **Actions**.

```
<folder name="Flash UI Components" tiptext="Flash UI Components">
```

Мы поместим наш компонент в ветвь `Flash UI Components`, чтобы не нарушать структуру списка.

```
<folder name="SoundVolume" tiptext="Sound volume component">
```

Эта ветвь задает наш компонент.

```
<folder name="Methods" tiptext="SoundVolume methods">
```

В этой ветви находится описание обоих методов нашего компонента.

```
<string name="getHandlerSize" tiptext="Returns a handler size."
    ⚡text=".getHandlerSize()"/>
<string name="setHandlerSize" tiptext="Sets a handler size."
    ⚡text=".setHandlerSize(% size %)/>
```

А это — описание методов.

```
</folder>
</folder>
</folder>
</actionspanel>
```

С панелью **Actions** покончено. Не забываем закрыть все парные теги, кроме `<customactions>`.

Со следующего тега начнутся описания, отображаемые в панели **Reference**. Заметьте, что они тоже вложены в тег `<customactions>`.

```
<reference path="Flash UI Components/SoundVolume">
```

Это описание самого компонента. Обратите внимание на путь, заданный атрибутом `path`, и сравните его со структурой ветвей, заданной в теге `<actionspanel>`.

```
<p class="titleStyle">Sound volume component</p>
<p>Allows to adjust a sound volume.</p>
```

А это текст описания в формате HTML. Как видите, для заголовка мы использовали стиль `titleStyle`.

```
</reference>
<reference path="Flash UI
    ⚡Components/SoundVolume/Methods/getHandlerSize">
    <p class="titleStyle">getHandlerSize method</p>
    <p>Returns a handler size.</p>
</reference>
<reference path="Flash UI
    ⚡Components/SoundVolume/Methods/setHandlerSize">
    <p class="titleStyle">setHandlerSize method</p>
    <p>Sets a handler size.</p>
</reference>
```

Конец описаний для панели **Reference**.

```
</customactions>
```

Конец файла.

Сохраните пока созданный вами XML-файл описания компонента. Ниже мы расскажем, что с ним делать.

Очень странная вещь: в диалоговом окне **Component Description** предусмотрен переключатель **Description is a Reference Panel Item**, выводящий в нижней части этого окна иерархический список панели **Reference**. Но зачем это нужно, не знают, наверно, даже разработчики Flash. Вроде бы, это должно помочь при создании описания в формате XML, но реально ничуть не помогает. Еще одна загадка Flash MX

Создание клипа "живого просмотра"

Когда пользователь вводит новое значение для какого-либо параметра компонента, компонент, помещенный на рабочий лист, никак на это не "отзывается". Так, если мы поместим на лист экземпляр компонента `SoundVolume` и зададим новое значение размера ручки регулятора **Handler Size**, то сам размер ручки не изменится. Вам придется запустить проигрывание фильма, чтобы увидеть это.

Прямого пути сделать компонент "отзывчивым" нет. Для этого потребуется создать особый клип, называемый *клипом "живого просмотра"* (в терминологии Flash — "Live Preview"). Flash выведет этот клип на рабочий лист вместо выделенного экземпляра компонента, если пункт-выключатель **Enable Live Preview** в меню **Control** включен. (Если же этот пункт отключен, то на листе будет отображаться сам компонент, конечно же, "не отзывчивый".) Все значения параметров, введенные вами, будут передаваться в этот клип, а сам клип будет показывать, что произойдет с компонентом при вводе того или иного значения параметра.

Задача эта отнюдь не так сложна, как вы представляете. А единственная трудность — клип "живого просмотра" не может быть создан в том же самом документе, что и компонент — вполне преодолима.

Чтобы реализовать клип "живого просмотра", сначала сделайте копию вашего документа Flash, где находится создаваемый компонент. Проще всего для этого выбрать пункт **Save As** в меню **File** или нажать комбинацию клавиш <Ctrl>+<Shift>+<S>. Затем введите имя нового документа в стандартном диалоговом окне сохранения файла Windows и нажмите кнопку сохранения.

После этого откройте новый документ в среде Flash. Выведите на экран окно библиотеки и удалите все ненужное. А именно, параметры компонента (после этого компонент `SoundVolume` превратится в обычный образец-клип), папку `FCustomIcons` библиотеки и ее содержимое, сценарии, привязанные к компоненту. Здесь все это нам не понадобится, а размер документа увеличивается сильно.

Создайте экземпляр образца-клипа `SoundVolume` на рабочем листе. Назовите его, например, `soundVolume`. Затем переместите клип в левый верхний угол рабочего листа. Проще всего сделать это, введя нули в поля ввода **X** и **Y**, расположенные в левой части редактора свойств. Напоследок выделите его,

прочитайте в полях ввода **W** и **H** редактора свойств значения ширины и высоты клипа, вызовите диалоговое окно **Document Properties**, нажмите кнопки **Content** и **ОК**. Таким образом мы зададим размеры рабочего листа, равные размерам его содержимого, т. е. клипа `soundVolume`.

Теперь создайте в библиотеке новый "пустой" образец-клип. Назовите его `xch` (именно так, не иначе — это важно!). Создайте на рабочем листе экземпляр этого клипа и назовите его так же — `xch`. Этот клип станет "посредником" по передаче данных от среды Flash к клипу "живого просмотра".

После этого создайте новый слой и назовите его `actions`. Присвойте первому кадру этого слоя следующий сценарий (но не изменяйте имя функции!):

```
function onUpdate() {  
    with (soundVolume.volume_mc) {  
        _height = _width = xch.handlerSize;  
    }  
}
```

Этот сценарий просто берет из свойства `handlerSize` клипа-"посредника" `xch` значение и присваивает его ширине и высоте ручки регулятора `volume_mc`. Свойство `handlerSize` в клипе `xch` создается и заполняется значением самой средой Flash.

Собственно, это все. Сохраните готовый документ и экспортируйте его в распространяемый формат Shockwave/Flash. После этого можете его закрыть — он нам больше не понадобится.

Переключитесь в документ, где находится компонент `SoundVolume`. Откройте диалоговое окно **Component Definition** (см. рис. 23.16). Щелкните кнопку **Set**, расположенную правее поля ввода **Live Preview**. На экране появится диалоговое окно **Live Preview** (рис. 23.21).

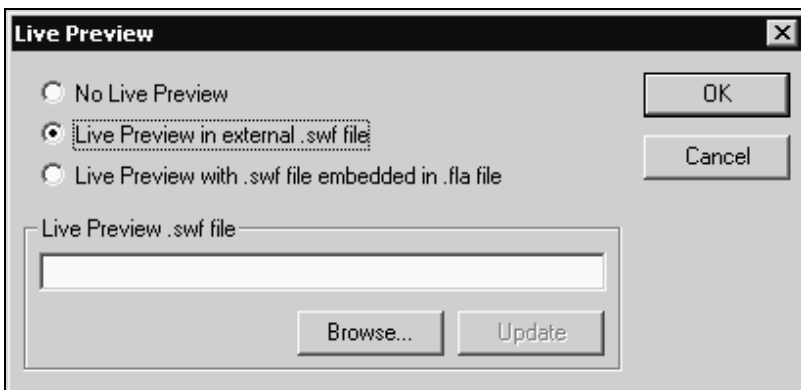


Рис. 23.21. Диалоговое окно **Live Preview**

Flash предоставляет вам две возможности привязать к компоненту клип "живого просмотра". Во-первых, вы можете внедрить файл с клипом "живого просмотра" в файл документа, где находится компонент, это позволит уменьшить количество файлов на диске. Во-вторых, вы можете привязать клип "живого просмотра" к компоненту, не внедряя его в файл документа. Вам решать, какой возможностью воспользоваться.

Если вы хотите внедрить клип "живого просмотра", включите переключатель **Live Preview with .swf file embedded in .fla file**. Если вы не хотите его внедрять, включите переключатель **Live Preview in external .swf file**. (Переключатель **No Live Preview** отключает "живой просмотр".) После этого становится доступным поле ввода **Live Preview .swf file**, где вы должны будете ввести имя файла, содержащего клип "живого просмотра". Можно также нажать кнопку **Browse** и выбрать нужный файл в появившемся на экране стандартном диалоговом окне открытия файла Windows.

Задав клип "живого просмотра", нажмите кнопку **ОК**, чтобы сохранить его. Не забудьте после этого также нажать кнопку **ОК** диалогового окна **Component Definition**. Если же вы решили отменить ваш ввод, нажмите кнопку **Cancel**.

Если вы изменили клип "живого просмотра" после того, как внедрили его в документ компонента, то откройте диалоговое окно **Live Preview** и нажмите кнопку **Update**. И не забудьте после этого нажать кнопки **ОК** в обоих открытых диалоговых окнах. После того, как вы это сделаете, Flash обновит внедренный файл. Заметьте, что кнопка **Update** доступна только тогда, когда включен переключатель **Live Preview with .swf file embedded in .fla file**.

После этого сохраните документ. И можете, в принципе, проверить, как работает "живой просмотр". Для этого поместите на рабочий лист экземпляр компонента и введите новое значение параметра **Handler Size**. Если размер ручки регулятора не изменился, проверьте, включен ли пункт-выключатель **Enable Live Preview** в меню **Control**.

Создание панели параметров

Для ввода параметров компонента служат редактор свойств и панель **Component Parameters**. Однако Flash позволяет создать для компонента еще и так называемую *панель параметров*, которая будет использоваться вместо обычного списка параметров. Эта панель будет содержать набор элементов управления для ввода значений.

Вообще-то, нам представляется, что панели параметров — излишество в среде Flash MX. Они перешли в новую версию из Flash 5, где не существовало "законного" средства для задания значений параметров. В Flash MX есть редактор свойств и панель **Component Parameters**, которых зачастую вполне достаточно. Другое дело, если вы хотите сделать для ввода параметров вашего компонента какой-нибудь нестандартный интерфейс.

Как и клип "живого просмотра", панель параметров реализуется в отдельном файле. Поэтому создайте новый документ Flash. Поместите на рабочий лист динамический текстовый блок и две кнопки `PushButton` — мы сделаем поле со счетчиком. Измените размеры рабочего листа так, чтобы он вмещал только созданные вами элементы управления. То, что у вас должно получиться, показано на рис. 23.22.

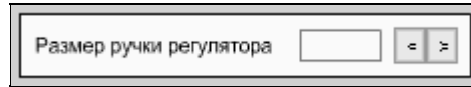


Рис. 23.22. Панель параметров для компонента `SoundVolume`

Теперь зададим параметры динамического текстового блока:

- имя — `txtHS`;
- переменная — `xch.handlerSize`.

Как видите, мы привязываем динамический текстовый блок к свойству `handlerSize` объекта-"посредника" `xch`. Значение этого свойства задает среда Flash.

Параметры левой кнопки:

- имя — `btnDec`;
- Label — `"<"`;
- Click Handler — `"onBtnDecClick"`.

Параметры правой кнопки:

- имя — `btnInc`;
- Label — `">"`;
- Click Handler — `"onBtnIncClick"`.

После этого создайте новый слой и назовите его `actions`. Присвойте первому кадру этого слоя следующий несложный сценарий:

```
function onBtnDecClick() {
    --xch.handlerSize;
}
function onBtnIncClick() {
    ++xch.handlerSize;
}
```

Эти функции просто уменьшают или увеличивают значение свойства `hs` объекта `xch` на единицу в ответ на щелчки по кнопкам. Больше никакого кода в сценарии не будет.

Осталось сформировать объект-"посредник". Создайте новый "пустой" образец-клип и назовите его `xch` (именно так, не иначе — это важно!). Создайте на рабочем листе экземпляр этого клипа и назовите его так же — `xch`.

На этом работа над панелью параметров закончена. Сохраните готовый документ и экспортируйте его в распространяемый формат Shockwave/Flash. После этого можете его закрыть — он нам больше не понадобится.

Переключитесь в документ, где находится компонент `SoundVolume`. Откройте диалоговое окно **Component Definition** (см. рис. 23.16). Щелкните кнопку **Set**, расположенную правее поля ввода **Custom UI**. На экране появится диалоговое окно **Custom UI** (рис. 23.23).

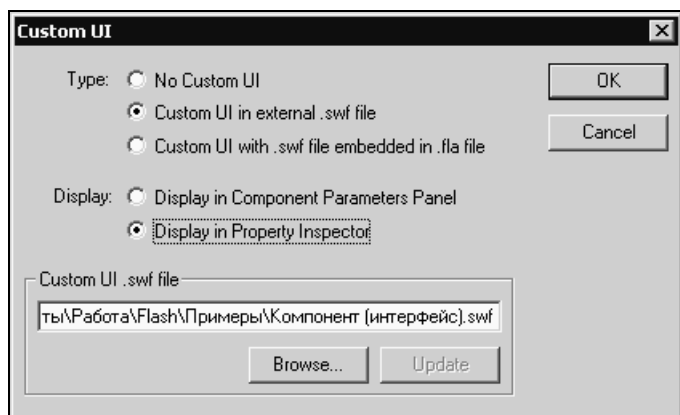


Рис. 23.23. Диалоговое окно **Custom UI**

В этом случае Flash также предоставляет вам две возможности привязать к компоненту панель параметров. Во-первых, вы можете внедрить файл с панелью параметров в файл документа, где находится компонент. Во-вторых, вы можете привязать файл панели к компоненту, не внедряя его в файл документа.

Если вы хотите внедрить панель параметров, включите переключатель **Custom UI with .swf file embedded in .fla file** в группе **Type**. Если вы не хотите ее внедрять, включите переключатель **Custom UI in external .swf file**. (Переключатель **No Custom UI** убирает панель параметров.) После этого становится доступным поле ввода **Custom UI .swf file**, где вы должны будете ввести имя файла, содержащего панель параметров. Также можно нажать кнопку **Browse** и выбрать нужный файл в появившемся на экране стандартном диалоговом окне открытия файла **Windows**.

Если вы хотите, чтобы панель параметров отображалась в редакторе свойств, включите переключатель **Display in Property Inspector** в группе **Display**. Это стоит делать, если ваша панель параметров имеет небольшие размеры. Если же она слишком велика для редактора свойств, включите переключатель **Display in Component Parameters Panel**. После этого панель параметров будет отображаться внутри панели **Component Parameters**.

Задав нужные параметры, нажмите кнопку **ОК**, чтобы сохранить их. Не забудьте после этого также нажать кнопку **ОК** диалогового окна **Component Definition**. Если же вы решили отменить ваш ввод, нажмите кнопку **Cancel**.

Если вы изменили панель параметров после того, как внедрили ее в документ компонента, то откройте диалоговое окно **Custom UI** и нажмите кнопку **Update**. И не забудьте после этого нажать кнопки **ОК** в обоих открытых диалоговых окнах. После того, как вы это сделаете, Flash обновит внедренный файл. Заметьте, что кнопка **Update** доступна только тогда, когда включен переключатель **Custom UI with .swf file embedded in .fla file**.

После этого сохраните документ. И можете проверить панель параметров в работе. Для этого поместите на рабочий лист экземпляр вашего компонента. Flash тотчас выведет панель параметров, либо в редакторе свойств (рис. 23.24), либо в панели **Component Parameters** (рис. 23.25), в зависимости от заданных вами настроек. Причем, если вы указали Flash отображать панель параметров внутри панели **Component Parameters**, в редакторе свойств появится кнопка **Launch Component Parameters Panel**, при нажатии которой на экран будет выведена панель **Component Parameters**.

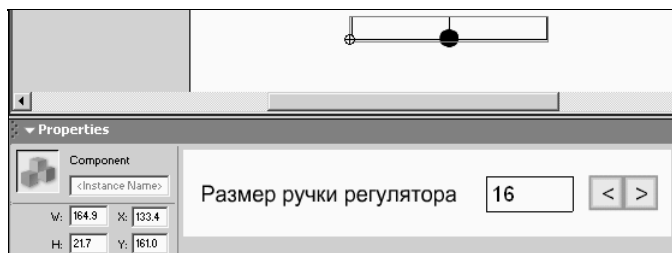


Рис. 23.24. Панель параметров, отображаемая в редакторе свойств



Рис. 23.25. Панель параметров, отображаемая в панели **Component Parameters**

Последние штрихи

Осталось сделать совсем немного.

Откройте диалоговое окно **Component Definition** (см. рис. 23.16). Здесь находятся еще три элемента управления, о которых мы еще не говорили.

Флажок **Parameters are locked in instances**, будучи включенным (а он включен по умолчанию), запрещает разработчикам, использующим ваш компонент, добавлять и удалять его параметры в окне библиотеки. Вероятно, стоит оставить этот флажок включенным.

Флажок **Display in Components panel** включает или отключает показ компонента в панели **Components**. Его лучше включить, если, конечно, у вас на этот компонент нет других планов.

Если включить флажок **Display in Components Panel**, становится доступно поле ввода **Tool Tip Text**. Введите в это поле текст, который будет выводиться во всплывающей подсказке, если пользователь задержит курсор мыши над значком компонента в панели **Components**.

Задав нужные настройки, не забудьте нажать кнопку **ОК**. Если же вы хотите их отменить, нажмите кнопку **Cancel**.

Установка компонента во Flash

Вот и подошел к концу долгий процесс создания нового компонента. Теперь мы имеем документ Flash, библиотека которого этот компонент содержит. Что с ним делать дальше?

А теперь наш компонент нужно установить в среду Flash. Мы опишем процесс установки по шагам.

1. Сохранить готовый документ и выйти из среды Flash. Проверить, готов ли XML-файл с описанием компонента (если, конечно, вы его создавали).
2. Скопировать файл документа Flash в подкаталог Application Data\Macromedia\Flex MX\Configuration\Components. Если вы работаете в среде Windows 95/98/Me, то этот подкаталог находится в каталоге Windows, если же вы предпочитаете Windows 2000/XP, то сможете найти данный подкаталог в каталоге пользовательского профиля. Также не забудьте поместить в этот же каталог все сопутствующие файлы, как-то: клип "живого просмотра" и панель параметров.
3. Скопировать файл описания компонента (если он есть) в подкаталог Application Data\Macromedia\Flex MX\Configuration\Action Panels/Custom Actions.
4. Запустить Flash и проверить работу компонента.

Вот и все! Результат наших трудов вы можете видеть на рис. 23.26. Как видите, в списке страниц панели **Components** появилась новая страница —

Компонент. Именно так автор назвал файл документа Flash, в котором находился новый компонент, — Компонент.fla. Из этого следует, что файл, в котором содержатся компоненты, дает имя странице этой панели.

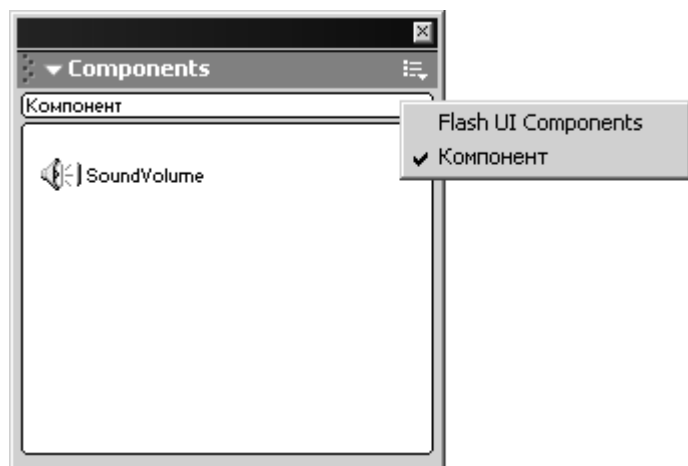
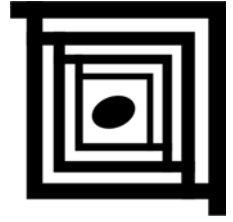


Рис. 23.26. Новый компонент на новой странице панели **Components**



Глава 24

Работа с внешними приложениями

Когда-то давно, лет двадцать назад, компьютеры были однозадачными. Это означало, что на одном отдельно взятом компьютере в данный момент времени могла выполняться только одна программа. Пользователь запускал эту программу, работал в ней, сохранял результат, завершал программу, запускал другую, опять работал в ней, опять сохранял результат, опять выходил и. т. п. Конечно, это было неудобно, но так в то время работали. И, надо сказать, не очень-то жаловались.

Почему? Во-первых, компьютеры двадцатилетней давности были бесконечно слабее современных и просто не могли потянуть даже две одновременно работающих более-менее серьезных программы. Во-вторых, программное обеспечение было принципиально однозадачным, оно просто не могло позволить пользователю запустить две программы, за редкими исключениями. В-третьих, кому это было нужно: техника делает вид, что работает, пользователи делают вид, что довольны, а значит, зачем напрягаться...

Конечно, существовали способы запускать на тогдашней технике сразу несколько программ, но они налагали на программы очень серьезные ограничения. Было даже что-то подобное настоящим многозадачным операционным системам (например, первая версия Microsoft Windows). Но все это так и не получило широкого распространения. Почему? Ответ на этот вопрос содержится в предыдущем абзаце.

Потом появилась Windows 3.0, за ней — 3.1, 3.11, 95... Компьютеры существенно прибавили в мощности, а программисты наконец-то поняли, что принцип "один компьютер — одна программа" не так уж и хорош. Многозадачная операционная система MS Windows вытеснила старую однозадачную MS-DOS. И каждый пользователь достаточно мощного компьютера смог уподобиться Юлию Цезарю.

Многозадачность принесла еще одну выгоду: теперь разные приложения могли взаимодействовать друг с другом, обмениваясь данными. Первой попыткой реализовать такой обмен данными был DDE (Dynamic Data

Exchange — динамический обмен данными), реализованный еще в Windows 3.0. Данные передавались от приложения к приложению небольшими порциями и весьма неспешно, да и надежность DDE была невысока. Вторая попытка — OLE (Object Linking and Embedding — связывание и внедрение объектов) — была несравнимо удачнее и применяется до сих пор, став частью технологии COM (Component Object Model — компонентная объектная модель). Существует еще несколько способов передачи данных от приложения к приложению, но они не столь распространены.

А потом получили широкое распространение компьютерные сети, и в мир пришел Его Величество Интернет. Появились *серверные приложения*, работающие на серверах, не имеющие пользовательского интерфейса и общающиеся с пользователем по сети. Этим они принципиально отличаются от *клиентских приложений*, работающих на компьютерах пользователей и взаимодействующих с пользователями напрямую. Самые распространенные и наверняка вам знакомые серверные программы — это сервер почты, Web-сервер, FTP-сервер и сервер баз данных.

Пользователь никогда не работает с серверной программой напрямую, а вместо этого использует особые клиентские программы, называемые *клиентом* или клиентской частью соответствующего сервера. Такие программы формируют запрос, основываясь на введенных пользователем данных, и отправляют его серверному приложению. В ответ последнее возвращает результат обработки этих данных, который клиентское приложение выдает пользователю. Так, для Web-сервера клиентом является Web-обозреватель, а для сервера данных клиентом может выступать любая современная настольная СУБД.

В этой главе мы поговорим о том, как осуществить взаимодействие приложения Flash с внешними по отношению к нему программами и выполнить обработку внешних данных. Вы узнаете, как открыть Web-страницу в Web-обозревателе, как посылать команды проигрывателю Flash из сценария на ActionScript. Вы научитесь общаться с серверными приложениями, посылать им запросы и принимать от них результаты. И напоследок вы узнаете, как обрабатывать во Flash-приложениях данные, написанные на новомодном языке описания и форматирования данных XML.

Начнем, как водится, с простого...

Управление внешними приложениями

Здесь мы выясним, как из приложения Flash можно управлять другими программами, установленными на том же компьютере. Также мы узнаем,

как можно управлять проигрывателем Flash, в котором работает это приложение.

Загрузка Web-страницы

Для загрузки и отображения Web-страницы вам следует воспользоваться действием `getURL`. Первым параметром этого действия передается интернет-адрес страницы, которую нужно отобразить, в строковом формате. Вторым параметром может быть передано имя окна Web-обозревателя или фрейма, в котором будет показана страница. Кроме того, вы можете передать вторым параметром одно из следующих предопределенных значений:

- `"_self"` — Web-страница выводится в текущий фрейм текущего окна Web-обозревателя;
- `"_blank"` — страница выводится в новое окно;
- `"_parent"` — выводится в родительский фрейм;
- `"_top"` — выводится в окно, в котором находится набор фреймов.

Сценарии, выводющие Web-страницу, чаще всего привязываются к кнопке. В этом случае обработчик события может быть таким:

```
on(release) {  
    getURL("http://www.macromedia.com", "_blank");  
}
```

Этот код выведет в новом окне Web-обозревателя главную страницу сайта фирмы Macromedia.

Наряду с действием `getURL` существует одноименный метод объекта `movieClip`. Он имеет тот же формат вызова и не предоставляет никаких преимуществ по сравнению с действием.

```
on(release) {  
    someClip.getURL("http://www.macromedia.com", "_blank");  
}
```

Управление проигрывателем Flash

Как вы знаете, все фильмы и приложения Flash отображаются в особой программе, называемой проигрывателем Flash. Этот проигрыватель может быть как отдельной, независимой программой, так и встраиваемым модулем для Web-обозревателя. (Также само приложение Flash может быть сохранено в виде исполняемого файла, содержащего проигрыватель, но и в этом случае оно считается отдельным приложением.) И вы можете управлять им из сценариев ActionScript, конечно, в некоторых пределах.

Для управления проигрывателем Flash используется действие `FSCommand`. Формат его вызова таков:

```
FSCommans("<Команда>", "<Аргумент>");
```

В табл. 24.1 приведены все доступные команды, которые можно передать действию `FSCommand`, и их аргументы.

Таблица 24.1. Команды действия `FSCommand` и их аргументы

Команда	Аргументы	Описание
<code>allowascale</code>	<code>true</code> или <code>false</code>	Разрешает масштабирование изображения, чтобы оно занимало все окно проигрывателя (<code>true</code>), или отменяет его (<code>false</code>)
<code>exec</code>	Путь к файлу приложения	Запускает внешнюю по отношению к проигрывателю программу
<code>fullscreen</code>	<code>true</code> или <code>false</code>	Задаёт полноэкранный режим (<code>true</code>) или отменяет его (<code>false</code>)
<code>quit</code>	—	Останавливает проигрывание фильма и закрывает проигрыватель
<code>showmenu</code>	<code>true</code> или <code>false</code>	Разрешает показ контекстного меню проигрывателя (<code>true</code>) или отменяет его (<code>false</code>). В последнем случае показывается только пункт About Flash Player , выводящий сведения о программе проигрывателя
<code>trapallkeys</code>	<code>true</code> или <code>false</code>	Если <code>true</code> , то проигрыватель Flash будет передавать все нажатия клавиш приложению Flash вместо того, чтобы обрабатывать их самому. Если <code>false</code> , то проигрыватель Flash будет обрабатывать некоторые комбинации клавиш сам, а остальные — передавать приложению Flash

Приведем несколько примеров использования действия `FSCommand`.

```
FSCommand("fullscreen", "true");
```

Это выражение заставляет окно проигрывателя Flash развернуться на весь экран. Лучшее место для помещения этого выражения — сценарий, привязанный к первому кадру одного из слоев фильма.

```
FSCommand("exec", "notepad.exe readme.txt");
```

Как насчет того, чтобы почитать файл `readme.txt`, поставляемый в составе Windows? Этот файл откроется в текстовом редакторе Блокнот, также поставляемом в составе Windows. Как говорится, ученье — свет, а неученых — тьма

```
FSCommand("quit", "");
```

Скажи "флэшке" "до свиданья"...

Взаимодействие со сценариями JavaScript

Web-страница, в которую внедрен фильм Flash, может содержать сценарии JavaScript. Вы можете вызывать эти сценарии и выполнять с их помощью какие-то действия над страницей, пользуясь уже знакомым вам действием `FSCommand`. Для этого нужно просто следовать несложным соглашениям об именовании сценариев JavaScript, а остальное берет на себя Flash.

Формат имен сценариев JavaScript таков:

```
<Имя фильма>_DoFSCommand("<Команда>", "<Параметр>");
```

Имя фильма указывается в качестве значения атрибутов `NAME` и `ID` тегов `<OBJECT>` и `<EMBED>`. При экспорте Flash автоматически подставляет в них имя файла фильма без расширения. Следовательно, если фильм назывался `Face.swf`, то в атрибуты `NAME` и `ID` подставляется значение `"Face"`, и имя фильма также будет `"Face"`.

Что касается Команды и Параметра, то они передаются в качестве аргументов действия `FSCommand`. Разумеется, Команда не должна совпадать ни с одной из перечисленных в табл. 24.1 команд, иначе она будет обработана самим проигрывателем Flash, а не передана в сценарий JavaScript.

Пример сценария JavaScript, внедренного в Web-страницу специально для взаимодействия с проигрывателем Flash:

```
<SCRIPT>
function Face_DoFSCommand(command, args) {
    if (command == "messagebox") {
        alert(args);
    }
}
</SCRIPT>
```

Если теперь в сценарии ActionScript, внедренном в фильм Flash, встретится выражение:

```
FSCommand("messagebox", "Привет, мир!");
```

Web-обозреватель выведет окно-предупреждение с текстом "Привет, мир!". И это притом, что сам Flash не поддерживает вывод подобных окон-предупреждений.

Если вы хотите "общаться" с Web-страницей посредством действия `FSCommand`, то при публикации вашего фильма или приложения используйте шаблон "Flash with FSCommand". (Подробнее о публикации и шаблонах см. главу 19.) Этот шаблон содержит необходимый сценарий JavaScript, перехватывающий вызовы действия `FSCommand` в сценариях ActionScript в приложении Flash. Вам остается только добавить свой JavaScript-код. Внутри

этого кода вы можете использовать переменную, чье имя совпадает с именем фильма, т. е. со значением атрибутов `NAME` и `ID`.

Чтобы проверить работу действия `FSCommand` и сценария JavaScript, перехватывающих его вызовы, вам нужно выбрать пункт **HTML** подменю **Publish Preview** меню **File** или нажать комбинацию клавиш `<Ctrl>+<F12>`. При этом флажок **HTML** на вкладке **Formats** окна **Publish Settings** должен быть включен.

Но вызовом внешних сценариев JavaScript возможности Flash не ограничиваются. Сценарии JavaScript, внедренные в Web-страницу, могут управлять проигрыванием фильмов Flash, используя так называемые *методы проигрывателя* Flash. Все эти методы перечислены в табл. 24.2.

Таблица 24.2. Методы проигрывателя Flash

Метод	Аргументы	Описание
<code>GotoFrame</code>	Номер кадра	Перемещает указатель на кадр с заданным номером. Если данный кадр еще не загружен, то перемещает указатель на последний загруженный кадр
<code>IsPlaying</code>	—	Возвращает <code>true</code> , если фильм Flash в настоящее время проигрывается, и <code>false</code> в противном случае
<code>Pan</code>	<code>X</code> , <code>Y</code> , <code>flag</code>	Сдвигает фильм, увеличенный методом <code>SetZoomRect</code> , на <code>X</code> по горизонтали и на <code>Y</code> по вертикали. Если <code>flag</code> равен 0, то <code>X</code> и <code>Y</code> измеряются в пикселах, если 1, то в процентах от размера окна проигрывателя Flash
<code>PercentLoaded</code>	—	Возвращает процент загрузки фильма от 0 до 100
<code>Play</code>	—	Запускает фильм Flash на проигрывание
<code>Rewind</code>	—	"Перематывает" фильм в начало, до первого кадра
<code>SetZoomRect</code>	<code>X1</code> , <code>Y1</code> , <code>X2</code> , <code>Y2</code>	Показывает в окне проигрывателя увеличенный фрагмент фильма так, чтобы он занял все его окно. Пользователь может перетаскивать фильм в окне проигрывателя, чтобы рассмотреть его целиком. <code>X1</code> и <code>Y1</code> — координаты левого верхнего угла, а <code>X2</code> и <code>Y2</code> — координаты правого нижнего угла этого фрагмента. Координаты задаются в твипах; 20 твипов составляют один пиксел
<code>StopPlay</code>	—	Останавливает проигрывание фильма Flash
<code>TotalFrames</code>	—	Возвращает полное количество кадров фильма

Таблица 24.2 (окончание)

Метод	Аргументы	Описание
Zoom	Величина масштаба	Масштабирует фильм в окне проигрывателя. Величина масштаба вычисляется по формуле: $100 / \langle \text{Масштаб в процентах} \rangle$

Здесь нужно дать некоторые пояснения по методу `Zoom`. Это самый "хитроумный" метод из всех, перечисленных в табл. 24.2.

Начать нужно с того, что в качестве параметра подставляется не процент масштабирования, а величина, вычисляемая по формуле: $100 / \langle \text{Масштаб в процентах} \rangle$. В том случае, если вы хотите увеличить изображение в два раза или, говоря другими словами, масштабировать его на 200%, вам нужно вызывать метод

```
myMovie.Zoom(50);
```

Если вызвать метод `Zoom` несколько раз, то все масштабирования, примененные к изображению, складываются. Это значит, что если вызвать следующие два метода:

```
myMovie.Zoom(50);
```

```
myMovie.Zoom(50);
```

то изображение увеличится не в два, а в четыре раза. А если вызвать методы

```
myMovie.Zoom(50);
```

```
myMovie.Zoom(200);
```

то изображение сначала увеличится в два раза, потом уменьшится опять же вдвое и в результате примет изначальный размер.

Привести изображение к изначальному размеру можно также, вызвав метод

```
myMovie.Zoom(0);
```

И еще. Изначально, когда изображение имеет свой исходный размер, вы можете только увеличивать его. Уменьшить размер фильма почему-то не удастся, в чем признаются сами специалисты фирмы Macromedia. Почему существует такое ограничение, и что оно дает, непонятно.

Теперь рассмотрим пример использования методов проигрывателя Flash. Очень часто они используются в сценариях `DoFSCommand`, обрабатывающих вызовы действий `FSCommand`. Мы приведем пример как раз такого сценария.

```
<SCRIPT>
```

```
function myMovie_DoFSCommand(command, args) {  
    var isIE = navigator.appName.indexOf("Microsoft") != -1;  
    var mc = isIE ? myMovie : document.myMovie;  
    if (command == "showPercentLoaded") {
```

```
        alert("Загружено " + myMovie.PercentLoaded() + "% фильма.");  
    }  
}  
</SCRIPT>
```

Использование внешних данных

Мы выяснили, каким образом приложение Flash может управлять внешними программами (а также как внешние программы могут управлять этим приложением). Теперь пришла пора выйти за пределы клиентского компьютера и обратиться к серверным программам.

Как вы уже знаете, серверные программы работают на удаленных компьютерах под управлением программы Web-сервера, с пользователями напрямую не взаимодействуют, а "общаются" с ними только по сети. По сети они принимают от пользователей запросы и так же высылают им ответы: тексты писем, счета, данные из баз и пр. Стало быть, клиентские приложения, "общающиеся" с серверными, должны уметь преобразовывать запросы пользователей в поддерживаемый серверными программами формат. Ну и, конечно, они должны выполнять обратное преобразование, иначе пользователь ничего не сможет разобрать в возвращенном результате.

Flash, как вы уже знаете, делает за вас очень много работы. Он сам кодирует данные, чтобы их можно было нормально передать по протоколу *HTTP* (HyperText Transfer Protocol — протокол передачи гипертекста), используемому при передаче Web-страниц от сервера к клиенту. (*Протоколом* называется набор правил обмена информацией по сети, которых должны придерживаться приложения.) Вам остается только правильно сформировать нужный запрос и получить и обработать ответ.

Посылка запроса серверному приложению

Какие данные могут понадобиться серверному приложению? Самые разные. Почтовому серверу обязательно понадобится имя почтового ящика пользователя и его пароль, адрес назначения и текст отправляемого письма. Серверу баз данных, чтобы вернуть результат, нужен текст запроса на языке SQL, ну и, конечно, имя (так называемый "логин" от английского login) и пароль пользователя. Все эти данные клиент должен собрать, закодировать особым способом и послать серверу.

Как собрать эти данные, вы уже знаете. Методы создания пользовательского интерфейса приложения Flash описывались в *главе 23*. Поэтому написание клиентского приложения не должно вызвать у вас затруднений.

Но как эти данные закодировать и отправить?

Прежде всего, их нужно поместить в переменные. Причем переменные эти должны быть уровня клипа (не локальные и не глобальные). Это ограничение, накладываемое самим Flash, — он не может никуда отправить данные, которые находятся в свойствах объекта.

И еще одно ограничение Flash: отправляются сразу все данные, находящиеся во всех объявленных в клипе переменных. Так что старайтесь не создавать лишних переменных уровня клипа.

Для посылки данных вам следует использовать уже знакомые вам методы и действия:

- ❑ `getURL` — загрузка Web-страницы;
- ❑ `loadMovie` и `loadMovieNum` — загрузка внешнего клипа;
- ❑ `loadSound` — загрузка внешнего звука.

Для этого все перечисленные выше действия (и методы) поддерживают особый параметр. Этот параметр передается самым последним и задает *метод передачи данных*. Всего таких методов два, и каждый из них имеет свои преимущества и недостатки.

Первый метод носит название *GET*. При его использовании данные передаются как часть интернет-адреса в HTTP-запросе, посылаемом Web-обозревателем, и вообще любым Web-клиентом, Web-серверу. Чтобы использовать этот метод, вам нужно передать последнему параметру описанных выше действий значение *GET*.

Рассмотрим пример передачи данных методом *GET*. Пусть у нас имеется набор переменных, описывающих имя и пароль пользователя:

```
name1 = Ivan  
name2 = Ivanov  
password = vanyusha
```

Тогда Flash при использовании метода *GET* передаст Web-серверу следующий адрес (сами данные выделены полужирным шрифтом):

```
http://www.site.ru/bin/program.exe?name1=Ivan&&name2=Ivanov&  
&password=vanyusha
```

Как видите, пересылаемые по методу *GET* данные объединяются в пары "переменная" = "значение", помещаются в самый конец интернет-адреса и отделяются от него вопросительным знаком. При этом одна пара от другой отделяется знаком "коммерческое и" ("&"). Как видите, все очень просто и наглядно.

Такая простота и наглядность представления данных — основное преимущество метода *GET*. Как говорится, все на виду. Также значительно упрощается отладка Web-страниц: поскольку передаваемый Web-серверу адрес всегда отображается в строке адреса Web-обозревателя, вы всегда сможете уви-

деть, что именно было передано. (Однако, как вы понимаете, конфиденциальные данные таким методом не передашь — их увидят все, кто стоит за вашей спиной. Так что плакал наш пароль...)

Метод GET обладает другим огромным недостатком: с его помощью невозможно передавать большие объемы данных. Это происходит из-за ограничения, накладываемого существующими стандартами на длину интернет-адреса: не более 256 символов. Вычтите отсюда длину собственно адреса серверной программы — и вы получите максимально допустимый размер ваших данных.

Метод GET стоит использовать, если пересылаемые серверной программе данные заведомо невелики и не являются секретными. В частности, он применяется для пересылки ключевых слов поисковым машинам. Для пересылки больших либо конфиденциальных данных лучше подходит другой метод передачи, называемый POST.

Метод POST передает данные серверной программе также в HTTP-запросе, также парами "переменная" = "значение", но уже не частью интернет-адреса, а в виде так называемых дополнительных данных. Поскольку размер дополнительных данных не ограничен (по крайней мере, очень велик), вы можете передавать все, что угодно, в каких угодно количествах. Чтобы использовать этот метод, вам нужно передать последнему параметру описанных выше действий значение POST.

Достоинства метода POST — отсутствие ограничения на объем передаваемых данных и их "невидимость". Недостатки — сложность расшифровки данных и трудность отладки серверных программ, принимающих такие данные. Методом POST передаются действительно секретные, а не игрушечные, пароли, анкетные данные, адреса покупателей в электронных магазинах, литературные произведения на сайты <http://www.stihi.ru> и <http://www.proza.ru> и т. п.

Теоретическая часть закончилась, пора переходить к практике. Приведем несколько примеров отправки данных серверной программе обоими рассмотренными методами.

```
getURL("http://www.site.ru/bin/program.pl", "_self", GET);
```

Это выражение отправит содержимое всех переменных серверному приложению методом GET. В результате серверное приложение вернет сформированную им Web-страницу, которая будет отображена в том же окне Web-обозревателя, в котором был загружен данный фильм Flash.

```
_root.loadMovie("http://www.site.ru/bin/getswf.exe", POST);
```

А это выражение использует для отправки данных метод POST. Серверное приложение вернет файл Shockwave/Flash, содержащий клип, который заменит в окне проигрывателя Flash основной фильм. Этот файл может быть как взят с диска серверного компьютера, так и сформирован самой серверной программой на основании отправленных данных.

Кроме того, ничто не мешает вам отправить данные методом GET в самой строке интернет-адреса:

```
getURL("http://www.site.ru/bin/program.pl?name1=Ivan&&name2=Ivanov&password=vanyusha", "_self");
```

Обратите внимание, что последний параметр в этом случае не указан. Это говорит Flash, что никакие данные отправлять не нужно.

Существует очень много Web-сайтов, все гиперссылки которых представляют собой вызов серверной программы. При этом программе передается некое значение, на основании которого она решает, какую Web-страницу нужно отправить клиенту. Причем Web-страницы могут храниться на дисках в виде файлов или формироваться самой программой на основании каких-либо данных. Пример такой гиперссылки:

```
getURL("http://site.ru/bin/getpage.pl?pagenum=4", "_self");
```

Можно и по-другому:

```
pagenum = 4;  
getURL("http://site.ru/bin/getpage.pl", "_self", GET);
```

Получение данных от серверного приложения

А теперь мы рассмотрим получение данных от серверного приложения.

Самый простой способ получить данные от серверного приложения — это использовать действие `loadVariables` или одноименный метод объекта `movieClip`. Это действие (метод) принимает данные в текстовом виде, записанные в виде пар "переменная" = "значение", разделенных знаком "коммерческое и" (то есть, так, как кодируются данные, передаваемые методом GET). В принципе с его помощью вы можете заставить ваше приложение Flash загружать данные даже из текстового файла, находящегося на сервере или даже локальном диске.

Формат вызова действия `loadVariables` таков:

```
loadVariables("Интернет-адрес приложения", "Путь клипа"[, GET|POST]);
```

Первым параметром передается интернет-адрес серверного приложения, которое должно послать данные, или текстового файла, содержащего их. Вторым параметром передается путь клипа, который будет обрабатывать эти данные. Третий параметр может задавать метод передачи переменных, если, конечно, серверная программа их требует.

Примеры использования этого действия:

```
loadVariables("http://site.ru/bin/program.pl", _root.dataHandler, POST);  
loadVariables("data.txt", _root.dataHandler);
```

Оба вышеперечисленных выражения загружают в клип `dataHandler` данные. Но только первое выражение получает их от серверного приложения, пере-

дав ему какие-то параметры методом POST, а второе получает их из текстового файла.

Метод `loadVariables` вызывается почти так же:

```
<Клип>.loadVariables("Интернет-адрес приложения"[, GET|POST]);
```

Например,

```
_root.dataHandler.loadVariables("http://site.ru/bin/program.pl", POST);
```

Существует, кстати, действие `loadVariablesNum`. Оно ссылается на клип не по его пути, а по уровню.

```
loadVariables("Интернет-адрес приложения", "Уровень клипа"[, GET|POST]);
```

Вроде бы, все просто. Но не совсем. Рассмотрим простенький сценарий (значение переменной `frameNumber` формируется серверной программой):

```
_root.loadVariables("http://site.ru/bin/program.exe");
```

```
_root.gotoAndStop(frameNumber);
```

Этот код работать не будет. Сейчас мы расскажем, почему.

Когда вы даете Flash команду забрать данные у серверного приложения или загрузить текстовый файл с данными, вы не можете быть уверены, что эти данные придут сию же секунду. Более того, вообще нельзя быть уверенным, что они будут получены в течение какого-то заданного промежутка времени. Они могут прийти через долю секунды, а могут задержаться на час. И вам придется ждать, пока они придут.

Но как поймать тот момент, когда Flash примет данные? Для этого вам будет нужно обработать событие `data` объекта `movieClip`. Это событие наступает, когда все данные до самой последней переменной будут приняты клипом.

Перепишем приведенный выше сценарий так, чтобы он заработал.

```
_root.onData = function() {  
    _root.gotoAndStop(frameNumber);  
}
```

```
_root.loadVariables("http://site.ru/bin/program.exe");
```

Этот сценарий может быть привязан к кадру фильма. Вы можете также привязать сценарий, загружающий данные, к кнопке, но это не имеет принципиального значения. Главное — не забывайте обработать событие `data`.

Использование объекта *LoadVars*

Существует другой, более сложный способ обмена данными с серверным приложением. Это использование экземпляров объекта `LoadVars`. Объект `LoadVars` предоставляет по сравнению с действиями `loadVariable` и `loadVariableNum` несколько больший контроль над процессом обмена данными, хотя выполняет те же самые задачи.

Сначала вы должны создать экземпляр объекта `LoadVars`:

```
myLoadVars = new LoadVars();
```

Затем вы просто создаете в нем новые свойства, чьи имена совпадают с именами переменных, и присваиваете этим свойствам нужные значения:

```
myLoadVars.name1 = "Ivan";
```

```
myLoadVars.name2 = "Ivanov";
```

```
myLoadVars.password = "vanyusha";
```

Чтобы отправить данные серверному приложению, используйте метод `send`. Формат его таков:

```
<Экземпляр объекта LoadVars>.send("<Интернет-адрес>"[, "<Цель>",  
    ⚡GET|POST]);
```

Первым параметром передается интернет-адрес серверного приложения, которое получит эти данные. Вторым параметром может быть передана цель, куда будет выведен результат обработки этих данных (как правило, Web-страница, сгенерированная серверной программой). Третьим параметром может быть передан метод отправки данных; если он не указан, используется метод `POST`.

```
myLoadVars.send("http://site.ru/bin/program.exe", "_blank", GET);
```

Для приема данных от серверной программы используется метод `load`. Единственным параметром этого метода является интернет-адрес серверного приложения или файла данных.

```
myLoadVars.load("http://site.ru/bin/program.exe");
```

Метод `sendAndLoad` одновременно отправляет данные и получает результат.

```
<Экземпляр объекта LoadVars>.sendAndLoad("<Интернет-адрес>",  
    ⚡<Экземпляр объекта LoadVars, принимающий данные>[, GET|POST]);
```

Первым параметром опять же передается интернет-адрес серверного приложения, которое получит эти данные. Вторым параметром передается ссылка на экземпляр объекта `LoadVars`, который получит результат. Третьим параметром может быть передан метод отправки данных; если он не указан, используется метод `POST`.

```
myLoadVars.sendAndLoad("http://site.ru/bin/program.exe", myLoadVars);
```

Для того чтобы узнать, получены ли данные, вам будет нужно обработать событие `load` объекта `LoadVars`. (Обратите внимание: не `data`, а `load`!) Функция-обработчик этого события должна принимать один логический параметр, обозначающий, что данные успешно получены (значение `true`) или почему-то не дошли (`false`).

```
myLoadVars.onLoad = function(flag) {  
    _root.gotoAndStop(frameNumber);  
}
```

Свойство `loaded` возвращает `true`, если данные успешно получены, или `false` в противном случае. Если операция приема данных не была запущена, возвращается значение `undefined`.

Объект `LoadVars` также предоставляет вам еще два метода, которые вы можете использовать для отображения процесса загрузки данных. Метод `getBytesLoaded` возвращает количество загруженных байт данных. А метод `getBytesTotal` возвращает общий объем загружаемых данных в байтах. Оба эти метода возвращают `undefined`, если операция загрузки данных не была запущена или еще реально не началась. Метод `getBytesTotal` также возвращает `undefined`, если Web-сервер, под управлением которого работает серверная программа, не сообщил размер передаваемых данных.

Использование данных XML

Интернет-общественность нашла себе новую игрушку. Это широко разрекламированный язык описания данных XML (`eXtensible Markup Language` — расширяемый язык разметки), который служит для структурирования данных. Считается, что он должен покончить с неразберихой, связанной с существованием множества несовместимых форматов хранения данных, привести всю информацию, накопленную трудолюбивым человечеством, в строгий порядок. Сбудутся ли эти прогнозы, оправдает ли XML надежды страждущих? Кто знает...

Сейчас мир охватила мода на XML. Очень и очень многие программы спешно обзаводятся поддержкой этого языка, невзирая, нужна она там реально или нет. Не стал исключением и Flash. Уже предыдущая, пятая версия поддерживала данные, отформатированные с использованием этого языка. А Flash MX эту поддержку развил и углубил.

Давайте же выясним, что предлагает нам Flash в плане поддержки этого языка.

Вводный курс языка XML

Собственно, в *главе 23* мы уже немного говорили об XML. Мы выяснили, что он пригоден для форматирования данных, записываемых в текстовом формате. Также мы выяснили, что он похож на язык HTML, с помощью которого создаются Web-страницы, тем, что также использует теги. Только в HTML теги служат для создания различных элементов Web-страницы, а в XML — для разбиения данных на логические части.

Рассмотрим фрагмент кода HTML:

```
<H1>Это заголовок, <B>выделенный полужирным шрифтом</B>.</H1>  
<H2>Это подзаголовок, <I>выделенный курсивом</I>.</H2>
```

```
<CITE>Это цитата.</CITE>
```

```
<P>Это текстовый абзац.</P>
```

Здесь теги используются для создания элементов Web-страницы: заголовка, подзаголовка, цитаты и обычного абзаца текста. Также они задают дополнительное форматирование текста: выделения полужирным шрифтом, курсивом и пр.

Теперь рассмотрим фрагмент кода XML:

```
<DEVELOPER DID="MM" NAME=" Macromedia">  
  <PRODUCT PID="DW">Dreamweaver</PRODUCT>  
  <PRODUCT PID="FL">Flash</PRODUCT>  
  <PRODUCT PID="FW">Fireworks</PRODUCT>  
</DEVELOPER>
```

Здесь представлен фрагмент отформатированных данных. Они могут быть использованы как угодно, специальное программное обеспечение может превратить их в текстовый файл, Web-страницу, файл Shockwave/Flash, документ PDF или просто загрузить в память и использовать для своих нужд.

Каждый тег XML называется *узлом* (по-английски *node* — узел). Узел имеет первый *тип*, если он содержит другие узлы XML, и третий, если он содержит текст. Так, тег `<DEVELOPER>` в нашем примере первого типа, а тег `<PRODUCT>` — третьего.

Вложенные узлы называются *дочерними*, а узел, в который они вложены, — *родительским*. Причем один и тот же узел может быть родительским для одних узлов и дочерним для других.

Теги XML могут содержать атрибуты. Так, тег `<DEVELOPER>` в нашем примере содержит атрибуты `DID` (уникальный код разработчика) и `NAME` (имя разработчика).

Все теги XML должны быть парными. Если же тег никак не может быть парным (в него ничего не вложено), то в его конце ставится знак косой черты:

```
<PRODUCT PID="FL" PRNAME="Flash"/>
```

Язык XML чувствителен к регистру символов. Это значит, что теги `<PRODUCT>` и `<product>` с его точки зрения разные.

Совокупность всех узлов документа XML образует структуру, называемую XML *DOM* (Document Object Model — объектная модель документа).

Сохраняются данные, отформатированные с помощью языка XML, в текстовых файлах с расширением `xml`. Создавать их можно в обычном текстовом редакторе или специальных XML-редакторах (если, конечно, сможете их отыскать).

Конечно, это краткое описание не охватывает всех возможностей весьма мощного языка XML, который, возможно, совершит революцию в форматах хране-

ния данных. Чтобы узнать о нем все, найдите посвященные ему книги, которых в последнее время издано довольно много, в том числе, и на русском языке.

Использование объекта XML

Для обмена с серверным приложением данными в формате XML используется экземпляр объекта, который так и называется — XML. Он содержит набор методов, с помощью которых вы можете "собрать" из имеющихся данных документ XML или, наоборот, "разобрать" его по частям и извлечь хранящиеся в нем данные. Этот же объект может служить для обработки информации, хранящейся в файлах XML.

Прежде всего, вы должны создать экземпляр объекта XML:

```
myXML = new XML();
```

При создании экземпляра объекта вы можете передать конструктору в качестве параметра уже сформированный XML-код.

```
myXML = new XML("<PRODUCT PID='FL' PRNAME='Flash'/>");
```

После этого вы можете добавлять во вновь созданный объект новые узлы.

Для добавления нового узла вы должны использовать метод `createElement`. Этому методу передается один параметр — имя узла. Метод `createElement` возвращает ссылку на экземпляр объекта XML, представляющий вновь созданный узел.

```
myNode = myXML.createElement("PRODUCT");
```

Для создания текстового узла вам нужно использовать метод `createTextNode` и передать ему в качестве параметра текст, который станет содержимым узла.

```
myTextNode = myXML.createTextNode("Лучшая программа векторной графики!");
```

Теперь следует добавить атрибуты узла. Для этого вам необходимо использовать внутренний объект — массив `attributes`. Просто присвойте нужному атрибуту требуемое значение, как вы делали это с массивами ранее.

```
myNode.attributes.PID = "FL";
```

```
myNode.attributes.PRNAME = "Flash";
```

Остается добавить созданные отдельные узлы в основной XML-документ. Это делается с помощью метода `appendChild`. Вызовите его у того экземпляра объекта XML, который должен стать родительским узлом. А в качестве параметра передайте этому методу ссылку на экземпляр объекта XML, представляющий дочерний узел.

```
myNode.appendChild(myTextNode);
```

Это выражение делает созданный нами текстовый узел дочерним по отношению к узлу `<PRODUCT>`.

```
myXML.appendChild(myNode);
```

А это выражение добавляет полученный нами ранее узел в пустой пока еще XML-документ. Содержимое этого документа станет таким:

```
<PRODUCT PID="FL" PRNAME="Flash">Лучшая программа векторной  
❧ графики!</PRODUCT>
```

Пользуясь методами `createElement`, `createTextNode` и `appendChild`, вы сможете создавать XML-документы любой сложности. В качестве домашнего задания попробуйте написать сценарий, создающий документ, чей текст был приведен выше, при рассмотрении языка XML.

Чтобы отправить данные серверному приложению, используйте метод `send`. Формат его таков:

```
<Экземпляр объекта XML>.send("<Интернет-адрес>[ , "<Цель>"]);
```

Первым параметром передается интернет-адрес серверного приложения, которое получит эти данные. Вторым параметром может быть передана цель, куда будет выведен результат обработки этих данных (как правило, Web-страница, сгенерированная серверной программой). Данные всегда передаются методом `POST`.

```
myXML.send("http://site.ru/bin/xmlparser.exe");
```

Для приема данных от серверной программы используется метод `load`. Единственным параметром этого метода передается интернет-адрес серверного приложения или файла, в котором хранится документ XML.

```
myXML.load("http://site.ru/bin/xmlparser.exe");
```

Метод `sendAndLoad` одновременно отправляет данные и получает результат.

```
<Экземпляр объекта XML>.sendAndLoad("<Интернет-адрес>",  
❧ <Экземпляр объекта XML, принимающий данные>);
```

Первым параметром опять же передается интернет-адрес серверного приложения, которое получит эти данные. Вторым параметром передается ссылка на экземпляр объекта XML, который получит результат. Для отправки данных опять же используется метод `POST`.

```
myXML.sendAndLoad("http://site.ru/bin/xmlparser.exe", myXMLLoader);
```

Для того чтобы узнать, успешно ли получены данные, вам будет нужно обработать событие `load` объекта XML. Функция-обработчик этого события должна принимать один логический параметр, обозначающий, что данные успешно получены (значение `true`) или почему-то не дошли (`false`).

```
myXML.onLoad = function(flag) { . . .
```

Свойство `loaded` объекта XML возвращает `true`, если данные были успешно получены, или `false` в противном случае. Если операция приема данных не была запущена, возвращается значение `undefined`.

Свойства `firstChild` и `lastChild` возвращают соответственно первый и последний дочерний узлы документа XML или его узла. Эти свойства доступны только для чтения.

```
dev = myXML.firstChild;  
product1 = dev.lastChild;
```

Для доступа к дочерним узлам вы можете использовать внутренний объект — массив `childNodes`.

```
nodes = new Array();  
dev = myXML.firstChild;  
for(i = 0; i < dev.childNodes.length - 1; i++) {  
    nodes[i] = new Object();  
    nodes[i].pID = dev.childNodes[i].attributes.PID;  
    nodes[i].prName = dev.childNodes[i].attributes.PRNAME;  
    nodes[i].prDescription = dev.childNodes[i].firstChild.nodeValue;  
}
```

В последнем выражении этого сценария мы применили свойство `firstChild`, чтобы добраться до текстового узла, являющегося дочерним по отношению к узлу `<PRODUCT>`. Можно также использовать массив `childNodes`, обратившись к первому (точнее, нулевому) его элементу. Сам текст, содержащийся в этом узле, мы получили с помощью свойства `nodeValue`.

Метод `hasChildNodes` возвращает `true`, если текущий узел имеет дочерние узлы, или `false` в противном случае. Этот метод не принимает параметров.

Свойство `nextSibling` возвращает ссылку на узел той же степени вложенности, что и текущий. Например, для нашего документа XML, чей текст мы привели в качестве примера при описании этого языка, узлами одинаковой степени вложенности являются узлы `<PRODUCT>`. А значит, для первого узла `<PRODUCT>` свойство `nextSibling` вернет ссылку на второй узел `<PRODUCT>`, для второго — на третий, а для третьего — значение `null`, т. к. больше узлов такой же степени вложенности нет.

Аналогично свойство `previousSibling` возвращает ссылку на предыдущий узел той же степени вложенности.

Свойства `nodeName` и `nodeType` возвращают соответственно имя и тип узла. Уже известное нам свойство `nodeValue` возвращает содержимое текстового узла, для обычных узлов оно возвращает `null`. Все эти три свойства доступны только для чтения.

И, наконец, не принимающий параметров метод `toString` возвращает текстовое представление документа XML. Этот метод вы можете использовать, например, для проверки правильности формирования документа.

Использование объекта *XMLSocket*

Итак, мы узнали, как можно передать данные серверной программе и получить от нее ответ. Теперь мы можем создавать свои собственные системы анкетирования посетителей своего Web-сайта, почтовые сервера, электронные магазины и пр. Осталось только написать серверное приложение, которое будет обрабатывать данные, но это выходит за рамки настоящей книги.

А теперь давайте выясним, как происходит пересылка данных серверному приложению. Распишем этот процесс по шагам.

1. Приложение Flash, отправляющее данные, устанавливает сетевое соединение с Web-сервером.
2. Приложение Flash отправляет данные серверному приложению через Web-сервер.
3. Web-сервер подтверждает принятие данных.
4. Приложение Flash разрывает сетевое соединение с Web-сервером. Передача данных закончена.

Точно так же серверное приложение (через Web-сервер) передает ответ приложению Flash. Не будем рассматривать обратную передачу. Поговорим о другом.

Способ передачи, рассмотренный нами выше, годится для отправки данных серверной программе, осуществляющей обработку анкет или покупательской "корзины". Сетевое соединение устанавливается на короткий промежуток времени, необходимый только для передачи данных в одну сторону, и после этого разрывается.

Но иногда между клиентским и серверным приложениями требуется организовать продолжительный обмен информацией. Такое может понадобиться, например, при организации системы сетевой "болтовни" (или "чата", от английского chat), когда текстовые сообщения посылаются постоянно от сервера клиенту и от клиента серверу. Если для передачи каждого сообщения создавать, а потом разрывать сетевое соединение, сильно пострадает быстродействие системы, что вряд ли понравится любителям почесать языки. Поэтому в таких случаях устанавливается постоянное сетевое соединение, которое будет открыто столько времени, сколько нужно.

Flash для поддержки постоянных сетевых соединений предоставляет объект *XMLSocket*. С помощью этого объекта вы можете открывать постоянное соединение, закрывать его, когда оно станет ненужным, передавать и принимать через него данные.

Данные по постоянному соединению, как вы уже поняли, передаются в виде документа XML. Для создания документа XML используется уже знакомый вам объект *XML*, экземпляр которого вы должны создать и "заполнить" необходимыми данными.

И еще. Для прямого сетевого обмена нужны особые серверные приложения. Эти приложения работают сами по себе, а не под управлением Web-сервера. Вопросы написания таких приложений рассмотрены в соответствующей литературе.

Для организации постоянного сетевого соединения между клиентским приложением Flash и серверной программой нужно, прежде всего, создать экземпляр объекта `XMLSocket`.

```
myXMLSocket = new XMLSocket();
```

Чтобы открыть соединение, используйте метод `connect`. Формат этого метода таков:

```
<Экземпляр объекта XMLSocket>.connect(<Интернет-адрес>, <Порт>);
```

Первым параметром передается интернет-адрес — внимание! — сервера, на котором работает серверное приложение. Вы также можете передать в качестве параметра `null` для соединения с сервером, откуда был загружен файл с приложением Flash. Вторым параметром передается номер *порта* — своеобразного виртуального канала, по которому передаются данные. Всего доступно 65535 портов, но Flash может использовать для создания постоянных соединений только порты с номерами 1024 и выше.

Метод `connect` возвращает `true`, если создание постоянного соединения успешно начато. В противном случае возвращается `false`.

```
myXMLSocket.connect("socket.site.ru", 2048);
```

Сразу после установления постоянного соединения наступает событие `connect`. Функция-обработчик этого события должна принимать один логический параметр. Если этот параметр равен `true`, то соединение успешно установлено, и можно начинать обмен. Если же он равен `false`, удача отвернулась от вас.

```
myXMLSocket.onConnect = function(success) {  
    if (success) {  
        gotoAndStop(chat);  
    } else {  
        gotoAndStop(error);  
    }  
}
```

Для отправки данных используйте метод `send`. В качестве единственного параметра этого метода передается ссылка на экземпляр объекта `XML`, содержащего нужные данные.

```
myXMLSocket.send(myXML);
```

Сразу после приема данных от серверной программы наступает событие `XML`. Функция-обработчик этого события должна принимать единственный пара-

метр — ссылку на экземпляр объекта XML, содержащего информацию, принятую от серверной программы.

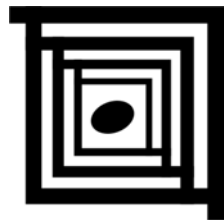
```
myXMLSocket.onXML = function(request) {  
    txtOutput += request.firstChild.firstChild.nodeValue;  
}
```

Для закрытия постоянного соединения вам нужно вызвать метод `close`. Этот метод не принимает параметров.

```
myXMLSocket.close();
```

Постоянное соединение может быть также разорвано серверной программой. Другие причины разрыва постоянного соединения: авария сети, ошибки в программном обеспечении и др. Для того чтобы корректно обработать разрыв соединения, объект `XMLSocket` предоставляет событие `close`.

```
myXMLSocket.onClose = function() {  
    gotoAndStop(connectionClosed);  
}
```



Глава 25

Средства отладки сценариев ActionScript

В мире нет ничего совершенного. Даже компьютеры — и те несовершенны, постоянно "зависают", ломаются, "глючат" по любому поводу, а то и без повода. И это понятно: ведь компьютеры — творение людей, а люди да что и говорить!.. люди есть люди...

Программы содержат ошибки. Не все, конечно, а те из них, что состоят более чем из двух строк кода. (Функционального кода, который что-то реально делает.) Чем больше и сложнее программа, тем больше (теоретически, по крайней мере) в ней ошибок. Вы и сами, конечно, это знаете: уже всем надоели истории об ошибках в операционных системах Microsoft Windows, да и другие большие программные пакеты не лучше. Производители ПО, разумеется, пытаются с этим бороться различными способами, но пока что толку особо не видно. А проистекает все это безобразие опять же оттого, что программы пишутся людьми. А люди есть люди...

Разумеется, ошибки необходимо исправлять. (Если вы думаете, что их нужно смывать кровью, попытайтесь вспомнить все совершенные вами ошибки и прикиньте, хватит ли у вас крови все их смыть.) Для этого используются мощные программные отладчики, организуются специальные, весьма дорогостоящие мероприятия, выпускаются бесконечные пакеты обновления и т. д. и т. п. И что в результате? Как говорят злые языки, "исправляются старые ошибки и добавляются новые". Не хотелось, конечно, чтобы это было на самом деле, но люди есть люди!

Людям свойственно ошибаться. Хорошо еще, что хоть компьютеры ошибаться не могут в принципе. (Конечно, имеются в виду исправные компьютеры.) Компьютеры лишены свободы воли, они только выполняют программный код, созданный людьми. Для них он всегда правилен.

Но хватит философских отступлений. Давайте поговорим о "вылавливании" ошибок, допущенных вами в сценариях ActionScript, и их исправлении. Одним словом, поговорим об отладке сценариев.

Но прежде — небольшое теоретическое введение. Рассмотрим средства, которые может использовать Flash-программист, чтобы найти ошибки в своих сценариях.

Как выявить ошибки

Ошибки, встречающиеся в сценариях ActionScript и программах вообще, можно разделить на два принципиально разных вида. Это ошибки синтаксические и логические.

Синтаксические ошибки — это ошибки и неточности в написании самого кода. Скажем, если вы написали вместо действия `else` что-то похожее на `elswe`, Flash предупредит вас о синтаксической ошибке. В самом деле, действия `elswe` в языке ActionScript нет, и предупреждение Flash вполне резонно.

Мы рассмотрели простейшую синтаксическую ошибку, которая, что называется, бьет в глаза. Более сложная синтаксическая ошибка — вызов несуществующего метода объекта. Тут сразу трудно понять, что вызывает ошибку. Хорошо, что Flash во многих случаях весьма точно дает знать, что ему не нравится в вашем коде.

Когда вы вводите код в обычном режиме панели **Actions**, Flash сам следит за правильностью написания кода. Если вы сделаете что-то не так, он подсветит некорректный фрагмент кода красным (рис. 25.1). Вам будет нужно исправить его, пользуясь соответствующим элементом управления. (Подробнее использование панели **Actions** было описано в *главе 20*.)

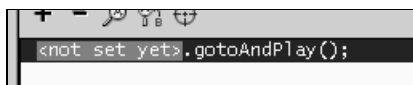


Рис. 25.1. Ошибочный фрагмент кода в панели **Actions** (выделен светло-серым)

Ситуация осложняется, если вы предпочитаете пользоваться профессиональным режимом панели **Actions**. В этом случае вам придется набирать код сценария самим, без помощи Flash. А значит, Flash не может проследить за корректностью этого кода, и риск возникновения ошибок значительно возрастает. Что делать в этом случае?

Выберите пункт **Check Syntax** в дополнительном меню панели **Actions** или нажмите комбинацию клавиш <Ctrl>+<T>. Если в коде нет синтаксических ошибок, Flash выдаст окно-предупреждение с сообщением "This script contains no errors". Если же Flash найдет ошибки, текст сообщения будет таким: "This script contains errors. The errors encountered are listed in Output Window". Сами же найденные ошибки будут перечислены в открывшемся на экране окне **Output** (рис. 25.2), которое мы подробно рассмотрим далее в этой главе.

Кроме того, Flash автоматически проверяет сценарии на наличие ошибок при их выполнении. Описания найденных ошибок выдаются также в окно **Output**.

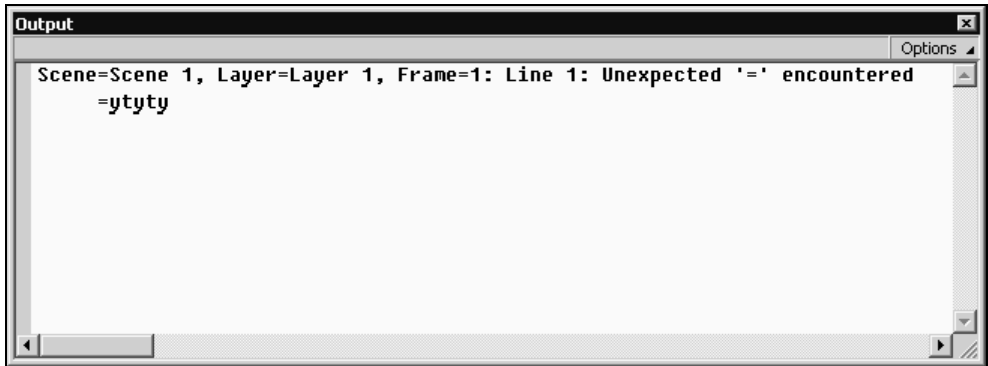


Рис. 25.2. Окно **Output**, содержащее список найденных ошибок

С синтаксическими ошибками иметь дело проще всего. Как правило, программный код, содержащий такие ошибки, вообще не работает, поэтому выявить их просто. Иначе обстоит дело со второй разновидностью ошибок, которую мы сейчас рассмотрим.

Логические ошибки — это ошибки в логике работы программы. В этом случае программный код выглядит абсолютно правильным, но работает неправильно. Такие ошибки выявить и исправить намного сложнее.

Логические ошибки могут быть вызваны многими причинами. Скажем, если вы неправильно написали переменную в сценарии, код останется правильным, но работать будет неверно. Давайте рассмотрим типичный пример такой ошибки — вместо `a` было написано `aa` (ошибка в коде выделена полужирным шрифтом).

```
a = b + c;  
d = aa * 2;
```

В этом случае Flash создаст новую переменную `aa` и использует ее значение для вычисления значения переменной `d`. Конечно, результат не будет правильным.

Еще более сложные случаи — ошибки в реализации самого сценария. Такие ошибки выявить труднее всего. А чтобы их исправить, зачастую приходится переписывать весь сценарий.

Процесс выявления ошибок в сценарии называется *отладкой*. Отладка заключается в том, что приложение Flash, содержащее отлаживаемые сценарии, испытывается самим разработчиком. Как правило, большинство и синтаксических, и логических ошибок "вылезают" на божий свет при первом же выполнении этого приложения. Если какой-то сценарий работает неправильно, это сразу видно.

Да, но если ошибка такая хитрая, что ее нельзя выявить даже после нескольких запусков приложения? В этом случае на помощь программисту

приходит особая программа, называемая *отладчиком*. Такой отладчик встроен прямо в среду Flash и запускается при выборе пункта **Debug Movie** меню **Control** или нажатии клавишной комбинации <Ctrl>+<Shift>+<Enter>.

Какие же возможности предлагает нам встроенный отладчик Flash?

- ❑ Установка так называемых *точек останова*. Это особые метки в коде сценария, на которых его выполнение приостанавливается, после чего программист может, не торопясь, оценить полученные результаты. Далее выполнение сценария может быть продолжено до его конца или до следующей точки останова. Точки останова применяются, чтобы выяснить, что происходит в том или ином месте сценария.
- ❑ Пошаговое выполнение или, как говорят опытные программисты, *трассировка* кода. Программист может выполнять код выражение за выражением и наблюдать за результатами их выполнения. Пошаговое выполнение позволит точно узнать, как выполняется сценарий, что происходит во время его выполнения, и, вообще, правильно ли он работает.
- ❑ Просмотр значений переменных. Может пригодиться практически всегда. Позволяет сразу же выявить ошибки в написании имен переменных.

Еще одно весьма полезное средство выявления ошибок — это уже знакомое вам окно **Output**. Оно служит для выдачи различной служебной информации, которая может очень пригодиться в отладке. В частности, как вы уже знаете, в этом окне выдается описание всех найденных синтаксических ошибок.

Рассмотрим оба этих средства. И начнем с окна **Output**.

Окно Output

Чтобы вывести на экран окно **Output** (см. рис. 25.2), выберите пункт **Output** в меню **Control** или нажмите клавишу <F2>. Однако вам нечасто придется делать это самостоятельно. Обычно Flash сам выводит это окно на экран, если ему нужно что-то вам сообщить.

Какого же рода информация может отображаться в этом окне? Давайте это выясним.

Flash предусматривает особое действие `trace`, которое служит именно для отладки сценариев. Это действие принимает единственный параметр — либо переменную, либо выражение, значение которого будет выведено в окно **Output**. С помощью этого действия вы можете просмотреть значение любой переменной вашего сценария и проверить правильность работы любого выражения.

Так, в сценарии

```
for(x = -20; x < 21; x++) {  
    y = x * x;
```

```
trace(y);  
.  
.  
.  
}
```

вы сможете узнать значение переменной `y`. А с помощью выражения

```
this._alpha = 50;  
trace(this);
```

легко проверить, к тому ли объекту вы обращаетесь.

При выводе нового значения в окно **Output** все старые значения сдвигаются вверх. Чтобы просмотреть их, воспользуйтесь полосой прокрутки.

При экспорте готового фильма все действия `trace` остаются в коде. Конечно, при просмотре в проигрывателе Flash они работать не будут, но если экспортированный фильм открыть в среде Flash, они нормально заработают. Если вы хотите убрать все действия `trace` из кода при экспорте, включите флажок **Omit Trace Actions** на вкладке **Flash** диалогового окна **Publish Settings** (см. рис. 19.1).

Когда запущено проигрывание фильма, вы можете просмотреть в окне **Output** еще кое-какую информацию, а именно, списки всех объектов и переменных, содержащихся в фильме. В режиме рисования эти возможности недоступны.

Список объектов включает все экземпляры клипов, кнопок, полей ввода и динамических текстовых блоков, присутствующих в фильме. Также указываются все имеющиеся в фильме графические примитивы и текстовые блоки. Вы можете воспользоваться этим списком, чтобы определить корректный путь к нужному экземпляру. Выберите пункт **List Objects** в меню **Debug** или нажмите комбинацию клавиш `<Ctrl>+<L>`. Сам список объектов вы можете увидеть на рис. 25.3.

Список переменных включает все переменные, глобальные, уровня клипа и локальные, имеющиеся в данном фильме. В списке также приводятся значения всех этих переменных. Чтобы вызвать его на экран, выберите пункт **List Variables** в меню **Debug** или нажмите комбинацию клавиш `<Ctrl>+<Alt>+<V>`. Сам список переменных показан на рис. 25.4.

Учтите, что и список переменных, и список объектов, выданный Flash в окне **Output**, не изменяются во время проигрывания фильма. Даже если при этом будет создан новый или удален уже существующий объект, список объектов не обновится. Чтобы увидеть все изменения, вам придется снова вызвать соответствующий пункт дополнительного меню.

Окно **Output** предоставляет еще несколько полезных возможностей. Они доступны после выбора пунктов контекстного или дополнительного меню этого окна. Так, с помощью пункта **Copy** или комбинации клавиш `<Ctrl>+<C>` вы можете скопировать выделенный в этом окне текст в буфер обмена Windows. (Текст можно выделить мышью, как в обычном текстовом редакторе.) Пункт **Clear** вызывает очистку окна **Actions**. Выбрав пункт **Print**, вы можете распечатать содержимое этого окна, а, выбрав пункт **Save to File**, — сохранить его в текстовый файл с расширением `log` или `txt`.

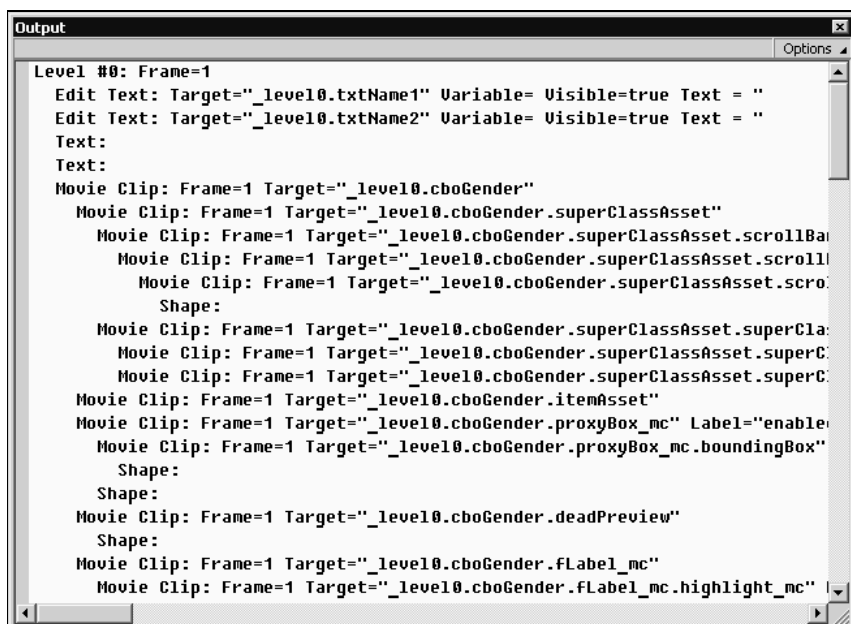


Рис. 25.3. Список объектов в окне Output

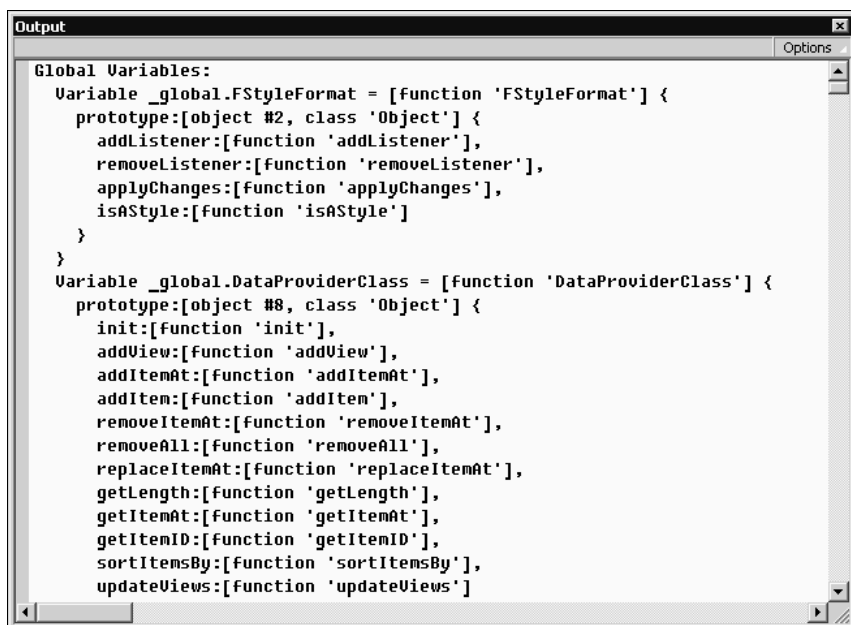


Рис. 25.4. Список переменных в окне Output

Если вы выберете пункт **Find** контекстного или дополнительного меню окна **Output** или нажмете комбинацию клавиш <Ctrl>+<F>, на экране появится диалоговое окно поиска **Find** (см. рис. 20.14). Введите в поле **Find What** строку, которую вам нужно найти, включите флажок **Match Case**, если хотите, чтобы Flash учитывал при поиске регистр символов, и нажмите кнопку **Find Next**. Flash выделит найденную строку. Если же такой строки найдено не будет, Flash выведет предупреждение "Cannot find the string '<Ваша строка или числовое значение>'".

Чтобы продолжить поиск строки далее по тексту сценария, снова нажмите кнопку **Find What** диалогового окна **Find**. Если же это окно уже закрыто вами (для чего достаточно нажать кнопку **Close**), выберите в контекстном или дополнительном меню окна пункт **Find Next** или нажмите клавишу <F3>.

Особняком стоит подменю **Debug Level**, находящееся в дополнительном меню окна **Output**. С помощью его пунктов вы можете выбрать информацию, которая будет отображаться в этом окне. Всего подменю **Debug Level** содержит четыре пункта-переключателя:

- ☐ **None** — не отображается никаких сообщений;
- ☐ **Errors** — отображаются сообщения только о критических ошибках, которые мешают Flash правильно выполнить код;
- ☐ **Warnings** — отображаются сообщения только о некритических ошибках, не мешающих правильному выполнению кода;
- ☐ **Verbose** — отображаются сообщения обо всех ошибках.

Список переменных и объектов, а также вывод действий `trace` отображается только тогда, когда включен пункт-переключатель **Errors**, **Warnings** или **Verbose**. То есть, когда разрешен вывод хоть каких-то сообщений.

Отладчик Flash

А теперь пришла пора поговорить об отладчике, встроенном в среду Flash.

Использование отладчика Flash

Отладчик Flash активизируется только при проигрывании фильма в так называемом режиме отладки. Во время обычного проигрывания и рисования фильма он недоступен.

Чтобы запустить проигрывание фильма в отладочном режиме, выберите пункт **Debug Movie** меню **Control** или нажмите комбинацию клавиш <Ctrl>+<Shift>+<Enter>. После этого фильм будет экспортирован и открыт в отдельном окне Flash, но проигрывание фильма будет приостановлено. Кроме того, на экране появится само окно отладчика (рис. 25.5). Чтобы запустить проигрывание фильма, щелкните кнопку **Continue**, расположенную в верхней части этого окна (рис. 25.6).

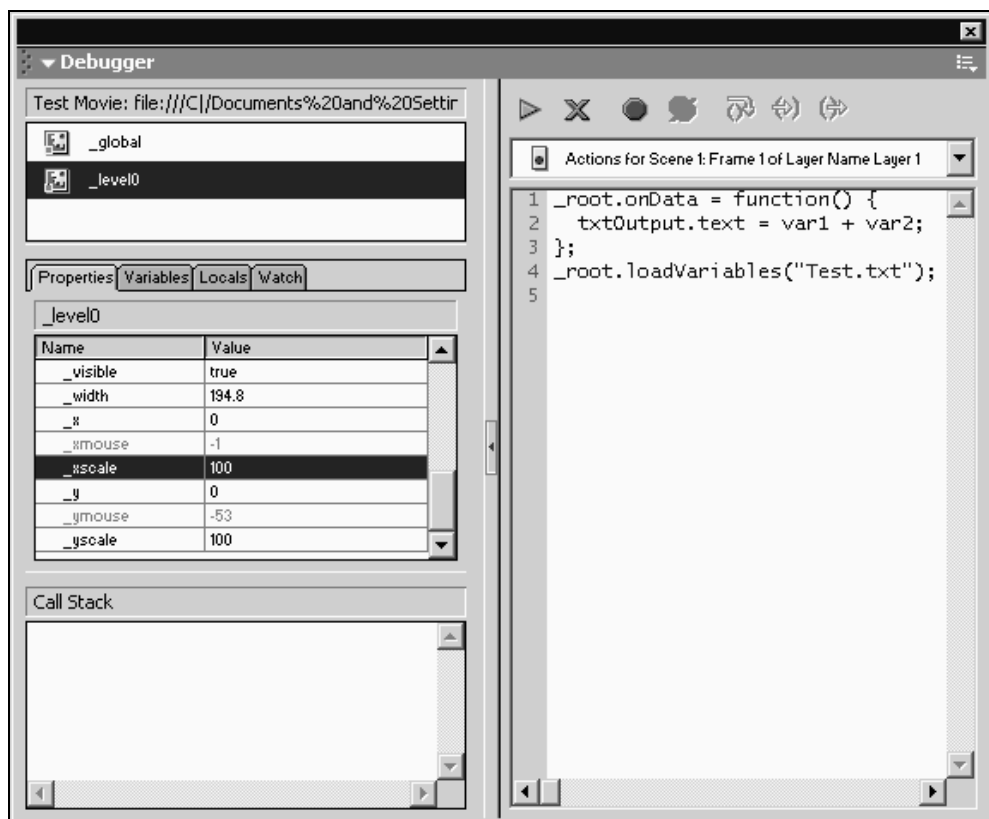


Рис. 25.5. Окно отладчика

Рис. 25.6. Кнопка **Continue** окна отладчика

Окно отладчика очень похоже на панель **Actions**. И там, и здесь в правой части находится текстовая область, где отображается отлаживаемый сценарий. Сами сценарии выбираются в раскрывающемся списке, находящемся прямо над этой текстовой областью. В левой части окна отладчика, в отличие от панели **Actions**, находятся сразу три списка, которые мы рассмотрим далее.

Разделяет эти две части окна отладчика довольно толстая серая полоса, перемещая ее мышью, можно изменять их относительные размеры. На этой полосе также имеется небольшая кнопка, щелкая которую, можно убирать последовательно то одну, то другую часть окна отладчика. Вместо щелчков по кнопке вы можете делать двойные щелчки по самой этой серой линии.

Здесь мы перечислили сходства окна отладчика и панели **Actions**. Теперь поговорим о различиях.

Прежде всего, вы не можете править код сценария, отображенный в окне отладчика. Для этого вам нужно закрыть окно, в котором проигрывается фильм, и исправить требуемый сценарий в панели **Actions**. Однако вы можете выделять код сценария в окне отладчика и копировать его в буфер обмена, а также печатать. Чтобы выделить весь текст в текстовой области, выберите пункт **Select All** в контекстном меню или нажмите комбинацию клавиш <Ctrl>+<A>. Можно выделять текст мышью и клавиатурными комбинациями. Чтобы скопировать выделенный текст в буфер обмена, выберите пункт **Copy** контекстного меню или нажмите комбинацию клавиш <Ctrl>+<C>. Для печати кода сценария выберите пункт **Print** в дополнительном меню окна отладчика.

Выше раскрывающегося списка сценариев находится небольшой инструментарий, содержащий кнопки для управления отладкой. Большинство кнопок мы опишем далее в этой главе, а сейчас рассмотрим только две. Уже знакомая вам кнопка **Continue** (см. рис. 25.6) указывает отладчику продолжить выполнение всех сценариев, вы также можете выбрать пункт **Continue** дополнительного меню или нажать клавишу <F8>. А кнопка **Stop Debugging** (рис. 25.7) заставляет Flash прекратить проигрывание фильма (но закрыть окно, в котором этот фильм проигрывается, вам придется самим), можно также выбрать пункт **Stop Debugging** дополнительного меню или нажать клавишу <F7>.



Рис. 25.7. Кнопка **Stop Debugging** окна отладчика

В самом верху левой части окна отладчика находится небольшая строка статуса. Там отображаются имя и путь файла, в котором сохранен проигрываемый в данный момент фильм. Учтите, что отображается путь экспортированного файла Shockwave/Flash, а не файла документа Flash.

А теперь рассмотрим все инструменты, предлагаемые окном отладчика, более подробно.

Просмотр значений переменных и свойств и списка вызовов

Как было сказано выше, в левой части окна отладчика находятся три списка. Причем один из этих списков имеет четыре вкладки, т. е. фактически целых четыре списка были объединены в один. Рассмотрим эти списки в порядке сверху вниз.

Самый верхний из списков показывает все клипы, входящие в фильм, в том числе, и основной фильм. Этот список имеет иерархический вид, показывающий вложенность клипов друг в друга. Выберите нужный клип, чтобы просмотреть его свойства и объявленные в нем переменные. Также вы можете выбрать объект `_global`, чтобы просмотреть все глобальные переменные.

Значения свойств и переменных просматриваются в среднем списке, том самом, который имеет четыре вкладки. Также с помощью этого списка вы можете задать для свойств и переменных новые значения и сразу же увидеть, что это даст. Рассмотрим все эти четыре списка по очереди.

Вкладка **Properties** открывает список свойств выбранного в верхнем списке клипа и их значений. Этот список состоит из двух колонок: **Name** (имя свойства) и **Value** (значение свойства). Чтобы изменить значение некоторого свойства, дважды щелкните по требуемой строке в колонке **Value**. В строке появится небольшое поле ввода, в котором вы сможете ввести нужное значение; после этого нажмите клавишу `<Enter>` для его сохранения или `<Esc>` для отмены. Имейте в виду, что значения свойств, набранных серым цветом, вы изменять не можете — они доступны только для чтения.

Вкладка **Variables** открывает список переменных уровня клипа. Он также состоит из двух колонок **Name** и **Value** и также позволяет менять значения переменных.

Под вкладкой **Locals** скрывается список локальных переменных. Как вы помните, локальные переменные присутствуют только в функциях, поэтому список **Locals** "вне" функций пуст. В остальном он полностью подобен двум уже рассмотренным спискам.

Имейте в виду, что вы не можете вводить выражения в списки **Properties**, **Variables** и **Locals** в качестве новых значений. Также недопустим ввод массивов и объектов.

А вот под вкладкой **Watch** скрывается совершенно особый список. В этот список вы сами можете добавлять необходимые вам переменные, уровня клипа или локальные, и просматривать или изменять их значения. Пользуясь этим списком, вы можете сформировать набор переменных, значения которых хотите постоянно держать у себя перед глазами.

Добавить переменную в список **Watch** можно двумя способами. Первый, и самый простой, — выделить в списке **Variables** или **Locals** нужную переменную и выбрать в контекстном меню пункт **Watch** или в дополнительном меню — пункт **Add Watch**. После этого в списке **Watch** появится новая строка, соответствующая добавленной переменной. Теперь вы можете просматривать и изменять ее значение, как и в любом другом списке.

Второй способ сложнее. Переключитесь в список **Watch** и выберите пункт **Add** в контекстном меню или пункт **Add Watch** в дополнительном меню. В списке **Watch** появится новая строка. Дважды щелкните по ней мышью

в колонке **Name**, введите в появившееся поле ввода имя переменной и нажмите клавишу <Enter>.

Вы можете удалить ненужную строку из списка **Watch**. Для этого выделите ее и выберите пункт **Remove** в контекстном меню или пункт **Remove Watch** в дополнительном меню.

Выше было сказано, что вы можете изменять значения любой переменной в списке **Watch** (как и в остальных трех списках). Однако этот список также позволяет вам изменить имя переменной. Для этого дважды щелкните по требуемой строке в колонке **Name**. В строке появится небольшое поле ввода, в котором вы сможете ввести новое имя переменной; после этого нажмите клавишу <Enter> для его сохранения или <Esc> для отмены.

Осталось рассмотреть самый нижний список, расположенный в левой части окна отладчика.

В программировании часто применяются так называемые *вложенные вызовы* функций, когда первая функция вызывает вторую, вторая, в свою очередь, — третью и т. д. И часто возникает необходимость отследить, какая функция какую функцию вызывает. Для этого и предназначен нижний список. Он показывает очередность вызова функций, причем очередность эта показана снизу вверх.

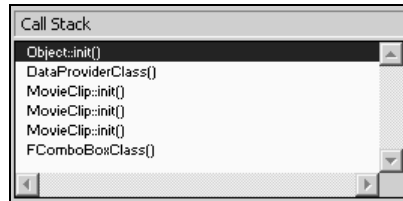


Рис. 25.8. Список очередности вложенных вызовов

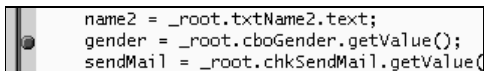
Для примера рассмотрим список очередности, показанный на рис. 25.8. Видно, что первым был конструктор объекта `FComboBoxClass` (это объект раскрывающегося списка `ComboBox`) — он находится в последней строке списка. Он вызвал метод `init` какого-то экземпляра объекта `movieClip`, который вызвал тот же метод уже другого экземпляра объекта `movieClip`. Заканчивает список метод `init` объекта `Object` (на самом деле, метод прототипа объекта `DataProviderClass`) — он находится в первой строке списка.

Использование точек останова

Иногда во время отладки сценария его нужно приостановить на время, чтобы выяснить значения переменных и свойств и посмотреть на сам фильм. Для этого предназначены точки останова.

Установить точки останова можно как в панели **Actions**, так и в окне отладчика. И там, и здесь для этого применяется один и тот же способ.

Чтобы установить точку останова, поставьте текстовый курсор на нужную строку и выберите пункт **Set Breakpoint** контекстного меню или нажмите комбинацию клавиш <Ctrl>+<Shift>+. Созданная точка останова показана на рис. 25.9. Как видите, она помечена большой красной точкой, находящейся слева от текста сценария.



```
name2 = _root.txtName2.text;  
gender = _root.cboGender.getValue();  
sendMail = _root.chkSendMail.getValue();
```

Рис. 25.9. Точка останова

Если вы находитесь в панели **Actions**, то можете нажать кнопку **Debug Options** (рис. 25.10). После нажатия этой кнопки раскроется небольшое меню, в котором вам будет нужно выбрать пункт **Set Breakpoint**.

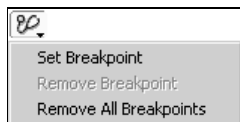


Рис. 25.10. Кнопка **Debug Options** и ее меню

Чтобы установить точку останова, находясь в окне отладчика, нажмите кнопку **Toggle Breakpoint** (рис. 25.11). Конечно, вы также можете выбрать пункт **Set Breakpoint** контекстного меню или нажмите комбинацию клавиш <Ctrl>+<Shift>+.



Рис. 25.11. Кнопка **Toggle Breakpoint** окна отладчика

Как только Flash достигнет точки останова, он приостановит выполнение сценария. Вы можете запустить выполнение дальше, нажав кнопку **Continue** (см. рис. 25.6), просмотреть и изменить значения переменных. Также вы можете оценить то, что успел выполнить ваш сценарий.

Чтобы удалить ненужную точку останова, поставьте текстовый курсор на строку, где она находится, и выберите пункт **Remove Breakpoint** в контекстном меню или еще раз нажмите комбинацию клавиш <Ctrl>+<Shift>+. В панели **Actions** вы можете также выбрать пункт **Remove Breakpoint** меню кнопки **Debug Options**, а в окне отладчика — еще раз нажать кнопку **Toggle Breakpoint**.

Существует также возможность удалить сразу все точки останова, которые вы установили во всех сценариях фильма. Для этого выберите пункт **Remove Breakpoint** в контекстном меню. В панели **Actions** вы можете также выбрать

пункт **Remove All Breakpoints** меню кнопки **Debug Options**, а в окне отладчика — нажать кнопку **Remove All Breakpoints** (рис. 25.12).



Рис. 25.12. Кнопка **Remove All Breakpoints** окна отладчика

Имейте в виду, что установить точку останова можно не на всякую строку, а только на содержащую функциональный код, т. е. код, который содержит какие-то выражения и реально что-то делает. На служебную строку, содержащую, например, фигурную скобку, установить точку останова нельзя.

И еще. Точки останова, которые вы установили в среде Flash, не сохраняются ни в файле документа Flash, ни тем более в экспортированном файле Shockwave/Flash. Это значит, что если вы закроете фильм и откроете его снова, все ваши точки останова пропадут.

Трассировка кода

Трассировкой, как вы помните, называется пошаговое исполнение кода сценария. Трассировка позволяет точно выяснить, как работает написанный вами или другим разработчиком код, и почему он работает не так, как надо.

Для выполнения трассировки кода отладчик Flash предлагает три кнопки, показанные на рис. 25.13. Давайте рассмотрим эти кнопки.



Рис. 25.13. Кнопки трассировки кода

Рассмотрение начнем со средней кнопки. Она называется **Step In** и вызывает выполнение одного выражения языка ActionScript. При этом если данное выражение содержит вызов пользовательской функции, начнет выполняться код этой функции. Вместо нажатия этой кнопки вы можете выбрать пункт **Step In** дополнительного меню окна отладчика или нажать клавишу <F10>.

Самая левая кнопка называется **Step Over**. Она также вызывает выполнение одного выражения, но "захода" в пользовательские функции при этом не происходит. Можно сказать, что при нажатии кнопки **Step Over** Flash "перескакивает" через пользовательские функции. Вместо нажатия этой кнопки вы также можете выбрать пункт **Step Over** дополнительного меню окна отладчика или нажать клавишу <F9>.

Правая кнопка носит название **Step Out** и работает только внутри пользовательских функций. При нажатии этой кнопки Flash быстро завершает вы-

полнение кода функции и "выходит" из нее. Вы также можете выбрать пункт **Step Out** дополнительного меню или нажать клавишу <F11>.

Если вы больше не хотите трассировать код, нажмите кнопку **Continue** (см. рис. 25.6). Можно также выбрать пункт **Continue** в дополнительном меню или нажать клавишу <F8>.

Удаленная отладка фильмов Flash

Отладчик Flash предоставляет еще одну интересную возможность — *удаленную отладку* фильмов. При удаленной отладке фильм загружается не с локального диска, а с Web-сервера. Таким образом, вы можете отлаживать чужие фильмы и приложения, и другие разработчики могут отлаживать ваши (пресловутое разделение труда).

Чтобы удаленная отладка стала возможной, вам следует поместить на Web-сервер вместе с файлом Shockwave/Flash, содержащим фильм, еще один особый файл, содержащий отладочную информацию. Этот файл имеет расширение swd, такое же имя, как у файла фильма, и формируется Flash при экспорте. Если же Flash не найдет на сервере SWD-файл, отладчик не будет работать правильно; в частности, вы не сможете ставить точки останова и трассировать код. Сейчас мы рассмотрим, как создать такой файл и как запустить удаленную отладку.

Сначала нужно экспортировать фильм, задав параметры, разрешающие удаленную отладку. Для этого, прежде всего, откройте нужный документ. В диалоговом окне **Publish Settings** (см. рис. 19.1), на вкладке **Flash** включите флажок **Debugging Permitted**. Как только вы включите этот флажок, станет доступно поле ввода **Password**, где вы сможете ввести пароль. После этого любой, кто захочет отлаживать ваш фильм, должен будет ввести этот пароль. Используйте его, чтобы не давать просматривать ваши сценарии случайным людям.

После этого опубликуйте или экспортируйте фильм. Выложите на сервер сформированные файлы swf и swd. Все, подготовка к удаленной отладке завершена.

Теперь расскажем, как выполняется удаленная отладка фильма. Предположим, кто-то попросил вас выловить ошибки в своем приложении. Также предположим, что все шаги по подготовке приложения Flash к удаленной отладке сделаны правильно.

Прежде всего, вам нужно включить удаленную отладку в самом Flash. Для этого запустите Flash, откройте любой документ (можно, в принципе, оставить пустой, созданный при запуске) и запустите его отладку. Когда на экране появится окно отладчика, проверьте, включен ли пункт-выключатель **Enable Remote Debugging** дополнительного меню. Если он отключен, включите его.

После этого откройте отлаживаемое приложение Flash. Вы можете сделать это, например, в Web-обозревателе, открыв Web-страницу, в которую внедрено это приложение. Также это можно сделать в обычном проигрывателе Flash. Удаленная отладка работает везде вне зависимости от того, где открыт SWF-файл.

После открытия файла swf на экране появится небольшое диалоговое окно **Remote Debug** (рис. 25.14). Это окно выводится проигрывателем Flash и служит для указания, на каком компьютере находится среда Flash. Если оно не открылось, значит, проигрыватель Flash не смог найти соответствующий файл swd. Выберите в контекстном меню проигрывателя пункт **Debugging** — и окно **Remote Debug** появится на экране.

Включите переключатель **Localhost**, если среда Flash установлена на этом же компьютере. Если же она находится на другом компьютере, и он доступен в сети, включите переключатель **Other Machine** и введите IP-адрес этого компьютера в поле ввода **Enter IP Address**. Затем нажмите кнопку **OK** для запуска удаленной отладки или **Cancel** для ее отмены.



Рис. 25.14. Диалоговое окно **Remote Debug**

Если вы нажали кнопку **OK**, то переключитесь после этого в запущенную ранее среду Flash. Введите в поле ввода **Password** диалогового окна **Debugging Password Required** (рис. 25.15) пароль, заданный при экспорте файла Shockwave/Flash и нажмите кнопку **OK**. После этого запустится отладчик Flash, в котором вы сможете отлаживать приложение Flash удаленно.

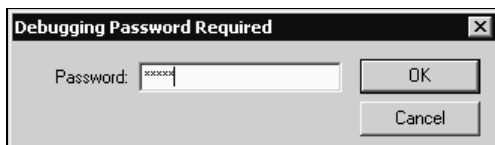


Рис. 25.15. Диалоговое окно **Debugging Password Required**

Заключение

Все в жизни рано или поздно кончается. Кончается зима, кончается терпение, кончается йогурт в пакете, даже "Санта-Барбара" и та закончилась. В этом мире нет ничего вечного, увы! Когда-нибудь закончится существование самой нашей Вселенной (правда, это будет не скоро). Что уж говорить о книгах, даже очень толстых...

Итак, нам больше нечего вам сказать о Macromedia Flash MX. И мы заканчиваем эту книгу.

Это, правда, еще не совсем конец. Еще будут приложения, описывающие язык ActionScript, теги HTML, применяющиеся для внедрения в Web-страницы фильмов Flash, и некоторые примеры, которые вам могут пригодиться. Но повесть (или роман) о Flash закончилась. Поэтому мы и пишем это заключение.

Все ли мы рассказали? Нет, конечно, не все. Мы рассмотрели все возможности Flash, описанные в поставляемом с ним электронном руководстве. Мы упомянули также те возможности, которые почему-то в руководстве не были описаны, вероятно, его авторы куда-то торопились и не доделали свою работу. Чтобы раздобыть эту "секретную" информацию, нам пришлось порыться на Web-сайте Macromedia — уж там-то есть все. И, разумеется, пришлось многое пробовать "методом научного тыка", а иначе нельзя узнать программный продукт, тем более, такой сложный.

Но очень многое осталось "за кадром". Мы не говорили о *расширениях* Flash — дополнительных модулях, подключаемых к основной среде и выполняющих какие-либо специальные задачи. Мы не упоминали о тонкостях работы Flash на компьютерах Apple Macintosh. Мы не описывали многие частные проблемы, с которыми вы вполне можете столкнуться при работе с Flash, так как они появляются достаточно редко, но все-таки появляются. Мы не рассматривали дополнительные программы, поддерживающие формат Shockwave/Flash, в том числе и выпущенные самой фирмой Macromedia. Мы, в конце концов, не говорили о создании серверных приложений и тонкостях языка HTML. Мы о многом не говорили.

Ибо невозможно объять необъятное. Даже в столь толстой книге.

Macromedia Flash MX — мощный программный продукт, который еще не раз преподнесет сюрпризы пользователям (надеемся, не только неприят-

ные). Чтобы овладеть им в полной мере, вам также могут понадобиться дополнительные знания по компьютерным сетям, Интернету, серверному программированию, языку HTML и прочим Web-технологиям.

Все это описано в других книгах. А эта — закончилась.

Поэтому мы прощаемся с вами.

Напоследок — небольшой список полезных интернет-ресурсов (табл. 3.1). Рекомендуем регулярно посещать их, если, конечно, у вас есть доступ в Интернет.

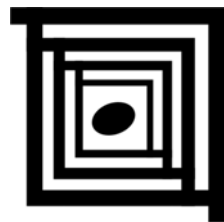
Таблица 3.1. Список интернет-ресурсов по теме книги

Интернет-адрес ресурса	Описание
http://www.macromedia.com/	"Домашний" сайт фирмы Macromedia, разработчика Flash. Посвящен как самому Flash, так и другим продуктам этой фирмы (Dreamweaver, Fireworks, Director и др.)
http://www.macromedia.com/support/flash/technotes.html	Страница, содержащая описание множества проблем, с которыми может столкнуться пользователь Flash, и их решений. Доступ к этой странице весьма затруднен, поэтому мы приводим ссылку на нее здесь
http://msdn.microsoft.com/ie/	"Дом" Internet Explorer. Исчерпывающее описание Web-обозревателя фирмы Microsoft, особенностей поддержки им HTML, CSS, JavaScript и многого другого. К несчастью, читать эту документацию можно только с сайта
http://developer.netscape.com/	"Дом" Netscape Navigator. Исчерпывающее описание Web-обозревателя фирмы Netscape, особенностей поддержки им HTML, CSS, JavaScript и многого другого. Почти вся документация доступна для загрузки в виде ZIP-архивов
http://www.w3c.org/	Сайт самого великого и ужасного комитета WWWW. Все интернет-стандарты в удобном для чтения виде
http://www.flasher.ru	Русский "флэшерский" сайт. Много документации, учебных курсов, примеров
http://subscribe.ru , http://www.maillist.ru	Русские сервера почтовых рассылок. Поищите на них рассылки, посвященные Flash, и сходной тематике
fido7.ru.flash	Группа новостей FIDO, посвященная Flash

А вот теперь действительно конец. До свидания!

Владимир Дронов, vlad@vgi.volsu.ru.

Часть V



Приложения

Приложение 1. Элементы языка ActionScript

Приложение 2. Внедрение фильмов Flash в Web-страницы

Приложение 1

Элементы языка ActionScript

В этом приложении перечислены и описаны все действующие на данный момент элементы языка ActionScript, а также коды клавиш и сообщения об ошибках.

Элементы языка ActionScript

Здесь описаны все элементы языка ActionScript.

///

Кавычки применяются для записи строковых значений. Формат использования:

"<Текст>"

Впервые появились во Flash 4.

--

Оператор декремента (уменьшения значения аргумента на единицу). Формат использования:

--<Аргумент>

<Аргумент>--

Если стоит перед аргументом, то сначала уменьшает его значение, а потом возвращает результат. Если стоит после аргумента, то сначала возвращает его значение (неизмененное), а потом уменьшает его. Выполняется быстрее, чем выражение вида <Аргумент> - 1.

Впервые появился во Flash 4.

++

Оператор инкремента (увеличения значения аргумента на единицу). Формат использования:

++<Аргумент>

<Аргумент>++

Если стоит перед аргументом, то сначала увеличивает его значение, а потом возвращает результат. Если стоит после аргумента, то сначала возвращает его значение (неизмененное), а потом увеличивает его. Выполняется быстрее, чем выражение вида <Аргумент> + 1.

Впервые появился во Flash 4.

!

Оператор логической инверсии (НЕ). Формат использования:

!<Логическое выражение>

Возвращает true, если значение аргумента равно false, и наоборот.

Впервые появился во Flash 4.

!=

Оператор неравенства. Формат использования:

<Аргумент 1> != <Аргумент 2>

Возвращает false, если оба аргумента равны друг другу, и true в противном случае. Строковые, числовые и логические величины сравниваются по значению. Объекты, массивы и функции сравниваются по ссылке, т. е. указывают ли они на один или тот же экземпляр объекта (массив, функцию), или нет. Если сравниваются аргументы различных типов, то выполняется преобразование.

Впервые появился во Flash 5.

!==

Оператор строгого неравенства. Формат использования:

<Аргумент 1> !== <Аргумент 2>

Аналогичен оператору неравенства !=, но, в отличие от него, не выполняет преобразование типов. То есть, возвращается false, если оба аргумента одинаковых типов равны друг другу, и true в противном случае.

Впервые появился во Flash MX.

%

Оператор взятия остатка от деления. Формат использования:

<Аргумент 1> % <Аргумент 2>

Впервые появился во Flash 5. Также поддерживался в среде Flash 4, но при экспорте в формат Shockwave/Flash преобразовывался в выражение $x - \text{int}(x/y) * y$.

%=

Оператор, совмещающий взятие остатка от деления и присваивание. Формат использования:

<Аргумент 1> %= <Аргумент 2>

Аналогичен выражению вида:

<Аргумент 1> = <Аргумент 1> % <Аргумент 2>

и работает быстрее.

Впервые появился во Flash 5. Также поддерживался в среде Flash 4, но при экспорте в формат Shockwave/Flash преобразовывается в выражение `x - int(x/y) * y`.

&

Оператор двоичного умножения (И). Формат использования:

<Аргумент 1> & <Аргумент 2>

Впервые появился во Flash 5. Имейте в виду, что во Flash 4 этот же знак обозначал оператор объединения строк.

&&

Оператор логического И. Формат использования:

<Аргумент 1> && <Аргумент 2>

Возвращает `true`, если оба аргумента равны `true`, и `false` во всех остальных случаях.

Впервые появился во Flash 4.

&=

Оператор, совмещающий двоичное умножение (И) и присваивание. Формат использования:

<Аргумент 1> &= <Аргумент 2>

Аналогичен выражению вида:

<Аргумент 1> = <Аргумент 1> & <Аргумент 2>

но работает быстрее.

Впервые появился во Flash 5.

(и)

Круглые скобки. Служат для изменения приоритета операторов в сложном выражении и для задания аргументов функций.

Впервые появились во Flash 4.

-

Оператор вычитания и изменения знака числа.

Формат использования в качестве оператора вычитания:

<Аргумент 1> - <Аргумент 2>

Формат использования в качестве оператора изменения знака числа:

-<Аргумент>

В этом случае преобразует положительное число в отрицательное и наоборот.

Впервые появился во Flash 4.

*

Оператор умножения. Формат использования:

<Аргумент 1> * <Аргумент 2>

Впервые появился во Flash 4.

***=**

Оператор, совмещающий умножение и присваивание. Формат использования:

<Аргумент 1> *= <Аргумент 2>

Аналогичен выражению вида:

<Аргумент 1> = <Аргумент 1> * <Аргумент 2>

но работает быстрее.

Впервые появился во Flash 4.

,

Запятая. Служит для разделения частей некоторых сложных выражений.

Впервые появилась во Flash 4.

.

Точка. Служит для отделения имени экземпляра объекта от имени свойства или метода.

Впервые появилась во Flash 4.

?:

Условный оператор. Формат использования:

<Аргумент 1> ? <Аргумент 2> : <Аргумент 3>

Если значение Аргумента 1 равно true, возвращает значение Аргумента 2. В противном случае возвращается значение Аргумента 3.

Впервые появился во Flash 4.

/

Оператор деления. Формат использования:

<Аргумент 1> / <Аргумент 2>

Впервые появился во Flash 4.

//

Оператор однострочного комментария. Формат использования:

// <Текст комментария>

Текст комментария может содержать любые символы. Flash игнорирует комментарии при выполнении кода и исключает их при экспорте фильма.

Впервые появился во Flash 1.

/*

Оператор многострочного комментария. Формат использования:

/*

<Текст комментария>

*/

Текст комментария может содержать любые символы. Flash игнорирует комментарии при выполнении кода и исключает их при экспорте фильма.

Впервые появился во Flash 5.

/=

Оператор, совмещающий деление и присваивание. Формат использования:

<Аргумент 1> /= <Аргумент 2>

Аналогичен выражению вида:

<Аргумент 1> = <Аргумент 1> / <Аргумент 2>

но работает быстрее.

Впервые появился во Flash 4.

[и]

Квадратные скобки. Служат для доступа к элементам массива.

Впервые появились во Flash 4.

^

Оператор двоичного исключающего сложения (исключающее ИЛИ). Формат использования:

<Аргумент 1> ^ <Аргумент 2>

Впервые появился во Flash 5.

^=

Оператор, совмещающий двоичное исключающее сложение (исключающее ИЛИ) и присваивание. Формат использования:

<Аргумент 1> ^= <Аргумент 2>

Аналогичен выражению вида:

<Аргумент 1> = <Аргумент 1> ^ <Аргумент 2>

но работает быстрее.

Впервые появился во Flash 5.

{ и }

Фигурные скобки. Служат для создания инициализаторов.

Впервые появились во Flash 5.

/

Оператор двоичного сложения (ИЛИ). Формат использования:

<Аргумент 1> | <Аргумент 2>

Впервые появился во Flash 5.

//

Оператор логического ИЛИ. Формат использования:

<Аргумент 1> || <Аргумент 2>

Возвращает true, если один из аргументов равен true, и false, если оба аргумента равны false.

Впервые появился во Flash 4.

/=

Оператор, совмещающий двоичное сложение (ИЛИ) и присваивание. Формат использования:

<Аргумент 1> |= <Аргумент 2>

Аналогичен выражению вида:

`<Аргумент 1> = <Аргумент 1> | <Аргумент 2>`

но работает быстрее.

Впервые появился во Flash 5.

~

Оператор двоичной инверсии (НЕ). Формат использования:

`~<Аргумент>`

Впервые появился во Flash 5.

+

Оператор сложения и объединения строк. Формат использования:

`<Аргумент 1> + <Аргумент 2>`

Впервые появился во Flash 4. Как оператор объединения строк начал использоваться во Flash 5.

+=

Оператор, совмещающий сложение или объединение строк и присваивание. Формат использования:

`<Аргумент 1> += <Аргумент 2>`

Аналогичен выражению вида:

`<Аргумент 1> = <Аргумент 1> + <Аргумент 2>`

но работает быстрее.

Впервые появился во Flash 4. Как оператор объединения строк начал использоваться во Flash 5.

<

Оператор сравнения "меньше, чем". Формат использования:

`<Аргумент 1> < <Аргумент 2>`

Возвращает `true`, если Аргумент 1 меньше Аргумента 2, и `false` в противном случае. Если сравниваются аргументы различных типов, выполняется преобразование типов.

Впервые появился во Flash 4.

<<

Оператор двоичного сдвига влево. Формат использования:

`<Аргумент> << <Количество разрядов>`

Впервые появился во Flash 5.

<<=

Оператор, совмещающий двоичный сдвиг влево и присваивание. Формат использования:

<Аргумент> <<= <Количество разрядов>

Аналогичен выражению вида:

<Аргумент> = <Аргумент> << <Количество разрядов>

но работает быстрее.

Впервые появился во Flash 5.

<=

Оператор сравнения "не больше, чем". Формат использования:

<Аргумент 1> <= <Аргумент 2>

Возвращает true, если Аргумент 1 меньше Аргумента 2 или равен ему, и false в противном случае. Если сравниваются аргументы различных типов, выполняется преобразование типов.

Впервые появился во Flash 4.

<>

Оператор неравенства. Формат использования:

<Аргумент 1> <> <Аргумент 2>

Возвращает true, если Аргумент 1 не равен Аргументу 2, и false в противном случае. Если сравниваются аргументы различных типов, выполняется преобразование типов.

Впервые появился во Flash 2.

=

Оператор простого присваивания значения переменной. Формат использования:

<Переменная> = <Значение>

Впервые появился во Flash 4.

-=

Оператор, совмещающий вычитание и присваивание. Формат использования:

<Аргумент 1> -= <Аргумент 2>

Аналогичен выражению вида:

```
<Аргумент 1> = <Аргумент 1> - <Аргумент 2>
```

но работает быстрее.

Впервые появился во Flash 4.

==

Оператор равенства. Формат использования:

```
<Аргумент 1> == <Аргумент 2>
```

Возвращает `true`, если Аргумент 1 равен Аргументу 2, и `false` в противном случае. Если сравниваются аргументы различных типов, выполняется преобразование типов.

Впервые появился во Flash 5.

===

Оператор строгого равенства. Формат использования:

```
<Аргумент 1> === <Аргумент 2>
```

Аналогичен оператору равенства `==`, но, в отличие от него, не выполняет преобразование типов, т. е. возвращается `true`, если оба аргумента одинаковых типов равны друг другу, и `false` в противном случае.

Впервые появился во Flash MX.

>

Оператор сравнения "больше, чем". Формат использования:

```
<Аргумент 1> > <Аргумент 2>
```

Возвращает `true`, если Аргумент 1 больше Аргумента 2, и `false` в противном случае. Если сравниваются аргументы различных типов, выполняется преобразование типов.

Впервые появился во Flash 5.

>=

Оператор сравнения "не меньше, чем". Формат использования:

```
<Аргумент 1> >= <Аргумент 2>
```

Возвращает `true`, если Аргумент 1 больше Аргумента 2 или равен ему, и `false` в противном случае. Если сравниваются аргументы различных типов, выполняется преобразование типов.

Впервые появился во Flash 4.

>>

Оператор двоичного сдвига вправо. Формат использования:

<Аргумент> >> <Количество разрядов>

Впервые появился во Flash 5.

>>=

Оператор, совмещающий двоичный сдвиг вправо и присваивание. Формат использования:

<Аргумент> >>= <Количество разрядов>

Аналогичен выражению вида:

<Аргумент> = <Аргумент> >> <Количество разрядов>

и работает быстрее.

Впервые появился во Flash 5.

>>>

Оператор двоичного сдвига вправо с заполнением нулями. Формат использования:

<Аргумент> >>> <Количество разрядов>

Впервые появился во Flash 5.

>>>=

Оператор, совмещающий двоичный сдвиг вправо с заполнением нулями и присваивание. Формат использования:

<Аргумент> >>>= <Количество разрядов>

Аналогичен выражению вида:

<Аргумент> = <Аргумент> >>> <Количество разрядов>

но работает быстрее.

Впервые появился во Flash 5.

Accessibility

Объект, предоставляющий доступ к средствам обеспечения доступности. Позволяет создавать фильмы и приложения Flash, доступные людям с физическими недостатками зрения.

Единственный экземпляр этого объекта под названием `Accesssebility` создается самим Flash.

Впервые появился во Flash MX.

Accessibility.isActive

Метод, возвращающий `true`, если программное обеспечение чтения с экрана активизировано. Не принимает параметров.

Впервые появился во Flash MX.

add

Оператор объединения строк. Формат использования:

`<Аргумент 1> add <Аргумент 2>`

Впервые появился во Flash 4. Во Flash 5 объявлен устаревшим и нерекомендованным к использованию; вместо него рекомендуется применять оператор `+`.

and

Оператор логического И. Формат использования:

`<Аргумент 1> and <Аргумент 2>`

Возвращает `true`, если оба аргумента равны `true`, и `false` во всех остальных случаях.

Впервые появился во Flash 4. Во Flash 5 объявлен устаревшим и нерекомендованным к использованию; вместо него рекомендуется применять оператор `&&`.

arguments

Объект, предоставляющий доступ к массиву параметров, переданных функции. Позволяет создавать функции с переменным числом параметров.

Единственный экземпляр объекта `arguments` под тем же именем — `arguments` — создается самим Flash. Он доступен только в теле функции; фактически переменная `arguments`, содержащая ссылку на этот экземпляр, является локальной для данной функции.

Доступ к аргументам выполняется так же, как к элементам массива:

```
return = arguments[0] + arguments[1] + arguments[2];
```

Впервые появился во Flash MX.

arguments.callee

Свойство, возвращающее ссылку на вызванную функцию. Доступно только для чтения.

Впервые появилось во Flash MX.

arguments.caller

Свойство, возвращающее ссылку на функцию, вызвавшую данную. Доступно только для чтения.

Впервые появилось во Flash MX.

arguments.length

Свойство, возвращающее количество переданных функции параметров, фактически — размер массива `arguments`. Доступно только для чтения.

Впервые появилось во Flash MX.

Array

Объект, позволяющий манипулировать массивами как объектами.

Для создания экземпляров этого объекта используются три варианта конструктора:

```
<Переменная> = new Array();
```

```
<Переменная> = new Array(<Размер>);
```

```
<Переменная> = new Array(<Список элементов, разделенных запятыми>);
```

Если конструктору не было передано никакого параметра, созданный массив будет иметь нулевой размер. Если же вы передадите в качестве параметра одно число, то `ActionScript` посчитает его размером массива и создаст массив, все элементы которого будут иметь тип `undefined`. Также конструктору можно передать несколько (больше двух) параметров, в этом случае он посчитает их значениями элементов нового массива.

```
arr1 = new Array();
```

```
arr2 = new Array(4);
```

```
arr3 = new Array(1, 2, 3, 4);
```

Массив можно также создать, присвоив переменной список элементов:

```
arr3 = [1, 2, 3, 4];
```

Возможно создание многомерных массивов — фактически, массивов, вложенных в массив:

```
martix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
```

Впервые появился во Flash 5.

Array.concat

Метод, позволяющий добавить к текущему массиву несколько новых элементов. Формат использования:

```
<Массив>.concat (<Список элементов, разделенных запятыми>);
```

В качестве параметра может быть передан также другой массив.

Метод возвращает новый массив, являющийся комбинацией текущего массива и добавленных к нему новых элементов.

Впервые появился во Flash 5.

Array.join

Метод, возвращающий строку, содержащую все элементы текущего массива, разделенные особым символом-разделителем. Формат использования:

```
<Массив>.join([<Символ или строка-разделитель>]);
```

Если символ-разделитель не указан, используется запятая.

Впервые появился во Flash 5.

Array.length

Свойство, возвращающее количество элементов массива, так называемый размер массива. Доступно только для чтения.

Впервые появилось во Flash 5.

Array.pop

Метод, возвращающий значение последнего элемента массива и удаляющий его. Не принимает параметров.

Впервые появился во Flash 5.

Array.push

Метод, добавляющий в конце текущего массива несколько новых элементов и возвращающий новый размер массива. Формат использования:

```
<Массив>.push(<Список элементов, разделенных запятыми>);
```

Впервые появился во Flash 5.

Array.reverse

Метод, меняющий порядок элементов массива на противоположный. Не принимает параметров и не возвращает значения.

Впервые появился во Flash 5.

Array.first

Метод, возвращающий значение первого элемента массива и удаляющий его. Не принимает параметров.

Впервые появился во Flash 5.

Array.slice

Метод, создающий и возвращающий новый массив, содержащий некоторые элементы текущего массива. Формат использования:

```
<Массив>.slice(<Начальный элемент>, <Конечный элемент>);
```

Новый массив будет содержать все элементы, начиная Начальным и заканчивая Конечным. Причем Начальный элемент будет включен в этот массив, а Конечный — не будет.

Впервые появился во Flash 5.

Array.sort

Метод, сортирующий массив. Формат использования:

```
<Массив>.sort([<Функция сортировки>]);
```

В качестве единственного параметра передается указатель на так называемую функцию сортировки. Эта функция осуществляет сравнение элементов массива. Она должна принимать два параметра, соответствующие двум сравниваемым элементам массива, а возвращать следующие значения:

- -1, если первый элемент массива меньше второго;
- 0, если оба элемента равны;
- 1, если первый элемент массива больше второго.

Используя функцию сортировки, вы можете сортировать массивы по сложному критерию. Если же вы пропустите единственный параметр этого метода, Flash отсортирует массив, используя оператор сравнения "меньше, чем" (<).

Метод не возвращает значения.

Впервые появился во Flash 5.

Array.sortOn

Метод, сортирующий массив однотипных объектов. Формат использования:

```
<Массив>.sortOn(<Имя свойства>);
```

Единственным параметром этого метода передается имя свойства, по значению которого осуществляется сортировка. Чтобы пояснить это, рассмотрим небольшой пример массива.

```
arr = new Array();  
arr[0] = {id: "FL", name: "Flash"};  
arr[1] = {id: "DW", name: "Dreamweaver"};  
arr[2] = {id: "FW", name: "FireWorks"};
```

Теперь, чтобы отсортировать массив по свойству name, нужно написать такое выражение:

```
arr.sortOn("name");
```

Метод не возвращает значения.

Впервые появился во Flash 5.

Array.splice

Метод, позволяющий удалить выбранные элементы массива и вставить новые. Формат использования:

```
<Массив>.splice(<Точка вставки и удаления>,
```

```
❏ <Количество удаляемых элементов>
```

```
❏ [, <Список вставляемых элементов, разделенных запятыми>]);
```

Первым параметром передается номер элемента, начиная с которого элементы будут удалены и после которого элементы будут вставлены. Вторым параметром передается количество удаляемых элементов, вместе с тем, номер которого был передан первым параметром. Если вы не хотите удалять элементы массива, передайте вторым параметром ноль. Вставляемые элементы массива (если они есть) перечисляются после второго параметра метода.

Метод не возвращает значения.

Впервые появился во Flash 5.

Array.toString

Метод, возвращающий строку, содержащую все элементы текущего массива, разделенные запятой. Метод не принимает параметров.

Впервые появился во Flash 5.

Array.unshift

Метод, добавляющий в начале текущего массива несколько новых элементов и возвращающий новый размер массива. Формат использования:

```
<Массив>.unshift(<Список элементов, разделенных запятыми>);
```

Впервые появился во Flash 5.

asfunction

Особая разновидность гиперссылок, позволяющая при щелчке выполнять любой сценарий ActionScript. Работает только в текстовых блоках Flash. Формат использования:

```
asfunction: <Имя функции>, <Значение параметра>
```

Например, при создании гиперссылки в текстовом блоке Flash вместо адреса было записано:

```
asfunction: showAlert, Щелчок!
```

А к первому кадру фильма был привязан следующий сценарий:

```
function showAlert(alertText) {  
    trace(alertText);  
}
```

После щелчка по такой гиперссылке в окне **Output** появится строка "Щелчок!".

Впервые появилась во Flash 5.

Boolean (функция)

Функция, преобразующая аргумент в логическую величину. При этом она руководствуется следующими правилами:

- ☐ если аргумент — логическое выражение, возвращается его значение;
- ☐ если аргумент — числовое выражение, возвращается `true`, если его значение не равно нулю, и `false` в противном случае;
- ☐ если аргумент — строковое выражение, оно преобразуется в числовое, и возвращается `true`, если его значение не равно нулю, и `false` в противном случае;
- ☐ если аргумент равен `undefined`, возвращается `false`;
- ☐ если аргумент представляет собой ссылку на объект, массив или функцию, возвращается `true`.

Впервые появилась во Flash 5.

Boolean (объект)

Объект, позволяющий манипулировать логическими переменными как объектами.

Для создания экземпляра этого объекта используется конструктор:

```
<Переменная> = new Boolean([<Логическое выражение>]);
```

Если параметр конструктора пропущен, экземпляру объекта `Boolean` присваивается `false`.

```
bool1 = new Boolean();
```

```
bool2 = new Boolean((a > 8) && (g <= 0));
```

Кроме того, можно просто присвоить нужное значение переменной:

```
bool2 = (a > 8) && (g <= 0);
```

Впервые появился во Flash 5.

Boolean.toString

Метод, возвращающий строковое представление значения логической величины — "true" или "false". Не принимает параметров.

Впервые появился во Flash 5.

Boolean.valueOf

Метод, возвращающий значение логической величины. Не принимает параметров.

Впервые появился во Flash 5.

break

Действие, прерывающее выполнение цикла или выражения выбора. Может использоваться только внутри них.

Впервые появилось во Flash 4.

Button

Объект, представляющий кнопку.

Для каждой кнопки, помещенной на рабочий лист в среде Flash или из сценария, создается отдельный экземпляр объекта `Button`. Этот экземпляр получает имя, заданное в редакторе свойств или одном из параметров метода `attachMovie` объекта `MovieClip`.

Впервые появился во Flash MX.

Button._alpha

Свойство, задающее прозрачность кнопки. Допускаются целые числовые значения от 0 (полная прозрачность) до 100 (полная непрозрачность). При этом кнопка остается активной, даже если она совершенно невидима (то есть, для свойства `_alpha` было задано значение 0).

Впервые появилось во Flash MX.

Button.enabled

Свойство, разрешающее или запрещающее доступ к кнопке. Если присвоено значение `true`, кнопка доступна, если `false` — кнопка недоступна и не реагирует на щелчки. По умолчанию кнопка доступна (значение `true`).

Впервые появилось во Flash MX.

Button.getDepth

Метод, возвращающий уровень кнопки в виде целого числа. Не принимает параметров.

Впервые появился во Flash MX.

Button._height

Свойство, задающее высоту кнопки в пикселах.

Впервые появилось во Flash MX.

Button._highquality

Свойство, задающее качество сглаживания растровой графики во всем фильме. Может принимать три числовых значения:

- ☐ 2 — растровая графика сглаживается всегда;
- ☐ 1 — растровая графика сглаживается только тогда, когда фильм не содержит анимации;
- ☐ 0 — растровая графика никогда не сглаживается.

Используйте это свойство, чтобы уменьшить количество системных ресурсов, потребляемых проигрывателем Flash. Это может понадобиться на медленных компьютерах.

Очень странно, но, судя по всему, это свойство просто ссылается на системную переменную `_highquality`, выполняющую ту же функцию. Зачем нужно было создавать еще и свойство `_highquality`, неясно.

Впервые появилось во Flash MX.

Button._name

Свойство, возвращающее имя кнопки, заданное в редакторе свойств, в строковом виде. Доступно только для чтения.

Впервые появилось во Flash MX.

Button.onDragOut

Обработчик события, наступающего, когда пользователь помещает курсор мыши над текущей кнопкой, нажимает левую кнопку мыши и "уводит" мышь прочь. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Button.onDragOver

Обработчик события, наступающего, когда пользователь что-то перетаскивает над текущей кнопкой. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Button.onKeyDown

Обработчик события, наступающего, когда текущая кнопка имеет фокус ввода, и пользователь нажимает клавишу на клавиатуре. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Button.onKeyUp

Обработчик события, наступающего, когда текущая кнопка имеет фокус ввода, и пользователь отпускает нажатую ранее клавишу. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Button.onKillFocus

Обработчик события, наступающего, когда текущая кнопка теряет фокус ввода. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события принимает единственный параметр, передающий ссылку на объект, получивший фокус ввода. Если ни один объект не получил фокус ввода, Flash присваивает этому параметру значение `null`. Функция-обработчик события не возвращает значения.

Впервые появился во Flash MX.

Button.onPress

Обработчик события, наступающего, когда пользователь щелкает текущую кнопку. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Button.onRelease

Обработчик события, наступающего, когда пользователь отпускает текущую кнопку. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Button.onReleaseOutside

Обработчик события, наступающего, когда пользователь отпускает текущую кнопку, причем курсор мыши находится за пределами кнопки. Это может случиться, когда пользователь щелкает кнопку, не отпуская ее, "уводит" мышь прочь и там отпускает кнопку мыши. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Button.onRollOut

Обработчик события, наступающего, когда пользователь "уводит" курсор мыши за пределы кнопки. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Button.onRollOver

Обработчик события, наступающего, когда пользователь помещает курсор мыши на кнопку. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Button.onSetFocus

Обработчик события, наступающего, когда текущая кнопка получает фокус ввода. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события принимает единственный параметр, передающий ссылку на объект, потерявший фокус ввода. Если ни один объект не имел ранее фокуса ввода, Flash присваивает этому параметру значение `null`. Функция-обработчик события не возвращает значения.

Впервые появился во Flash MX.

Button._parent

Свойство, возвращающее ссылку на внешний клип, содержащий текущую кнопку. Доступно только для чтения.

```
clip = someButton._parent;
```

Может быть использовано несколько раз:

```
clip = someButton._parent._parent._parent;
```

Впервые появилось во Flash MX.

Button._quality

Свойство, задающее качество графики во всем фильме. Может принимать четыре строковых значения:

- ☐ "LOW" — низкое качество, ни векторная, ни растровая графика не сглаживается;
- ☐ "MEDIUM" — среднее качество, векторная графика сглаживается, растровая не сглаживается;
- ☐ "HIGH" — высокое качество, векторная графика сглаживается, если фильм не содержит анимации, растровая не сглаживается (значение по умолчанию);
- ☐ "BEST" — наилучшее качество, и векторная, и растровая графика сглаживается.

Используйте это свойство, чтобы уменьшить количество системных ресурсов, потребляемых проигрывателем Flash. Это может понадобиться на медленных компьютерах.

Очень странно, но, судя по всему, это свойство просто ссылается на системную переменную `_quality`, выполняющую ту же функцию. Зачем нужно было создавать еще и свойство `_quality`, неясно.

Впервые появилось во Flash MX.

Button._rotation

Свойство, задающее угол поворота кнопки в градусах.

Впервые появилось во Flash MX.

Button._soundbuftime

Свойство, задающее размер буфера звука в секундах. Этот буфер используется при загрузке потоковых звуков для того, чтобы обеспечить их плавное воспроизведение.

Это свойство просто ссылается на системную переменную `_soundbuftime`, выполняющую ту же функцию. Зачем нужно было создавать еще и свойство `_soundbuftime`, неясно.

Впервые появилось во Flash MX.

Button.tabEnabled

Свойство, задающее, будет ли кнопка включена в порядок обхода по нажатиям клавиш `<Tab>` и `<Shift>+<Tab>`.

Если значение этого свойства равно `true` или `undefined` или свойству `tabIndex` присвоено целое число, то кнопка включается в порядок обхода. В противном случае она исключается из порядка обхода, но все еще может быть нажата щелчком мыши. Значение по умолчанию — `undefined`.

Впервые появилось во Flash MX.

Button.tabIndex

Свойство, задающее место текущей кнопки в порядке обхода. Может быть любым неотрицательным целым числом.

Если это свойство равно `undefined` и свойство `tabEnabled` не равно `false`, Flash сам формирует порядок обхода, зависящий от порядка, в котором кнопки и клипы были помещены на рабочий лист. Чтобы задать свой порядок обхода, присвойте всем элементам управления позиции в этом порядке, используя свойство `tabIndex`. Имейте в виду, что этот порядок не зависит от вложенности одних элементов управления в другие, т. е. он "плоский".

Значение по умолчанию — `undefined`.

Впервые появилось во Flash MX.

Button._target

Свойство, возвращающее путь экземпляра текущей кнопки. Доступно только для чтения.

Впервые появилось во Flash MX.

Button.trackAsMenu

Свойство, задающее, будет ли кнопка откликаться на события мыши, происходящие в других кнопках.

Если это свойство равно `false`, то кнопка не будет откликаться на события, которые в данный момент происходят в других кнопках. Так, если вы перемещаете кнопку над другими кнопками, эти другие кнопки не будут реагировать на события `onRollOver` и `onRollOut`, даже не будут изменять свой вид. Если же вы установите это свойство в `true`, кнопка будет реагировать на такие события во всех случаях.

Значение по умолчанию — `false`.

Это свойство можно также выставить в редакторе свойств, воспользовавшись раскрывающимся списком **Options for Buttons**. Пункт **Track as Button** эквивалентен значению `false`, а пункт **Track as Menu Item** — значению `true`.

Впервые появилось во Flash MX.

Button._url

Свойство, возвращающее интернет-адрес файла Shockwave/Flash, содержащего образец текущей кнопки. Доступно только для чтения.

Впервые появилось во Flash MX.

Button.useHandCursor

Свойство, задающее форму курсора мыши. Если равно `true`, то курсор мыши при наведении на текущую кнопку примет форму "перста указующего", если `false` — останется в виде стрелки. Значение по умолчанию — `true`.

Впервые появилось во Flash MX.

Button._visible

Свойство, позволяющее скрыть кнопку. Если равно `true`, то кнопка видна и воспринимает щелчки, если `false` — не видна и не доступна. Значение по умолчанию — `true`.

Впервые появилось во Flash MX.

Button._width

Свойство, задающее ширину кнопки в пикселах.

Впервые появилось во Flash MX.

Button._x

Свойство, задающее горизонтальную координату точки фиксации текущей кнопки в пикселах. Отсчет ведется от левого верхнего угла, если кнопка находится в основном фильме, и от точки фиксации, если она вложена во встроенный клип.

Впервые появилось во Flash MX.

Button._xmouse

Свойство, возвращающее горизонтальную координату курсора мыши относительно точки фиксации текущей кнопки в пикселах. Доступно только для чтения.

Впервые появилось во Flash MX.

Button._xscale

Свойство, задающее масштабирование текущей кнопки по горизонтали в процентах.

Впервые появилось во Flash MX.

Button._y

Свойство, задающее вертикальную координату точки фиксации текущей кнопки в пикселах. Отсчет ведется от левого верхнего угла, если кнопка находится в основном фильме, и от точки фиксации, если она вложена во встроенный клип.

Впервые появилось во Flash MX.

Button._ymouse

Свойство, возвращающее вертикальную координату курсора мыши относительно точки фиксации текущей кнопки в пикселах. Доступно только для чтения.

Впервые появилось во Flash MX.

Button._yscale

Свойство, задающее масштабирование текущей кнопки по вертикали в процентах.

Впервые появилось во Flash MX.

call

Действие, выполняющее сценарий, привязанный к кадру с заданным номером, без перемещения указателя на этот кадр. Формат использования:

```
call(<Номер кадра>);
```

Впервые появилось во Flash 4. Начиная с Flash 5, не рекомендовано к использованию; вместо него рекомендуется употреблять пользовательские функции.

call function

Это действие существует только в виде пункта иерархического списка панели **Actions**. Применяется для вызова пользовательской функции или метода пользовательского объекта.

Впервые появилось во Flash MX.

case

Действие, применяющееся в выражениях выбора для создания условия. Формат:

```
case <Условие> : <Фрагмент кода>
```

Если *Условие* истинно, выполняется фрагмент кода.

Впервые появилось во Flash 4.

chr

Функция, принимающая код символа ASCII и возвращающая его строковое представление. Формат использования:

```
chr(<Код символа>)
```

Впервые появилась во Flash 4. Начиная с Flash 5, объявлена устаревшей и не рекомендована к использованию; вместо нее рекомендуется употреблять метод `fromCharCode` объекта `String`.

clearInterval

Функция, удаляющая ранее созданный таймер. Формат использования:

```
clearInterval(<Идентификатор таймера>)
```

Функция не возвращает никакого значения.

Впервые появилась во Flash MX.

Color

Объект, позволяющий управлять цветом какого-либо клипа.

Для создания экземпляра этого объекта используется конструктор:

```
<Переменная> = new Color(<Клип>);
```

В качестве единственного параметра конструктора этого объекта задается имя клипа, цветом которого вы хотите управлять.

```
myColor = new Color(wheel);
```

Впервые появился во Flash 5.

Color.getRGB

Метод, возвращающий значение цвета. Не принимает ни одного параметра.

Впервые появился во Flash 5.

Color.getTransform

Метод, возвращающий ссылку на вспомогательный объект, описывающий преобразование цвета для клипа. Не принимает ни одного параметра.

Впервые появился во Flash 5.

Color.setRGB

Метод, задающий новое значение цвета. Формат использования:

```
<Экземпляр объекта Color>.setRGB(<Значение цвета>);
```

Цвет задается в шестнадцатеричном виде, в формате 0xRRGGBB, где RR — красная составляющая, GG — зеленая, а BB — синяя. Так, 0x00FF00 задает зеленый цвет, а 0xFFFFFF — белый.

Метод не возвращает значения.

Впервые появился во Flash 5.

Color.setTransform

Метод, задающий преобразования цвета для клипа с помощью особого вспомогательного объекта. Формат использования:

```
<Экземпляр объекта Color>.setTransform(<Экземпляр объекта>);
```

Преобразования цвета задаются с помощью экземпляра объекта `Object`, имеющего особые свойства. Все эти свойства перечислены в табл. П1.1.

Таблица П1.1. Свойства вспомогательного объекта цветowych преобразований

Свойство	Описание
ra	Процентное изменение красной составляющей, от -100 до 100
rb	Изменение красной составляющей, от -255 до 255
ga	Процентное изменение зеленой составляющей, от -100 до 100
gb	Изменение зеленой составляющей, от -255 до 255
ba	Процентное изменение синей составляющей, от -100 до 100
bb	Изменение синей составляющей, от -255 до 255
aa	Процентное изменение прозрачности, от -100 до 100
ab	Изменение прозрачности, от -255 до 255

Метод не возвращает значения.

Впервые появился во Flash 5.

continue

Действие, перезапускающее цикл. При этом обязательно выполняется проверка условия окончания цикла, если имеет место цикл с условием, изменение значения счетчика в случае цикла со счетчиком или выбор нового свойства объекта или элемента массива в случае цикла просмотра. Может использоваться только внутри циклов.

Впервые появилось во Flash 4.

CustomActions

Объект, позволяющий фильму или приложению Flash, проигрывающемуся в среде Flash, управлять позициями списка панели **Actions**, заданными пользователем. Такие позиции описываются в файле XML и служат для облегчения использования компонентов, созданных другими пользователями Flash. Подробнее об этом было написано в *главе 23*. Но, используя объект `CustomActions`, вы можете создавать их с помощью сценариев.

Единственный экземпляр этого объекта под именем `CustomActions` создается самим Flash.

Впервые появился во Flash MX.

CustomActions.get

Метод, возвращающий код XML, описывающий позицию списка панели **Actions**, созданную пользователем. Формат использования:

```
CustomActions.get(<Имя позиции>);
```

Имя позиции задается в строковом формате.

```
xml = CustomActions.get("SoundVolume");
```

Впервые появился во Flash MX.

CustomActions.install

Метод, создающий новую позицию списка панели **Actions**. Формат использования:

```
CustomActions.install(<Имя позиции>, <Описание позиции в формате XML>);
```

Имя позиции и Описание позиции задаются в строковом формате. Если позиция с таким именем уже существует, она перезаписывается новым кодом.

Если добавление прошло успешно, возвращается `true`. В противном случае возвращается `false`.

При добавлении новых позиций с помощью метода `install` в каталоге, где хранятся XML-файлы с описаниями пользовательских позиций, создается новый файл с именем, совпадающим с заданным в первом параметре именем позиции. Так что созданные вами позиции сохраняются, что можно использовать для создания своеобразных программ установки компонентов Flash.

Впервые появился во Flash MX.

CustomActions.list

Метод, возвращающий список всех созданных пользователем позиций списка панели **Actions** в виде массива. Не принимает параметров.

Впервые появился во Flash MX.

CustomActions.uninstall

Метод, удаляющий добавленную ранее методом `install` позицию списка панели **Actions**. Формат использования:

```
CustomActions.uninstall(<Имя позиции>);
```

Имя позиции задается в строковом формате.

Если удаление прошло успешно, возвращается `true`. Если позиция с таким именем не была обнаружена, возвращается `false`.

```
flag = CustomActions.uninstall("SoundVolume");
```

Впервые появился во Flash MX.

DataProviderClass

Объект-поставщик данных для компонентов `FComboBox` и `FListBox`. Служит для описания набора пунктов этих списков.

Для создания экземпляра этого объекта используется конструктор:

```
<Переменная> = new DataProviderClass();
```

Впервые появился во Flash MX.

DataProviderClass.addItem

Метод, добавляющий новый пункт в конец списка объекта-поставщика данных. Формат использования:

```
<Поставщик данных>.addItem(<Пункт>);
```

Пункт задается экземпляром объекта `Object`, содержащим свойства `label` (название пункта) и `data` (значение пункта). Название пункта задается в строковом виде. Значение пункта, которое вы можете задать, будет возвращаться при выборе данного пункта как значение списка. Оно может быть любого типа, поддерживаемого Flash.

Не возвращает значения.

Впервые появился во Flash MX.

DataProviderClass.addItemAt

Метод, добавляющий новый пункт в заданную позицию списка объекта-поставщика данных. Формат использования:

```
<Поставщик данных>.addItemAt(<Номер добавляемого пункта>, <Пункт>);
```

Пункт задается экземпляром объекта `Object`, содержащим свойства `label` (название пункта) и `data` (значение пункта). Название пункта задается в строковом виде. Значение пункта, которое вы можете задать, будет возвращаться при выборе данного пункта как значение списка. Оно может быть любого типа, поддерживаемого Flash.

Не возвращает значения.

Впервые появился во Flash MX.

DataProviderClass.getItemAt

Метод, возвращающий пункт с заданным номером списка объекта-поставщика данных. Формат использования:

```
<Поставщик данных>.getItemAt(<Номер пункта>);
```

Заданный пункт возвращается в виде экземпляра объекта `Object`, содержащего свойства `label` (название пункта) и `data` (значение пункта).

Впервые появился во Flash MX.

DataProviderClass.getLength

Метод, возвращающий количество пунктов списка объекта-поставщика данных. Не принимает параметров.

Впервые появился во Flash MX.

DataProviderClass.removeAll

Метод, удаляющий все пункты в списке объекта-поставщика данных. Не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

DataProviderClass.removeItemAt

Метод, удаляющий пункт с заданным номером в списке объекта-поставщика данных. Формат использования:

```
<Поставщикданных>.removeItemAt(<Номер пункта>);
```

Возвращает удаленный пункт в виде экземпляра объекта `Object`, содержащего свойства `label` (название пункта) и `data` (значение пункта).

Впервые появился во Flash MX.

DataProviderClass.replaceItemAt

Метод, заменяющий пункт с заданным номером в списке объекта-поставщика данных на другой пункт. Формат использования:

```
<Поставщик данных>.replaceItemAt(<Номер заменяемого пункта>, <Пункт>);
```

Новый пункт задается экземпляром объекта `Object`, содержащим свойства `label` (название пункта) и `data` (значение пункта). Название пункта задается в строковом виде. Значение пункта, которое вы можете задать, будет возвращаться при выборе данного пункта как значение списка. Оно может быть любого типа, поддерживаемого Flash.

Не возвращает значения.

Впервые появился во Flash MX.

DataProviderClass.sortItemsBy

Метод, сортирующий пункты списка объекта-поставщика данных. Формат использования:

```
<Поставщик данных>.sortItemsBy("label"|"data", "ASC"|"DESC");
```

Первым параметром задается поле, по которому будут сортироваться пункты: `"label"` (название пункта) или `"data"` (значение пункта). Вторым пара-

метром задается порядок сортировки: "ASC" (по возрастанию) или "DESC" (по убыванию).

Не возвращает значения.

Впервые появился во Flash MX.

Date

Объект, позволяющий манипулировать величинами даты и времени как объектами. Дата и время хранится в виде обычного числа, которое Flash обрабатывает особым образом.

Для создания экземпляра этого объекта используется конструктор:

```
<переменная> = new Date([<Год>, <Месяц>, [<Число>[, <Часы>[, <Минуты>  
✂[, <Секунды>[, <Миллисекунды>]]]]]])
```

Краткое описание всех параметров конструктора этого объекта:

- Год может быть задан двумя или четырьмя цифрами. С четырьмя цифрами все просто; если же год задан двумя цифрами, то значение 0 соответствует 1900 году, а 99 — 1999 году;
- Месяц задается числом от 0 (январь) до 11 (декабрь);
- Число задается числом от 1 до 31;
- Часы задаются числом от 0 (полночь) до 23;
- Минуты и Секунды задаются числом от 0 до 59;
- Миллисекунды задаются числом от 0 до 999.

Если же ни один из параметров не указан, в экземпляр объекта *Date* заносится значение текущей даты.

Впервые появился во Flash 5.

Date.getDate

Метод, возвращающий число. Не принимает параметров.

Впервые появился во Flash 5.

Date.getDay

Метод, возвращающий число, обозначающее день недели. При этом 0 обозначает воскресенье, 1 — понедельник, а 6 — субботу. Не принимает параметров.

Впервые появился во Flash 5.

Date.getFullYear

Метод, возвращающий год в виде четырехзначного числа. Не принимает параметров.

Впервые появился во Flash 5.

Date.getHours

Метод, возвращающий часы. Не принимает параметров.

Впервые появился во Flash 5.

Date.getMilliseconds

Метод, возвращающий миллисекунды. Не принимает параметров.

Впервые появился во Flash 5.

Date.getMinutes

Метод, возвращающий минуты. Не принимает параметров.

Впервые появился во Flash 5.

Date.getMonth

Метод, возвращающий число, обозначающее месяц. При этом 0 обозначает январь, 1 — февраль, а 11 — декабрь. Не принимает параметров.

Впервые появился во Flash 5.

Date.getSeconds

Метод, возвращающий секунды. Не принимает параметров.

Впервые появился во Flash 5.

Date.getTime

Метод, возвращающий текущее время как количество миллисекунд, прошедших с полночи 1 января 1970 года по универсальному времени. Может использоваться для сравнения двух значений дат. Не принимает параметров.

Впервые появился во Flash 5.

Date.getTimezoneOffset

Метод, возвращающий разницу между локальным и универсальным временем в минутах. Локальное время задается в региональных настройках операционной системы клиентского компьютера. Не принимает параметров.

Впервые появился во Flash 5.

Date.getUTCDate

Метод, возвращающий число по универсальному времени. Не принимает параметров.

Впервые появился во Flash 5.

Date.getUTCDay

Метод, возвращающий число, обозначающее день недели, по универсальному времени. При этом 0 обозначает воскресенье, 1 — понедельник, а 6 — субботу. Не принимает параметров.

Впервые появился во Flash 5.

Date.getUTCFullYear

Метод, возвращающий год по универсальному времени в виде четырехзначного числа. Не принимает параметров.

Впервые появился во Flash 5.

Date.getUTCHours

Метод, возвращающий часы по универсальному времени. Не принимает параметров.

Впервые появился во Flash 5.

Date.getUTCMilliseconds

Метод, возвращающий миллисекунды по универсальному времени. Не принимает параметров.

Впервые появился во Flash 5.

Date.getUTCMinutes

Метод, возвращающий минуты по универсальному времени. Не принимает параметров.

Впервые появился во Flash 5.

Date.getUTCMonth

Метод, возвращающий число, обозначающее месяц, по универсальному времени. При этом 0 обозначает январь, 1 — февраль, а 11 — декабрь. Не принимает параметров.

Впервые появился во Flash 5.

Date.getUTCSeconds

Метод, возвращающий секунды по универсальному времени. Не принимает параметров.

Впервые появился во Flash 5.

Date.getYear

Метод, возвращающий год. При этом для получения значения года из четырехзначного значения года вычитается 1900; так 2002 год будет обозначен числом 102. Не принимает параметров.

Впервые появился во Flash 5.

Date.setDate

Метод, задающий число. Формат использования:

```
<Экземпляр объекта Date>.setDate(<Число>);
```

Число задается целым числом от 1 до 31.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setFullYear

Метод, задающий год в четырехзначном формате. Формат использования:

```
<Экземпляр объекта Date>.setFullYear(<Год>[, <Месяц>[, <Число>]]);
```

Год задается четырехзначным числом, Месяц — числом от 0 (январь) до 11 (декабрь), а Число — числом от 1 до 31.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setHours

Метод, задающий часы. Формат использования:

```
<Экземпляр объекта Date>.setHours(<Часы>);
```

Часы задаются целым числом от 0 (полночь) до 23.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setMilliseconds

Метод, задающий миллисекунды. Формат использования:

```
<Экземпляр объекта Date>.setMilliseconds(<Миллисекунды>);
```

Миллисекунды задаются целым числом от 0 до 999.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setMinutes

Метод, задающий минуты. Формат использования:

```
<Экземпляр объекта Date>.setMinutes(<Минуты>);
```

Минуты задаются целым числом от 0 до 59.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setMonth

Метод, задающий месяц. Формат использования:

```
<Экземпляр объекта Date>.setMonth(<Месяц>[, <Число>]);
```

Месяц задается числом от 0 (январь) до 11 (декабрь), Число — числом от 1 до 31.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setSeconds

Метод, задающий секунды. Формат использования:

```
<Экземпляр объекта Date>.setSeconds(<Секунды>);
```

Секунды задаются целым числом от 0 до 59.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setTime

Метод, задающий текущее время как количество миллисекунд, прошедших с полночи 1 января 1970 года по универсальному времени. Формат использования:

```
<Экземпляр объекта Date>.setTime(<Миллисекунды>);
```

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setUTCDate

Метод, задающий число по универсальному времени. Формат использования:

```
<Экземпляр объекта Date>.setUTCDate(<Число>);
```

Число задается целым числом от 1 до 31.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setUTCFullYear

Метод, задающий год в четырехзначном формате по универсальному времени. Формат использования:

```
<Экземпляр объекта Date>.setUTCFullYear(<Год>[, <Месяц>[, <Число>]]);
```

Год задается четырехзначным числом, Месяц — числом от 0 (январь) до 11 (декабрь), а Число — числом от 1 до 31.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setUTCHours

Метод, задающий часы по универсальному времени. Формат использования:

```
<Экземпляр объекта Date>.setUTCHours(<Часы>[, <Минуты>[, <Секунды>  
[, <Миллисекунды>]]]);
```

Часы задаются целым числом от 0 (полночь) до 23, Минуты и Секунды — числом от 0 до 59, а Миллисекунды — числом от 0 до 999.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setUTCMilliseconds

Метод, задающий миллисекунды по универсальному времени. Формат использования:

```
<Экземпляр объекта Date>.setUTCMilliseconds(<Миллисекунды>);
```

Миллисекунды задаются целым числом от 0 до 999.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setUTCMinutes

Метод, задающий минуты по универсальному времени. Формат использования:

```
<Экземпляр объекта Date>.setUTCMinutes(<Минуты>[, <Секунды>
```

```
    [, <Миллисекунды>]]);
```

Минуты и Секунды задаются числом от 0 до 59, Миллисекунды — числом от 0 до 999.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setUTCMonth

Метод, задающий месяц по универсальному времени. Формат использования:

```
<Экземпляр объекта Date>.setUTCMonth(<Месяц>[, <Число>]);
```

Месяц задается числом от 0 (январь) до 11 (декабрь), Число — числом от 1 до 31.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setUTCSeconds

Метод, задающий секунды по универсальному времени. Формат использования:

```
<Экземпляр объекта Date>.setUTCSeconds(<Секунды>[, <Миллисекунды>]);
```

Секунды задаются целым числом от 0 до 59, Миллисекунды — числом от 0 до 999.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.setYear

Метод, задающий год. Формат использования:

```
<Экземпляр объекта Date>.setYear(<Год>);
```

Год задается целым числом. Если оно находится в диапазоне между 0 и 99, то для получения значения года Flash прибавляет к нему 1900.

Метод возвращает новое значение времени в миллисекундах, прошедших с полночи 1 января 1970 года.

Впервые появился во Flash 5.

Date.toString

Метод, возвращающий строковое представление значения даты. Не принимает параметров.

Очень странно, но при выполнении этого метода Flash не принимает во внимание региональные настройки операционной системы. Дата выводится в некоем "универсальном" формате.

Впервые появился во Flash 5.

Date.UTC

Конструктор объекта Date. Позволяет задать дату и время в универсальном формате.

```
<переменная> = new Date(Date.UTC(<Год>, <Месяц>, [<Число>[, <Часы>  
✂[, <Минуты>[, <Секунды>[, <Миллисекунды>]]]]))
```

Краткое описание всех параметров этого конструктора:

- Год может быть задан двумя или четырьмя цифрами. С четырьмя цифрами все ясно; если же год задан двумя цифрами, то значение 0 соответствует 1900 году, а 99 — 1999 году;
- Месяц задается числом от 0 (январь) до 11 (декабрь);
- Число задается числом от 1 до 31;
- Часы задаются числом от 0 (полночь) до 23;
- Минуты и Секунды задаются числом от 0 до 59;
- Миллисекунды задаются числом от 0 до 999.

Впервые появился во Flash 5.

default

Действие, применяющееся в выражениях выбора для создания блока, который выполняется, если ни одно условие не выполнилось. Формат:

```
case : <Фрагмент кода>
```

Впервые появилось во Flash MX.

delete

Оператор, удаляющий экземпляр объекта. Формат использования:

```
delete <Экземпляр объекта>;
```

Этот оператор не может быть применен для удаления экземпляров объектов, созданных самим Flash.

Впервые появился во Flash 5.

do... while

Действия, применяемые для задания цикла с постусловием. Формат использования:

```
do {  
    Тело цикла  
} while (<Условие>;
```

Тело цикла выполняется до тех пор, пока Условие остается истинным.

Впервые появились во Flash 4.

duplicateMovieClip

Действие, создающее копию клипа. Формат использования:

```
duplicateMovieClip(<Путь исходного клипа>, "<Имя нового клипа>",  
    ⚡<Уровень нового клипа по отношению к исходному>);
```

Проигрывание нового клипа всегда начинается с первого кадра, независимо от того, на каком кадре исходного клипа находился указатель. Содержимое переменных исходного клипа не копируется в переменные нового клипа. Если исходный клип был удален, новый клип также удаляется.

```
duplicateMovieClip(_root.car.wheel1, "wheel2", 0);
```

Для удаления созданной копии клипа воспользуйтесь действием `removeMovieClip`.

Впервые появилось во Flash 4.

else

Действие, используемое в условных выражениях для создания блока, исполняемого при невыполнении условия. Формат использования:

`else`

```
[{ } <Фрагмент кода> { }]
```

Впервые появилось во Flash 4.

else if

Действие, используемое для создания многозвенных условных выражений. Формат использования:

```
else if (<Условие>) { <Фрагмент кода> }
```

Если *Условие* истинно, выполняется *Фрагмент кода*.

```
if (a<0) {  
    s = "Отрицательное число."  
}  
else if (a=0) {  
    s = "Ноль."  
}  
else if (a>0) {  
    s = "Положительное число."  
}  
}
```

Впервые появилось во Flash 4.

#endinitclip

Действие, задающее конец кода инициализации компонента, заданного действием `#initclip`.

Впервые появилось во Flash MX.

eq

Оператор равенства. Формат использования:

```
<Аргумент 1> eq <Аргумент 2>
```

Возвращает `true`, если строковое представление *Аргумента 1* равно строковому представлению *Аргумента 2*, и `false` в противном случае.

Впервые появился во Flash 4. Во Flash 5 объявлен устаревшим и нерекомендованным к использованию; вместо него рекомендуется применять оператор `==`.

escape

Функция, возвращающая строку, закодированную для передачи методом GET. Формат использования:

```
escape(<Строковое выражение>);
```

Впервые появилась во Flash 5.

eval

Функция, принимающая строку, представляющую собой выражение, и возвращающая его результат. Формат использования:

```
eval(<Строка, содержащая выражение>);
```

Если выражение не может быть вычислено (например, содержит ошибки), возвращается `undefined`.

```
x = 3;
```

```
y = 6;
```

```
c = eval("x * y");
```

Переменная `c` будет содержать 18.

Впервые появилась во Flash 4 с некоторыми ограничениями (не поддерживался доступ к свойствам объектов). Во Flash 5 и более поздних версиях поддерживается полностью.

evaluate

Это действие существует только в виде пункта иерархического списка панели **Actions**. Применяется для написания различных выражений.

Впервые появилось во Flash 5.

false

Ключевое слово, обозначающее "ложь" — одно из значений, которые может принимать логическая переменная.

Впервые появилось во Flash 5.

FCheckBox

Объект, представляющий флажок Flash. Реализован как образец-клип.

Для каждого флажка, помещенного на рабочий лист в среде Flash, создается отдельный экземпляр объекта `FCheckBox`. Этот экземпляр получает имя, заданное для флажка в редакторе свойств.

Впервые появился во Flash MX.

FCheckBox.setEnabled

Метод, возвращающий `true`, если флажок доступен, и `false` в противном случае. Не принимает параметров.

Впервые появился во Flash MX.

CheckBox.getLabel

Метод, возвращающий текстовую надпись флажка в строковом виде. Не принимает параметров.

Впервые появился во Flash MX.

CheckBox.getValue

Метод, возвращающий `true`, если флажок включен, и `false` в противном случае. Не принимает параметров.

Впервые появился во Flash MX.

CheckBox.registerSkinElement

Метод, присваивающий одному из фрагментов "шкур" флажка одно из свойств объекта `FStyleFormat`. После этого вы можете менять цвет или другие параметры фрагмента "шкур", задавая нужные значения этого свойства. Формат использования:

```
<Флажок>.registerSkinElement(<Имя фрагмента>, <Свойство>);
```

Каждый фрагмент "шкур" есть образец-клип, хранящийся в особой папке библиотеки. Имя фрагмента — это имя экземпляра этого образца-клипа.

Не возвращает значения.

```
chkSendMeAMail.registerSkinElement(shadow_mc, "shadow");
```

Впервые появился во Flash MX.

CheckBox.setChangeHandler

Метод, задающий функцию-обработчик изменения состояния флажка. Формат использования:

```
<Флажок>.setChangeHandler("<Имя функции>"[, <Расположение функции>]);
```

Расположение функции — это путь к объекту, где реализована нужная нам функция.

Функция-обработчик события должна принимать единственный параметр — ссылку на компонент, в котором произошло это событие.

Не возвращает значения.

```
chkSendMeAMail.setChangeHandler("chkSendMeAMail_click");
```

```
chkSendMeAMail.setChangeHandler("chkSendMeAMail_click", _root);
```

Впервые появился во Flash MX.

FCheckBox.setEnabled

Метод, разрешающий или запрещающий доступ к флажку. Формат использования:

```
<Флажок>.setEnabled(<Логическое выражение>);
```

Если передано значение `true`, доступ к флажку разрешается. Если передать `false`, флажок станет недоступен.

Не возвращает значения.

Впервые появился во Flash MX.

FCheckBox.setLabel

Метод, задающий текстовую надпись для флажка. Формат использования:

```
<Флажок>.setLabel(<Текстовая надпись>);
```

Текстовая надпись передается в строковом виде.

Не возвращает значения.

Впервые появился во Flash MX.

FCheckBox.setLabelPlacement

Метод, задающий местоположение текстовой надписи. Формат использования:

```
<Флажок>.setLabelPlacement("left"|"right");
```

Этот метод может принимать два строковых значения:

☐ "left" — надпись находится левее флажка;

☐ "right" — надпись находится правее флажка (значение по умолчанию).

Не возвращает значения.

Впервые появился во Flash MX.

FCheckBox.setSize

Метод, задающий ширину флажка в пикселах. Формат использования:

```
<Флажок>.setSize(<Ширина>);
```

Не возвращает значения.

Впервые появился во Flash MX.

FCheckBox.setStyleProperty

Метод, присваивающий новое значение свойству экземпляра объекта `FStyleFormat` для данного флажка. Формат использования:

```
<Флажок>.setStyleProperty(<Имя свойства>, <Значение свойства>);
```

Имя свойства задается в строковом виде. Значение свойства должен иметь тип, соответствующий данному свойству.

Чтобы присвоить выбранному свойству значение по умолчанию, задайте значение `undefined`.

Не возвращает значения.

```
chkSendMeAMail.setStyleProperty("shadow", 0xAAAAAA);
```

Впервые появился во Flash MX.

FCheckBox.setValue

Метод, задающий новое значение для флажка. Формат использования:

```
<Флажок>.setValue(<Логическое выражение>);
```

Если передано значение `true`, флажок станет включенным; это его значение по умолчанию. Если передать `false`, флажок отключится.

Не возвращает значения.

Впервые появился во Flash MX.

FComboBox

Объект, представляющий раскрывающийся список Flash. Реализован как образец-клип.

Для каждого раскрывающегося списка, помещенного на рабочий лист в среде Flash, создается отдельный экземпляр объекта `FComboBox`. Этот экземпляр получает имя, заданное для списка в редакторе свойств.

Для хранения и представления набора пунктов списка используется массив или особый объект-поставщик данных. Каждый пункт списка имеет название, отображающееся на экране; кроме того, он может иметь значение, которое при выборе данного пункта станет значением списка. Если пункт не имеет значения, возвращается его название.

Впервые появился во Flash MX.

FComboBox.addItem

Метод, добавляющий новый пункт в конец раскрывающегося списка. Формат использования:

```
<Раскрывающийся список>.addItem(<Название пункта>[, <Значение пункта>]);
```

Название пункта задается в строковом виде. Значение пункта, которое вы можете задать, будет возвращаться при выборе данного пункта как значение списка. Оно может быть любого типа, поддерживаемого Flash.

Не рекомендуется добавлять в список более 400 пунктов в одном кадре. Иначе работа списка может сильно замедлиться.

Не возвращает значения.

```
cboCities.addItem("Волжский", "VLZ");
```

Впервые появился во Flash MX.

FComboBox.addItemAt

Метод, добавляющий новый пункт в заданную позицию раскрывающегося списка. Формат использования:

```
<Раскрывающийся список>.addItemAt(<Номер добавляемого пункта>,  
❏<Название пункта>[, <Значение пункта>]);
```

Название пункта задается в строковом виде. Значение пункта, которое вы можете задать, будет возвращаться при выборе данного пункта как значение списка. Оно может быть любого типа, поддерживаемого Flash.

Не рекомендуется добавлять в список более 400 пунктов в одном кадре. Иначе работа списка может сильно замедлиться.

Не возвращает значения.

```
cboCities.addItemAt(10, "Волжский", "VLZ");
```

Впервые появился во Flash MX.

FComboBox.setEnabled

Метод, возвращающий `true`, если раскрывающийся список доступен, и `false` в противном случае. Не принимает параметров.

Впервые появился во Flash MX.

FComboBox.getItemAt

Метод, возвращающий пункт раскрывающегося списка с заданным номером. Формат использования:

```
<Раскрывающийся список>.getItemAt(<Номер пункта>);
```

Заданный пункт списка возвращается в виде экземпляра объекта `Object`, содержащего свойства `label` (название пункта) и `data` (значение пункта).

Впервые появился во Flash MX.

FComboBox.getLength

Метод, возвращающий количество пунктов раскрывающегося списка. Не принимает параметров.

Впервые появился во Flash MX.

FComboBox.getRowCount

Метод, возвращающий количество пунктов, видимых в раскрывающемся списке. Не принимает параметров.

Впервые появился во Flash MX.

FComboBox.getScrollPosition

Метод, возвращающий номер самого верхнего пункта, видимого в раскрывающемся списке. Не принимает параметров.

Впервые появился во Flash MX.

FComboBox.getSelectedIndex

Метод, возвращающий номер выбранного в раскрывающемся списке пункта. Не принимает параметров.

Если ни один пункт списка не выбран, возвращается `undefined`.

Впервые появился во Flash MX.

FComboBox.getSelectedItem

Метод, возвращающий выбранный в раскрывающемся списке пункт. Не принимает параметров.

Выбранный пункт списка возвращается в виде экземпляра объекта `Object`, содержащего свойства `label` (название пункта) и `data` (значение пункта).

Если ни один пункт списка не выбран, возвращается `undefined`.

Впервые появился во Flash MX.

FComboBox.getValue

Метод, возвращающий значение выбранного пункта раскрывающегося списка. Если значение не задано, возвращается название пункта. Если раскрывающийся список сделан редактируемым, т. е. содержит поле ввода, возвращается значение, введенное в этом поле. Не принимает параметров.

Впервые появился во Flash MX.

FComboBox.registerSkinElement

Метод, присваивающий одному из фрагментов "шкуры" раскрывающегося списка одно из свойств объекта `FStyleFormat`. После этого вы можете ме-

нять цвет или другие параметры фрагмента "шкуры", задавая нужные значения этого свойства. **Формат использования:**

```
<Раскрывающийся список>.registerSkinElement(<Имя фрагмента>, <Свойство>);
```

Каждый фрагмент "шкуры" есть образец-клип, хранящийся в особой папке библиотеки. Имя фрагмента — это имя экземпляра этого образца-клипа.

Не возвращает значения.

```
cboCities.registerSkinElement(shadow_mc, "shadow");
```

Впервые появился во Flash MX.

FCComboBox.removeAll

Метод, удаляющий все пункты раскрывающегося списка. Не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

FCComboBox.removeItemAt

Метод, удаляющий пункт раскрывающегося списка с заданным номером. **Формат использования:**

```
<Раскрывающийся список>.removeItemAt(<Номер пункта>);
```

Возвращает удаленный пункт списка в виде экземпляра объекта `Object`, содержащего свойства `label` (название пункта) и `data` (значение пункта).

Впервые появился во Flash MX.

FCComboBox.replaceItemAt

Метод, заменяющий пункт раскрывающегося списка с заданным номером на другой пункт. **Формат использования:**

```
<Раскрывающийся список>.replaceItemAt(<Номер заменяемого пункта>,
```

```
    <Название пункта>[, <Значение пункта>]);
```

Название пункта задается в строковом виде. Значение пункта, которое вы можете задать, будет возвращаться при выборе данного пункта как его значение списка. Оно может быть любого типа, поддерживаемого Flash.

Не возвращает значения.

Впервые появился во Flash MX.

FCComboBox.setChangeHandler

Метод, задающий функцию-обработчик выбора в раскрывающемся списке. **Формат использования:**

```
<Раскрывающийся список>.setChangeHandler("<Имя функции>"
```

```
    <[, <Расположение функции>]);
```

Расположение функции — это путь к объекту, где реализована нужная нам функция.

Функция-обработчик события должна принимать единственный параметр — ссылку на компонент, в котором произошло это событие.

Не возвращает значения.

Впервые появился во Flash MX.

FCComboBox.setDataProvider

Метод, задающий для раскрывающегося списка новый объект-поставщик данных. Формат использования:

```
<Раскрывающийся список>.setDataProvider(<Экземпляр объекта-поставщика  
данных>);
```

Объект-поставщик данных может быть:

- ☐ обычным массивом. В таком случае элементы этого массива станут названиями пунктов раскрывающегося списка;
- ☐ массивом, каждый элемент которого представляет собой объект, содержащий свойства `label` (название пункта) и `data` (значение пункта);
- ☐ экземпляром объекта `DataProviderClass`.

Не возвращает значения.

Впервые появился во Flash MX.

FCComboBox.setEditable

Метод, позволяющий сделать раскрывающийся список редактируемым. Формат использования:

```
<Раскрывающийся список>.setEditable(<Логическое выражение>);
```

Если передано значение `true`, раскрывающийся список становится редактируемым, и в нем появляется поле ввода, в котором пользователь может ввести любой текст. Когда пользователь вводит в него какую-либо строку, в списке автоматически выбирается пункт, чье название начинается с этой строки. Если передано значение `false`, список становится нередатируемым, и пользователь может только выбирать доступные пункты.

Не возвращает значения.

Впервые появился во Flash MX.

FCComboBox.setEnabled

Метод, разрешающий или запрещающий доступ к раскрывающемуся списку. Формат использования:

```
<Раскрывающийся список>.setEnabled(<Логическое выражение>);
```

Если передано значение `true`, доступ к списку разрешается. Если передать `false`, список станет недоступен.

Не возвращает значения.

Впервые появился во Flash MX.

FComboBox.setItemSymbol

Метод, задающий образец-клип, служащий для отображения пунктов списка. Формат использования:

```
<Раскрывающийся список>.setItemSymbol (<Имя образца>);
```

Имя образца задается в строковом виде. По умолчанию используется образец-клип `FComboBoxItem`, находящийся в папке `Flash UI Components/Core Assets - Developer Only/FUIComponent Class Tree/FUIComponent SubClasses/FSelectableItem SubClasses` библиотеки.

Не возвращает значения.

Впервые появился во Flash MX.

FComboBox.setRowCount

Метод, задающий количество пунктов, одновременно видимых в раскрывающемся списке. Формат использования:

```
<Раскрывающийся список>.setRowCount (<Количество пунктов>);
```

Не возвращает значения.

Впервые появился во Flash MX.

FComboBox.setSelectedIndex

Метод, задающий выбранный в раскрывающем списке пункт по его номеру. Формат использования:

```
<Раскрывающийся список>.setSelectedIndex (<Номер пункта>);
```

Не возвращает значения.

Впервые появился во Flash MX.

FComboBox.setSize

Метод, задающий ширину раскрывающегося списка в пикселах. Формат использования:

```
<Список>.setSize (<Ширина>);
```

Не возвращает значения.

Впервые появился во Flash MX.

FCombobox.setStyleProperty

Метод, присваивающий новое значение свойству экземпляра объекта `FStyleFormat` для данного раскрывающегося списка. Формат использования:

```
<Раскрывающийся список>.setStyleProperty(<Имя свойства>,  
❏ <Значение свойства>);
```

Имя свойства задается в строковом виде. Значение свойства должно иметь тип, соответствующий данному свойству.

Чтобы присвоить выбранному свойству значение по умолчанию, задайте значение `undefined`.

Не возвращает значения.

Впервые появился во Flash MX.

FCombobox.setValue

Метод, задающий новое значение для редактируемого раскрывающегося списка. Это значение появится в поле ввода этого списка. Формат использования:

```
<Раскрывающийся список>.setValue(<Значение>);
```

Значение задается в строковом виде.

Недоступен для нередатируемых списков.

Не возвращает значения.

Впервые появился во Flash MX.

FCombobox.sortItemsBy

Метод, сортирующий пункты раскрывающегося списка. Формат использования:

```
<Раскрывающийся список>.sortItemsBy("label"|"data", "ASC"|"DESC");
```

Первым параметром задается поле, по которому будут сортироваться пункты: `"label"` (название пункта) или `"data"` (значение пункта). Вторым параметром задается порядок сортировки: `"ASC"` (по возрастанию) или `"DESC"` (по убыванию).

Не возвращает значения.

Впервые появился во Flash MX.

FListBox

Объект, представляющий обычный список Flash. Реализован как образец-клип.

Для каждого списка, помещенного на рабочий лист в среде Flash, создается отдельный экземпляр объекта `FLListBox`. Этот экземпляр получает имя, заданное для списка в редакторе свойств.

Для хранения и представления набора пунктов списка используется массив или особый объект-поставщик данных. Каждый пункт списка имеет название, отображающееся на экране; кроме того, он может иметь значение, которое при выборе данного пункта станет значением списка. Если пункт не имеет значения, возвращается его название.

Впервые появился во Flash MX.

FLListBox.addItem

Метод, добавляющий новый пункт в конец списка. Формат использования:

```
<Список>.addItem(<Название пункта>[, <Значение пункта>]);
```

Название пункта задается в строковом виде. Значение пункта, которое вы можете задать, будет возвращаться при выборе данного пункта как значение списка. Оно может быть любого типа, поддерживаемого Flash.

Не рекомендуется добавлять в список более 400 пунктов в одном кадре. Иначе работа списка может сильно замедлиться.

Не возвращает значения.

```
lstCities.addItem("Волжский", "VLZ");
```

Впервые появился во Flash MX.

FLListBox.addItemAt

Метод, добавляющий новый пункт в заданную позицию списка. Формат использования:

```
<Список>.addItemAt(<Номер добавляемого пункта>, <Название пункта>
```

```
[, <Значение пункта>]);
```

Название пункта задается в строковом виде. Значение пункта, которое вы можете задать, будет возвращаться при выборе данного пункта как значение списка. Оно может быть любого типа, поддерживаемого Flash.

Не рекомендуется добавлять в список более 400 пунктов в одном кадре. Иначе работа списка может сильно замедлиться.

Не возвращает значения.

```
lstCities.addItemAt(10, "Волжский", "VLZ");
```

Впервые появился во Flash MX.

FListBox.setEnabled

Метод, возвращающий `true`, если список доступен, и `false` в противном случае. Не принимает параметров.

Впервые появился во Flash MX.

FListBox.getItemAt

Метод, возвращающий пункт списка с заданным номером. Формат использования:

```
<Список>.getItemAt(<Номер пункта>);
```

Заданный пункт списка возвращается в виде экземпляра объекта `Object`, содержащего свойства `label` (название пункта) и `data` (значение пункта).

Впервые появился во Flash MX.

FListBox.getLength

Метод, возвращающий количество пунктов списка. Не принимает параметров.

Впервые появился во Flash MX.

FListBox.getRowCount

Метод, возвращающий количество пунктов, видимых в списке. Не принимает параметров.

Впервые появился во Flash MX.

FListBox.getScrollPosition

Метод, возвращающий номер самого верхнего пункта, видимого в списке. Не принимает параметров.

Впервые появился во Flash MX.

FListBox.getSelectedIndex

Метод, возвращающий номер выбранного в списке пункта. Не принимает параметров.

Если не один пункт списка не выбран, возвращается `undefined`.

Впервые появился во Flash MX.

FListBox.getSelectedIndices

Метод, возвращающий номера всех выбранных в списке пунктов в виде массива. Не принимает параметров.

Если не один пункт списка не выбран, возвращается `undefined`.

Впервые появился во Flash MX.

FListBox.getSelectedItem

Метод, возвращающий выбранный в списке пункт. Не принимает параметров.

Выбранный пункт списка возвращается в виде экземпляра объекта `Object`, содержащего свойства `label` (название пункта) и `data` (значение пункта).

Если ни один пункт списка не выбран, возвращается `undefined`.

Впервые появился во Flash MX.

FListBox.getSelectedItems

Метод, возвращающий все выбранные в списке пункты в виде массива. Не принимает параметров.

Каждый элемент возвращенного массива представляет собой экземпляр объекта `Object`, содержащего свойства `label` (название пункта) и `data` (значение пункта).

Если ни один пункт списка не выбран, возвращается `undefined`.

Впервые появился во Flash MX.

FListBox.selectMultiple

Метод, возвращающий `true`, если пользователь может выбирать в списке сразу несколько пунктов. В противном случае возвращается значение `false`. Не принимает параметров.

Впервые появился во Flash MX.

FListBox.getValue

Метод, возвращающий значение выбранного пункта списка. Если значение не задано, возвращается название пункта. Не принимает параметров.

Впервые появился во Flash MX.

FListBox.registerSkinElement

Метод, присваивающий одному из фрагментов "шкур" списка одно из свойств объекта `FStyleFormat`. После этого вы можете менять цвет или

другие параметры фрагмента "шкур", задавая нужные значения этого свойства. Формат использования:

```
<Список>.registerSkinElement(<Имя фрагмента>, <Свойство>);
```

Каждый фрагмент "шкур" есть образец-клип, хранящийся в особой папке библиотеки. Имя фрагмента — это имя экземпляра этого образца-клипа.

Не возвращает значения.

```
lstCities.registerSkinElement(shadow_mc, "shadow");
```

Впервые появился во Flash MX.

FListBox.removeAll

Метод, удаляющий все пункты списка. Не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

FListBox.removeItemAt

Метод, удаляющий пункт списка с заданным номером. Формат использования:

```
<Список>.removeItemAt(<Номер пункта>);
```

Возвращает удаленный пункт списка в виде экземпляра объекта `Object`, содержащего свойства `label` (название пункта) и `data` (значение пункта).

Впервые появился во Flash MX.

FListBox.replaceItemAt

Метод, заменяющий пункт списка с заданным номером на другой пункт. Формат использования:

```
<Список>.replaceItemAt(<Номер заменяемого пункта>, <Название пункта>
```

```
[, <Значение пункта>]);
```

Название пункта задается в строковом виде. Значение пункта, которое вы можете задать, будет возвращаться при выборе данного пункта как его значение списка. Оно может быть любого типа, поддерживаемого Flash.

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.setAutoHideScrollBar

Метод, позволяющий сделать полосу прокрутки списка автоматически скрываемой в случае ненадобности. Формат использования:

```
<Список>.setAutoHideScrollBar(<Логическое выражение>);
```

Если задано значение `true`, полоса прокрутки списка будет исчезать, если количество пунктов списка не превышает его размера. Если задано значение `false`, то полоса прокрутки будет присутствовать всегда.

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.setChangeHandler

Метод, задающий функцию-обработчик выбора в списке. Формат использования:

```
<Список>.setChangeHandler("<Имя функции>"[, <Расположение функции>]);
```

Расположение функции — это путь к объекту, где реализована нужная нам функция.

Функция-обработчик события должна принимать единственный параметр — ссылку на компонент, в котором произошло это событие.

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.setDataProvider

Метод, задающий для списка новый объект-поставщик данных. Формат использования:

```
<Список>.setDataProvider(<Экземпляр объекта-поставщика данных>);
```

Объект-поставщик данных может быть:

- ☐ обычным массивом. В таком случае элементы этого массива станут названиями пунктов списка;
- ☐ массивом, каждый элемент которого представляет собой объект, содержащий свойства `label` (название пункта) и `data` (значение пункта);
- ☐ экземпляром объекта `DataProviderClass`.

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.setEnabled

Метод, разрешающий или запрещающий доступ к списку. Формат использования:

```
<Список>.setEnabled(<Логическое выражение>);
```

Если передано значение `true`, доступ к списку разрешается. Если передать `false`, список станет недоступен.

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.setItemSymbol

Метод, задающий образец-клип, служащий для отображения пунктов списка. Формат использования:

```
<Список>.setItemSymbol (<Имя образца>);
```

Имя образца задается в строковом виде. По умолчанию используется образец-клип `FListBoxItem`, находящийся в папке `Flash UI Components/Core Assets - Developer Only/FUIComponent Class Tree/FUIComponent SubClasses/FSelectableItem SubClasses` библиотеки.

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.setRowCount

Метод, задающий количество пунктов, одновременно видимых в списке. Формат использования:

```
<Список>.setRowCount (<Количество пунктов>);
```

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.setScrollPosition

Метод, заставляющий список прокрутиться так, чтобы пункт с заданным номером стал самым верхним из видимых в нем пунктов. Формат использования:

```
<Список>.setSelectedIndex (<Номер пункта>);
```

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.setSelectedIndex

Метод, задающий выбранный в списке пункт по его номеру. Формат использования:

```
<Список>.setSelectedIndex (<Номер пункта>);
```

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.setSelectedIndices

Метод, выбирающий в списке пункты, чьи номера были переданы ему в виде массива. Формат использования:

```
<Список>.setSelectedIndices(<Массив номеров пунктов>);
```

Пункт списка, чей номер находится в нулевом элементе массива, будет расположен в верхней строке списка.

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.setSelectMultiple

Метод, позволяющий разрешить пользователю выбирать одновременно несколько пунктов списка. Формат использования:

```
<Список>.setSelectMultiple(<Логическое выражение>);
```

Если передано значение `true`, пользователь может одновременно выбирать несколько пунктов списка. Если передано значение `false`, пользователь может выбрать только один пункт. Значение по умолчанию — `false`.

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.setSize

Метод, задающий ширину и высоту списка в пикселах. Формат использования:

```
<Список>.setSize(<Ширина>, <Высота>);
```

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.setStyleProperty

Метод, присваивающий новое значение свойству экземпляра объекта `FStyleFormat` для данного списка. Формат использования:

```
<Список>.setStyleProperty(<Имя свойства>, <Значение свойства>);
```

Имя свойства задается в строковом виде. Значение свойства должно иметь тип, соответствующий данному свойству.

Чтобы присвоить выбранному свойству значение по умолчанию, задайте значение `undefined`.

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.setWidth

Метод, задающий ширину списка в пикселах. Формат использования:

```
<Список>.setWidth(<Ширина>);
```

Не возвращает значения.

Впервые появился во Flash MX.

FListBox.sortItemsBy

Метод, сортирующий пункты списка. Формат использования:

```
<Список>.sortItemsBy("label"|"data", "ASC"|"DESC");
```

Первым параметром задается поле, по которому будут сортироваться пункты: "label" (название пункта) или "data" (значение пункта). Вторым параметром задается порядок сортировки: "ASC" (по возрастанию) или "DESC" (по убыванию).

Не возвращает значения.

Впервые появился во Flash MX.

_focusrect

Системная переменная, задающая, будет ли элемент управления иметь желтую рамку, если на ней находится фокус ввода. Если присвоено значение `true`, то кнопка имеет такую рамку, если `false` — не имеет.

В отличие от свойства `_focusrect` объекта `Button`, затрагивает все кнопки фильма.

Впервые появилась во Flash 4.

for

Действие, используемое для создания циклов со счетчиком. Формат использования:

```
for (<Выражение инициализации>; <Условие>; <Приращение счетчика>)
```

Тело цикла

Выражение инициализации присваивает счетчику начальное значение. Далее проверяется условие цикла, и, если его значение истинно, выполняется тело цикла. После этого выполняется выражение приращения счетчика, изменяющее значение счетчика, затем снова проверяется условие, и т. д. пока условие не станет ложным (`false`), т. е. пока счетчик не достигнет предельного значения.

Впервые появилось во Flash 5.

for... in

Действие, используемое для создания цикла просмотра. Формат использования:

```
for (<Переменная-ссылка на свойство> in <Экземпляр объекта>)  
    Тело цикла
```

В Переменную-ссылку на свойство каждый раз помещается значение очередного свойства экземпляра объекта. Вы можете использовать эту переменную для получения доступа к найденному свойству в теле цикла.

Впервые появилось во Flash 5.

FPushButton

Объект, представляющий кнопку Flash. Реализован как образец-клип.

Для каждой кнопки, помещенной на рабочий лист в среде Flash, создается отдельный экземпляр объекта *FPushButton*. Этот экземпляр получает имя, заданное для кнопки в редакторе свойств.

Впервые появился во Flash MX.

FPushButton.setEnabled

Метод, возвращающий *true*, если кнопка доступна, и *false* в противном случае. Не принимает параметров.

Впервые появился во Flash MX.

FPushButton.getLabel

Метод, возвращающий текстовую надпись на кнопке в строковом виде. Не принимает параметров.

Впервые появился во Flash MX.

FPushButton.registerSkinElement

Метод, присваивающий одному из фрагментов "шкур" кнопки одно из свойств объекта *FStyleFormat*. После этого вы можете менять цвет или другие параметры фрагмента "шкур", задавая нужные значения этого свойства. Формат использования:

```
<Кнопка>.registerSkinElement(<Имя фрагмента>, <Свойство>);
```

Каждый фрагмент "шкур" есть образец-клип, хранящийся в особой папке библиотеки. Имя фрагмента — это имя экземпляра этого образца-клипа.

Не возвращает значения.

```
btnOK.registerSkinElement(shadow_mc, "shadow");
```

Впервые появился во Flash MX.

FPushButton.setClickHandler

Метод, задающий функцию-обработчик отпускания нажатой кнопки. Формат использования:

```
<Кнопка>.setClickHandler("<Имя функции>"[, <Расположение функции>]);
```

Расположение функции — это путь к объекту, где реализована нужная нам функция.

Функция-обработчик события должна принимать единственный параметр — ссылку на компонент, в котором произошло это событие.

Не возвращает значения.

```
btnOK.setClickHandler("btnOK_click");  
btnOK.setClickHandler("btnOK_click", _root);
```

Впервые появился во Flash MX.

FPushButton.setEnabled

Метод, разрешающий или запрещающий доступ к кнопке. Формат использования:

```
<Кнопка>.setEnabled(<Логическое выражение>);
```

Если передано значение `true`, доступ к кнопке разрешается. Если передать `false`, кнопка станет недоступной.

Не возвращает значения.

Впервые появился во Flash MX.

FPushButton.setLabel

Метод, задающий текстовую надпись для кнопки. Формат использования:

```
<Кнопка>.setLabel(<Текстовая подпись>);
```

Текстовая надпись передается в строковом виде.

Не возвращает значения.

Впервые появился во Flash MX.

FPushButton.setSize

Метод, задающий ширину и высоту кнопки в пикселах. Формат использования:

```
<Кнопка>.setSize(<Ширина>, <Высота>);
```

Не возвращает значения.

Впервые появился во Flash MX.

FPushButton.setStyleProperty

Метод, присваивающий новое значение свойству экземпляра объекта `FStyleFormat` для данной кнопки. Формат использования:

```
<Кнопка>.setStyleProperty(<Имя свойства>, <Значение свойства>);
```

Имя свойства задается в строковом виде. Значение свойства должно иметь тип, соответствующий данному свойству.

Чтобы присвоить выбранному свойству значение по умолчанию, задайте значение `undefined`.

Не возвращает значения.

Впервые появился во Flash MX.

FRadioButton

Объект, представляющий переключатель Flash. Реализован как образец-клип.

Для каждого переключателя, помещенного на рабочий лист в среде Flash, создается отдельный экземпляр объекта `FRadioButton`. Этот экземпляр получает имя, заданное для переключателя в редакторе свойств.

Переключатели должны быть объединены в группы, поскольку смысла в одном переключателе нет. В каждой такой группе может быть включен только один переключатель.

Каждый переключатель, как и каждый пункт списка, должен иметь название. Кроме того, он может иметь значение, которое будет возвращено при выборе этого переключателя как значение группы. Если включенный переключатель не имеет значения, будет возвращено его название.

Впервые появился во Flash MX.

FRadioButton.getData

Метод, возвращающий значение, присвоенное переключателю, в строковом виде. Если значение не присвоено, возвращается пустая строка. Не принимает параметров.

Впервые появился во Flash MX.

FRadioButton.setEnabled

Метод, возвращающий `true`, если переключатель доступен, и `false` в противном случае. Не принимает параметров.

Впервые появился во Flash MX.

FRadioButton.getLabel

Метод, возвращающий текстовую надпись переключателя в строковом виде. Не принимает параметров.

Впервые появился во Flash MX.

FRadioButton.getState

Метод, возвращающий `true`, если переключатель включен, и `false` в противном случае. Не принимает параметров.

Впервые появился во Flash MX.

FRadioButton.getValue

Метод, возвращающий значение включенного переключателя в группе. Если не включен ни один переключатель, возвращается `undefined`. Не принимает параметров.

Впервые появился во Flash MX.

FRadioButton.registerSkinElement

Метод, присваивающий одному из фрагментов "шкуры" переключателя одно из свойств объекта `FStyleFormat`. После этого вы можете менять цвет или другие параметры фрагмента "шкуры", задавая нужные значения этого свойства. Формат использования:

```
<Переключатель>.registerSkinElement(<Имя фрагмента>, <Свойство>);
```

Каждый фрагмент "шкуры" есть образец-клип, хранящийся в особой папке библиотеки. Имя фрагмента — это имя экземпляра этого образца-клипа.

Не возвращает значения.

```
optMail.registerSkinElement(shadow_mc, "shadow");
```

Впервые появился во Flash MX.

FRadioButton.setChangeHandler

Метод, задающий функцию-обработчик изменения состояния переключателя. Формат использования:

```
<Переключатель>.setChangeHandler("<Имя функции>"
```

```
⌘ [, <Расположение функции>]);
```

Расположение функции — это путь к объекту, где реализована нужная нам функция.

Функция-обработчик события должна принимать единственный параметр — ссылку на компонент, в котором произошло это событие.

Не возвращает значения.

```
optMale.setChangeHandler("optGender_click");  
optFemale.setChangeHandler("optGender_click", _root);
```

Впервые появился во Flash MX.

FRadioButton.setData

Метод, задающий значение переключателя. Формат использования:

```
<Переключатель>.setData(<Значение>);
```

Значение задается в строковом виде. Чтобы отменить значение, передайте этому методу в качестве параметра пустую строку.

Не возвращает значения.

Впервые появился во Flash MX.

FRadioButton.setEnabled

Метод, разрешающий или запрещающий доступ к переключателю. Формат использования:

```
<Переключатель>.setEnabled(<Логическое выражение>);
```

Если передано значение `true`, доступ к переключателю разрешается. Если передать `false`, переключатель станет недоступен.

Не возвращает значения.

Впервые появился во Flash MX.

FRadioButton.setGroupName

Метод, задающий группу, в которую входит переключатель. Очевидно, что для переключателей, входящих в одну группу, это значение должно быть одинаковым. Формат использования:

```
<Переключатель>.setGroupName(<Имя группы>);
```

Имя группы задается в строковом виде.

Не возвращает значения.

Впервые появился во Flash MX.

FRadioButton.setLabel

Метод, задающий текстовую надпись для переключателя. Формат использования:

```
<Переключатель>.setLabel(<Текстовая подпись>);
```

Текстовая надпись передается в строковом виде.

Не возвращает значения.

Впервые появился во Flash MX.

FRadioButton.setLabelPlacement

Метод, задающий местоположение текстовой надписи. Формат использования:

```
<Переключатель>.setLabelPlacement("left"|"right");
```

Этот метод может принимать два строковых значения:

- "left" — надпись находится левее переключателя;
- "right" — надпись находится правее переключателя (значение по умолчанию).

Не возвращает значения.

Впервые появился во Flash MX.

FRadioButton.setSize

Метод, задающий ширину переключателя в пикселах. Формат использования:

```
<Переключатель>.setSize(<Ширина>);
```

Не возвращает значения.

Впервые появился во Flash MX.

FRadioButton.setState

Метод, задающий новое состояние переключателя. Формат использования:

```
<Переключатель>.setState(<Логическое выражение>);
```

Если передано значение `true`, переключатель станет включенным; при этом ранее включенный в этой группе переключатель отключится. Если передать `false`, переключатель отключится; это его значение по умолчанию.

Не возвращает значения.

Впервые появился во Flash MX.

FRadioButton.setStyleProperty

Метод, присваивающий новое значение свойству экземпляра объекта `FStyleFormat` для данного переключателя. Формат использования:

```
<Переключатель>.setStyleProperty(<Имя свойства>, <Значение свойства>);
```

Имя свойства задается в строковом виде. Значение свойства должно иметь тип, соответствующий данному свойству.

Чтобы присвоить выбранному свойству значение по умолчанию, задайте значение `undefined`.

Не возвращает значения.

```
optMale.setStyleProperty("shadow", 0xAFFFFFFF);
```

Впервые появился во Flash MX.

FRadioButton.setValue

Метод, задающий включенный переключатель в группе. Формат использования:

```
<Переключатель>.setValue(<Значение или название>);
```

Будет включен переключатель со Значением или названием, переданным в качестве параметра. Значение или название передаются в строковом виде.

Не возвращает значения.

Впервые появился во Flash MX.

FScrollBar

Объект, представляющий полосу прокрутки Flash. Реализован как образец-клип.

Для каждой полосы прокрутки, помещенной на рабочий лист в среде Flash, создается отдельный экземпляр объекта `FScrollBar`. Этот экземпляр получает имя, заданное для полосы прокрутки в редакторе свойств.

Полоса прокрутки может быть использована как отдельный компонент и привязана к полю ввода или динамическому текстовому блоку.

Впервые появился во Flash MX.

FScrollBar.setEnabled

Метод, возвращающий `true`, если полоса прокрутки доступна, и `false` в противном случае. Не принимает параметров.

Впервые появился во Flash MX.

FScrollBar.getScrollPosition

Метод, возвращающий позицию указателя полосы прокрутки в целочисленном виде. Не принимает параметров.

Впервые появился во Flash MX.

FScrollBar.registerSkinElement

Метод, присваивающий одному из фрагментов "шкуры" полосы прокрутки одно из свойств объекта `FStyleFormat`. После этого вы можете менять цвет или другие параметры фрагмента "шкуры", задавая нужные значения этого свойства. Формат использования:

```
<Полоса прокрутки>.registerSkinElement(<Имя фрагмента>, <Свойство>);
```

Каждый фрагмент "шкуры" есть образец-клип, хранящийся в особой папке библиотеки. Имя фрагмента — это имя экземпляра этого образца-клипа.

Не возвращает значения.

```
scbVolume.registerSkinElement(shadow_mc, "shadow");
```

Впервые появился во Flash MX.

FScrollBar.setChangeHandler

Метод, задающий функцию-обработчик перемещения указателя полосы прокрутки. Формат использования:

```
<Полоса прокрутки>.setChangeHandler("<Имя функции>")
```

```
⌘ [, <Расположение функции>]);
```

Расположение функции — это путь к объекту, где реализована нужная нам функция.

Функция-обработчик события должна принимать единственный параметр — ссылку на компонент, в котором произошло это событие.

Не возвращает значения.

```
scbVolume.setChangeHandler("scbVolume_move");
```

```
scbVolume.setChangeHandler("scbVolume_move", _root);
```

Впервые появился во Flash MX.

FScrollBar.setEnabled

Метод, разрешающий или запрещающий доступ к полосе прокрутки. Формат использования:

```
<Полоса прокрутки>.setEnabled(<Логическое выражение>);
```

Если передано значение `true`, доступ к полосе прокрутки разрешается. Если передать `false`, полоса прокрутки станет недоступна.

Не возвращает значения.

Впервые появился во Flash MX.

FScrollBar.setHorizontal

Метод, позволяющий сделать полосу прокрутки горизонтальной или вертикальной. Формат использования:

```
<Полоса прокрутки>.setHorizontal(<Логическое выражение>);
```

Если задано значение `true`, полоса прокрутки станет горизонтальной. Если же задано значение `false`, вы получите вертикальную полосу прокрутки. Значение по умолчанию — `false`.

Не возвращает значения.

Впервые появился во Flash MX.

FScrollBar.setLargeScroll

Метод, задающий шаг прокрутки при "листании", когда пользователь щелкает мышью по полосе прокрутки. Формат использования:

```
<Полоса прокрутки>.setLargeScroll(<Шаг прокрутки>);
```

Шаг прокрутки задается в целочисленном виде. Значение по умолчанию равно первому параметру метода `setScrollProperties`.

Не возвращает значения.

Впервые появился во Flash MX.

FScrollBar.setScrollPosition

Метод, задающий позицию указателя полосы прокрутки. Формат использования:

```
<Полоса прокрутки>.setScrollPosition(<Позиция указателя>);
```

Позиция указателя задается в целочисленном виде.

Не возвращает значения.

Впервые появился во Flash MX.

FScrollBar.setScrollProperties

Метод, задающий различные параметры полосы прокрутки. Формат использования:

```
<Полоса прокрутки>.setScrollProperties(<Количество "делений">,  
<Минимальное значение>, <Максимальное значение>);
```

Первый параметр задает количество воображаемых "делений", по которым будет "прыгать" указатель полосы прокрутки. Второй и третий параметры задают значения, присваиваемые минимальному и максимальному "делению". Все эти параметры задаются целыми числами.

Не возвращает значения.

Впервые появился во Flash MX.

FScrollBar.setScrollTarget

Метод, привязывающий полосу прокрутки к полю ввода или динамическому текстовому блоку. Формат использования:

```
<Полоса прокрутки>.setScrollTarget(<Ссылка>);
```

В качестве параметра передается ссылка на поле ввода или динамический текстовый блок. Чтобы "отвязать" полосу прокрутки, передайте этому методу значение `undefined`.

Учтите, что поле ввода или динамический текстовый блок должны располагаться в том же клипе и на том же уровне, что и полоса прокрутки.

Не возвращает значения.

```
optScroller.setScrollTarget(txtOutput);
```

Впервые появился во Flash MX.

FScrollBar.setSize

Метод, задающий длину полосы прокрутки в пикселах. Формат использования:

```
<Полоса прокрутки>.setSize(<Длина>);
```

Не возвращает значения.

Впервые появился во Flash MX.

FScrollBar.setSmallScroll

Метод, задающий шаг прокрутки при щелчке мышью по стрелкам полосы прокрутки. Формат использования:

```
<Полоса прокрутки>.setSmallScroll(<Шаг прокрутки>);
```

Шаг прокрутки задается в целочисленном виде. Значение по умолчанию — 1.

Не возвращает значения.

Впервые появился во Flash MX.

FScrollBar.setStyleProperty

Метод, присваивающий новое значение свойству экземпляра объекта `FStyleFormat` для данной полосы прокрутки. Формат использования:

```
<Полоса прокрутки>.setStyleProperty(<Имя свойства>, <Значение свойства>);
```

Имя свойства задается в строковом виде. Значение свойства должно иметь тип, соответствующий данному свойству.

Чтобы присвоить выбранному свойству значение по умолчанию, задайте значение `undefined`.

Не возвращает значения.

```
scbScroller.setStyleProperty("arrow", 0x000000);
```

Впервые появился во Flash MX.

FScrollPane

Объект, представляющий панель с прокруткой Flash. Реализован как образец-клип.

Для каждой панели с прокруткой, помещенной на рабочий лист в среде Flash, создается отдельный экземпляр объекта `FScrollPane`. Этот экземпляр получает имя, заданное для панели с прокруткой в редакторе свойств.

Имейте в виду, что сама панель с прокруткой не обеспечивает прокрутку своего содержимого. Для этого вам придется использовать полосы прокрутки `FScrollBar`.

Впервые появился во Flash MX.

FScrollPane.getHeight

Метод, возвращающий высоту панели с прокруткой в пикселах. Не принимает параметров.

Впервые появился во Flash MX.

FScrollPane.getWidth

Метод, возвращающий ширину панели с прокруткой в пикселах. Не принимает параметров.

Впервые появился во Flash MX.

FScrollPane.getScrollContent

Метод, возвращающий ссылку на клип, помещенный в панель с прокруткой. Не принимает параметров.

Впервые появился во Flash MX.

FScrollPane.getScrollPosition

Метод, возвращающий позиции указателей обеих полос прокрутки. Не принимает параметров.

Метод возвращает экземпляр объекта `Object`, содержащий свойства `x` (позиция указателя горизонтальной полосы прокрутки) и `y` (позиция указателя вертикальной полосы прокрутки).

Впервые появился во Flash MX.

FScrollPane.loadScrollContent

Метод, загружающий клип или растровое изображение в панель с прокруткой. Формат использования:

```
<Панель с прокруткой>.loadScrollContent(<Интернет-адрес>
```

```
[, <Функция-обработчик окончания загрузки>[, <Расположение функции>]]);
```

Первым параметром передается интернет-адрес файла Shockwave/Flash или JPEG, содержащего нужный клип или растровое изображение, в строковом виде. Второй параметр задает также в строковом виде имя функции, вызываемой после окончания загрузки клипа или изображения. Третьим параметром передается путь к объекту, где реализована функция-обработчик.

Функция-обработчик окончания загрузки должна принимать единственный параметр — ссылку на компонент, в котором произошло это событие.

Не возвращает значения.

```
panView.loadScrollContent("banner.jpg", "panView_loaded", _root);
```

Впервые появился во Flash MX.

FScrollPane.refreshPane

Метод, обновляющий содержимое панели с прокруткой. Должен быть использован после изменения размеров панели или изменения ее содержимого. Не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

FScrollPane.registerSkinElement

Метод, присваивающий одному из фрагментов "шкур" панели с прокруткой одно из свойств объекта `FStyleFormat`. После этого вы можете менять цвет или другие параметры фрагмента "шкур", задавая нужные значения этого свойства. Формат использования:

```
<Панель с прокруткой>.registerSkinElement(<Имя фрагмента>, <Свойство>);
```

Каждый фрагмент "шкур" есть образец-клип, хранящийся в особой папке библиотеки. Имя фрагмента — это имя экземпляра этого образца-клипа.

Не возвращает значения.

```
panView.registerSkinElement(shadow_mc, "shadow");
```

Впервые появился во Flash MX.

FScrollPane.setDragContent

Метод, разрешающий или запрещающий пользователю перетаскивать мышью содержимое панели с прокруткой. Формат использования:

```
<Панель с прокруткой>.setDragContent(<Логическое выражение>);
```

Если передано значение `true`, то пользователь может перетаскивать содержимое внутри панели с прокруткой. Если же передано значение `false`, содержимое не будет перетаскиваться мышью.

Не возвращает значения.

Впервые появился во Flash MX.

FScrollPane.setHScroll

Метод, позволяющий сделать горизонтальную полосу прокрутки видимой или невидимой. Формат использования:

```
<Панель с прокруткой>.setHScroll(true|false|"auto");
```

Если задано значение `true`, горизонтальная полоса прокрутки всегда будет видна. Если задано значение `false`, то полоса прокрутки не будет видна никогда, даже в тех случаях, когда она необходима. Если задать строковое значение `"auto"`, то полоса прокрутки появится только в случае необходимости. Значение по умолчанию — `"auto"`.

Не возвращает значения.

Впервые появился во Flash MX.

FScrollPane.setScrollContent

Метод, задающий содержимое панели с прокруткой. Формат использования:

```
<Панель с прокруткой>.setScrollContent(<Имя образца-клипа>|
```

```
⌚ <Путь к клипу>);
```

Вы можете передать этому методу имя образца-клипа, находящегося в библиотеке; при этом на рабочем листе не обязательно должны присутствовать экземпляры данного образца. Имейте в виду, что образец-клип, который вы хотите поместить в панель, должен быть сценарным. Также вы можете передать этому методу путь уже существующего на рабочем листе экземпляра-клипа; в этом случае он будет показан с учетом всех примененных к нему преобразований.

Не возвращает значения.

Впервые появился во Flash MX.

FScrollPane.setScrollPosition

Метод, задающий позиции указателей обеих полос прокрутки. Формат использования:

```
<Панель с прокруткой>.setScrollPosition(<X>, <Y>);
```

Первым параметром этого метода передается позиция указателя горизонтальной полосы прокрутки, а вторым — позиция указателя вертикальной полосы прокрутки.

Не возвращает значения.

Впервые появился во Flash MX.

FScrollPane.setSize

Метод, задающий ширину и высоту панели с прокруткой в пикселах. Формат использования:

```
<Панель с прокруткой>.setSize(<Ширина>, <Высота>);
```

Не возвращает значения.

Впервые появился во Flash MX.

FScrollPane.setStyleProperty

Метод, присваивающий новое значение свойству экземпляра объекта `FStyleFormat` для данной панели с прокруткой. Формат использования:

```
<Панель с прокруткой>.setStyleProperty(<Имя свойства>,
```

```
    <Значение свойства>);
```

Имя свойства задается в строковом виде. Значение свойства должно иметь тип, соответствующий данному свойству.

Чтобы присвоить выбранному свойству значение по умолчанию, задайте значение `undefined`.

Не возвращает значения.

Впервые появился во Flash MX.

FScrollPane.setVScroll

Метод, позволяющий сделать вертикальную полосу прокрутки видимой или невидимой. Формат использования:

```
<Панель с прокруткой>.setVScroll(true|false|"auto");
```

Если задано значение `true`, вертикальная полоса прокрутки всегда будет видна. Если задано значение `false`, то полоса прокрутки не будет видна никогда, даже в тех случаях, когда она необходима. Если задать строковое зна-

чение "auto", то полоса прокрутки появится только в случае необходимости. Значение по умолчанию — "auto".

Не возвращает значения.

Впервые появился во Flash MX.

FStyleFormat

Объект, предоставляющий доступ к различным параметрам компонентов, так называемый стилевой объект.

Для создания экземпляра этого объекта используется конструктор:

```
<Переменная> = new FStyleFormat();
```

Кроме того, Flash автоматически создает глобальный экземпляр этого объекта под именем `globalStyleFormat` и по одному экземпляру для каждого имеющегося на рабочем листе компонента. Для изменения свойств таких внутренних объектов служит метод `setStyleProperty`.

Впервые появился во Flash MX.

FStyleFormat.addListener

Метод, привязывающий к созданному пользователем стилевому объекту имеющиеся на рабочем листе компоненты. Формат использования:

```
<Стилевой объект>.addListener(<Список компонентов, разделенных  
⌘ запятыми>);
```

Не возвращает значения.

```
darkStyle.addListener(cboGender, chkSendMeAMail, btnOK, btnCancel);
```

Впервые появился во Flash MX.

FStyleFormat.applyChanges

Метод, обновляющий внешний вид привязанных к стилевому объекту компонентов после изменения свойств этого стилевого объекта. Формат использования:

```
<Стилевой объект>.applyChanges ([<Список свойств, разделенных  
⌘ запятыми>]);
```

Если задать `Список свойств`, то будут обновлены только параметры, связанные с данными свойствами. Свойства задаются в строковом виде.

Не возвращает значения.

```
darkStyle.applyChanges ("arrow", "background");
```

Впервые появился во Flash MX.

FStyleFormat.arrow

Свойство, задающее цвет стрелки обычного и раскрывающегося списка и полос прокрутки. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.background

Свойство, задающее цвет фона компонента. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.backgroundDisabled

Свойство, задающее цвет фона компонента, доступ к которому запрещен. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.check

Свойство, задающее цвет галочки у флажка. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.darkShadow

Свойство, задающее цвет внутренней рамки или "темной тени" компонента. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.face

Свойство, задающее "главный" цвет компонента, например, цвет "тела" кнопки или полосы прокрутки. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.foregroundDisabled

Свойство, задающее "главный" цвет компонента, доступ к которому запрещен. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.highlight

Свойство, задающее цвет внутренней рамки или "темной тени" активного компонента. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.highlight3D

Свойство, задающее цвет внешней рамки или "светлой тени" активного компонента. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.radioDot

Свойство, задающее цвет точки у переключателя. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.removeListener

Метод, удаляющий привязку компонента к стилевому объекту. Формат использования:

```
<Стилевой объект>.removeListener(<Компонент>);
```

При удалении привязки компонента к пользовательскому стилевому объекту он автоматически привязывается к глобальному стилевому объекту `globalStyleFormat`. Если же удалить его привязку и к глобальному объекту, компонент будет показан в своем изначальном виде.

Не возвращает значения.

Впервые появился во Flash MX.

FStyleFormat.scrollTrack

Свойство, задающее цвет указателя полосы прокрутки. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.selection

Свойство, задающее цвет выделения пункта списка. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.selectionDisabled

Свойство, задающее цвет выделения пункта списка, доступ к которому запрещен. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.selectionUnfocused

Свойство, задающее цвет выделения пункта списка, не имеющего фокуса ввода. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.shadow

Свойство, задающее цвет внешней рамки или "светлой тени" компонента. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.textAlign

Свойство, задающее выравнивание текста надписи. Может принимать три строковых значения:

- ☐ "left" — выравнивание по левому краю (значение по умолчанию);
- ☐ "center" — выравнивание по центру;
- ☐ "right" — выравнивание по правому краю.

Впервые появилось во Flash MX.

FStyleFormat.textBold

Свойство, задающее "жирность" текста надписи. Если равно true, надпись выводится полужирным шрифтом, если false — обычным.

Впервые появилось во Flash MX.

FStyleFormat.textColor

Свойство, задающее цвет текста компонента. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.textDisabled

Свойство, задающее цвет текста компонента, доступ к которому запрещен. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.textFont

Свойство, задающее шрифт текста надписи в строковом виде.

```
darkStyle.textFont = "Arial Black";
```

Впервые появилось во Flash MX.

FStyleFormat.textIndent

Свойство, задающее отступ красной строки текста в пикселах.

Впервые появилось во Flash MX.

FStyleFormat.textItalic

Свойство, задающее курсивное начертание текста надписи. Если равно `true`, надпись выводится курсивом, если `false` — обычным шрифтом.

Впервые появилось во Flash MX.

FStyleFormat.textLeftMargin

Свойство, задающее отступ от левой границы текста в пикселах.

Впервые появилось во Flash MX.

FStyleFormat.textRightMargin

Свойство, задающее отступ от правой границы текста в пикселах.

Впервые появилось во Flash MX.

FStyleFormat.textSelected

Свойство, задающее цвет текста выделенной надписи. Должно использоваться вместе со свойством `selection`. Значение цвета должно быть в формате 0xRRGGBB.

Впервые появилось во Flash MX.

FStyleFormat.textSize

Свойство, задающее размер шрифта надписи в пунктах. Значение по умолчанию — 12.

Впервые появилось во Flash MX.

FStyleFormat.textUnderline

Свойство, задающее подчеркивание текста надписи. Если равно `true`, надпись подчеркивается, если `false` — нет.

Впервые появилось во Flash MX.

Function

Объект, позволяющий манипулировать функцией как объектами.

Для создания экземпляра этого объекта используется конструктор:

```
{Имя функции} = new Function({Список аргументов, заключенных в кавычки и  
разделенных запятыми} {Тело функции, заключенное в кавычки});
```

Пример создания функции как экземпляра объекта `Function`:

```
someFunc = new Function("a", "b", "(a + b) / 2");
```

Кроме того, функцию можно создать по старинке:

```
function someFunc(a, b) {  
    return (a + b) / 2;  
}
```

Впервые появился во Flash MX.

Function.apply

Метод, служащий для вызова функции или метода. Формат использования:

```
<Функция>.apply(<Объект>, <Массив параметров>);
```

Первым параметром этого метода передается ссылка на объект, к которому можно будет обратиться изнутри вызываемой функции, используя ключевое слово `this`. Вторым параметром передается массив параметров вызываемой функции.

Метод возвращает значение, возвращенное вызываемой функцией.

```
btnOK.setSize(100, 100);  
btnOK.setSize.apply(btnOK, [100, 100]);
```

Впервые появился во Flash MX.

Function.call

Метод, служащий для вызова функции или метода. Аналогичен методу `apply`. Формат использования:

```
<Функция>.call(<Объект>[, <Список параметров, разделенных запятыми>]);
```

Первым параметром этого метода передается ссылка на объект, к которому можно будет обратиться изнутри вызываемой функции, используя ключевое слово `this`. Остальными параметрами передаются значения параметров вызываемой функции.

Метод возвращает значение, возвращенное вызываемой функцией.

```
btnOK.setSize(100, 100);  
btnOK.setSize.call(btnOK, 100, 100);
```

Впервые появился во Flash MX.

Function.prototype

Свойство, возвращающее ссылку на объект-прототип создаваемого объекта. Может использоваться только в конструкторах.

Впервые появилось во Flash MX.

fscommand

Действие, позволяющее внешним сценариям на языке JavaScript управлять проигрывателем Flash, внедренным в Web-страницу. Также позволяет сценариям ActionScript вызывать внешний код на JavaScript.

Подробное описание приведено в *главе 24*.

Впервые появилось во Flash 3.

function

Действие, служащее для создания функции. Формат использования:

```
[<Имя функции> = ]function [Имя функции]([Список формальных параметров,  
❖ разделенных запятыми]) {  
    Тело функции  
}
```

Формальные параметры, перечисленные в скобках, используются внутри тела функции. При вызове созданной функции вам будет нужно подставить на их место реальные значения параметров.

```
function x2(x) { return x * x; };  
x2 = function (x) { return x * x; };  
a = x2(2);
```

Впервые появилось во Flash 5.

ge

Оператор сравнения "не меньше". Формат использования:

<Аргумент 1> ge <Аргумент 2>

Возвращает true, если строковое представление Аргумента 1 больше или равно строковому представлению Аргумента 2, и false в противном случае.

Впервые появился во Flash 4. Во Flash 5 объявлен устаревшим и нерекомендованным к использованию; вместо него рекомендуется применять оператор >=.

getProperty

Функция, возвращающая значение свойства объекта. Формат использования:

```
getProperty(<Экземпляр объекта>, <Свойство объекта>);
```

Два следующих выражения эквивалентны:

```
a = getProperty(_root.car.wheel, _x);
```

```
a = _root.car.wheel._x;
```

Впервые появилась во Flash 4.

getTimer

Функция, возвращающая количество миллисекунд, прошедших с начала проигрывания фильма. Не принимает параметров.

Впервые появилась во Flash 4.

getURL

Действие, загружающее Web-страницу и выводящее ее в окне Web-обозревателя. Может также применяться для передачи данных серверному приложению и приема от него результатов. Формат использования:

```
getURL(<Интернет-адрес>[, <Цель>[, <Метод передачи данных>]]);
```

Первым параметром передается интернет-адрес нужной Web-страницы или серверного приложения. Вторым параметром передается имя фрейма, в который будет загружена страница, или одно из предопределенных значений:

- ☐ `_self` — страница загружается в текущий фрейм текущего окна (значение по умолчанию);
- ☐ `_blank` — страница загружается в новом окне;
- ☐ `_parent` — страница загружается во фрейм, являющийся внешним для текущего фрейма текущего окна, или в само это окно;
- ☐ `_top` — страница загружается в текущее окно, заполняя его целиком.

Третий параметр задает метод передачи данных серверному приложению. Он может принимать два значения:

- "GET" — для передачи данных используется метод GET;
- "POST" — для передачи данных используется метод POST.

Если третий параметр пропущен, данные не передаются.

Впервые появилось во Flash 2. Третий параметр стал поддерживаться Flash 4.

getVersion

Функция, возвращающая строку, содержащую сведения о клиентской платформе и версии проигрывателя Flash. Не принимает параметров.

Впервые появилась во Flash 5.

_global

Модификатор, используемый для создания глобальных переменных, функций и объектов.

Впервые появился во Flash MX.

globalStyleFormat

Объект, задающий глобальные стилевые параметры для всех элементов управления, присутствующих на рабочем листе. Создается самим Flash как экземпляр объекта `FStyleFormat`.

```
globalStyleFormat.textColor = 0xFF0000;
```

Впервые появился во Flash MX.

gotoAndPlay

Действие, перемещающее указатель на заданный кадр заданной сцены и продолжающее проигрывание фильма с него. Формат использования:

```
gotoAndPlay([<Сцена>, ]<Кадр>);
```

Первым параметром может быть задано имя сцены в строковом виде. Если оно не задано, Flash считает, что имелась в виду текущая сцена. Вторым параметром задается номер или имя нужного кадра.

Впервые появилось во Flash 2.

gotoAndStop

Действие, перемещающее указатель на заданный кадр заданной сцены и останавливающее проигрывание фильма. Формат использования:

```
gotoAndStop([<Сцена>, ]<Кадр>);
```

Первым параметром может быть задано имя сцены в строковом виде. Если оно не задано, Flash считает, что имелась в виду текущая сцена. Вторым параметром задается номер или имя нужного кадра.

Впервые появилось во Flash 2.

gt

Оператор сравнения "больше". Формат использования:

```
<Аргумент 1> gt <Аргумент 2>
```

Возвращает `true`, если строковое представление Аргумента 1 больше строкового представления Аргумента 2, и `false` в противном случае.

Впервые появился во Flash 4. Во Flash 5 объявлен устаревшим и нерекомендованным к использованию; вместо него рекомендуется применять оператор `>`.

_highquality

Свойство, задающее качество сглаживания растровой графики в фильме. Может принимать три числовых значения:

- ☐ 2 — растровая графика сглаживается всегда;
- ☐ 1 — растровая графика сглаживается только тогда, когда фильм не содержит анимации;
- ☐ 0 — растровая графика никогда не сглаживается.

Используйте это свойство, чтобы уменьшить количество системных ресурсов, потребляемых проигрывателем Flash. Это может понадобиться на медленных компьютерах.

Впервые появилось во Flash 4.

if

Действие, используемое для создания ветвления. Формат использования:

```
if (<Условие>) {  
    Код 1  
} else {  
    Код 2  
}
```

Если `Условие` верно, выполняется Код 1, в противном случае — Код 2.

```
if (a<0) {  
    s = "Отрицательное число";  
} else {
```

```
s = "Положительное число или ноль.";
}
```

Впервые появилось во Flash 4.

ifFrameLoaded

Действие, проверяющее, загружен ли заданный кадр фильма, и выполняющее или не выполняющее в зависимости от этого некий код. Формат использования:

```
ifFrameLoaded([<Сцена> ,]<Кадр>) {
    Код
}
```

Первым параметром может быть задано имя сцены в строковом виде. Если оно не задано, Flash считает, что имелась в виду текущая сцена. Вторым параметром задается номер или имя нужного кадра. Если этот кадр загружен, выполняется Код.

Впервые появилось во Flash 3. Во Flash 5 объявлено устаревшим и нерекондованным к использованию; вместо него рекомендуется применять свойство `_framesLoaded` объекта `movieClip`.

#include

Действие, включающее в фильм Flash содержимое внешнего файла. Формат использования:

```
#include "<Имя файла>";
```

Внешний файл, содержащий код *ActionScript*, должен содержать расширение `as`. Его содержимое включается в фильм Flash при его публикации и экспорте, а также участвует в проверке синтаксиса.

#initclip

Действие, задающее блок кода инициализации. Формат использования:

```
#initclip [<Порядок>]
    Код
#endinitclip
```

Код инициализации компонента выполняется перед всеми сценариями, присвоенными первому кадру фильма, даже если ни одного экземпляра компонента в этом кадре нет. Код инициализации всегда выполняется только один раз. Если вы не выделите код инициализации с помощью действий `#initclip...#endinitclip`, он будет выполнен в том кадре, где впервые появляется экземпляр этого компонента, и будет выполняться далее для каждого экземпляра.

Если в фильме используется несколько компонентов, содержащих код инициализации, вы можете управлять порядком, в котором он выполняется. Для этого задайте `Порядок` в виде целого числа; чем больше это число, тем позже будет выполнен соответствующий код.

Впервые появилось во Flash MX.

instanceof

Оператор, проверяющий, содержит ли данная переменная экземпляр заданного объекта. Формат использования:

```
<Переменная> instanceof <Объект>
```

Если `Переменная` содержит экземпляр `Объекта`, возвращается `true`, в противном случае возвращается `false`.

Впервые появился во Flash MX.

int

Функция, округляющая число до целых. Формат использования:

```
int(<Число>);
```

Впервые появилась во Flash 4. Во Flash 5 объявлена устаревшей и не рекомендованной к использованию; вместо нее рекомендуется применять метод `round` объекта `Math`.

isFinite

Функция, позволяющая узнать, является ли аргумент конечным числом. Формат использования:

```
isFinite(<Числовое выражение>);
```

Если результатом `Числового выражения` является конечное число, возвращается `true`. Если же в результате его вычисления получается бесконечность, возвращается `false`.

Впервые появилась во Flash 5.

isNaN

Функция, позволяющая узнать, содержит ли числовое выражение ошибки. Формат использования:

```
isNaN(<Числовое выражение>);
```

Если `Числовое выражение` содержит ошибки или, как говорят, выдает в результате значение `NaN` (Not a Number — не число), возвращается `true`. Если же выражение не содержит ошибок, возвращается `false`.

Впервые появилась во Flash 5.

Key

Объект, предоставляющий доступ к клавиатурному вводу.

Единственный экземпляр объекта `Key` под тем же именем — `Key` — создается самим Flash.

Впервые появился во Flash 5.

Key.addListener

Метод, добавляющий объект-перехватчик событий клавиатуры. Формат использования:

```
Key.addListener(<Объект-перехватчик>);
```

Единственным параметром этого метода передается ссылка на объект-перехватчик. Этот объект представляет собой экземпляр объекта `Object`, содержащий методы `onKeyDown` и `onKeyUp`.

Метод `onKeyDown` вызывается, когда пользователь нажимает клавишу клавиатуры, а метод `onKeyUp` — когда пользователь отпускает нажатую ранее клавишу. Этим методам не передаются никакие параметры.

Не возвращает значения.

Впервые появился во Flash MX.

Key.BACKSPACE

Свойство, возвращающее код клавиши `<BackSpace>` (8).

Впервые появилось во Flash 5.

Key.CAPSLOCK

Свойство, возвращающее код клавиши `<CapsLock>` (20).

Впервые появилось во Flash 5.

Key.CONTROL

Свойство, возвращающее код клавиши `<Ctrl>` (17).

Впервые появилось во Flash 5.

Key.DELETEKEY

Свойство, возвращающее код клавиши `` (46).

Впервые появилось во Flash 5.

Key.DOWN

Свойство, возвращающее код клавиши <Стрелка вниз> (40).

Впервые появилось во Flash 5.

Key.END

Свойство, возвращающее код клавиши <End> (35).

Впервые появилось во Flash 5.

Key.ENTER

Свойство, возвращающее код клавиши <Enter> (13).

Впервые появилось во Flash 5.

Key.ESCAPE

Свойство, возвращающее код клавиши <Escape> (27).

Впервые появилось во Flash 5.

Key.getAscii

Метод, возвращающий код ASCII последней нажатой клавиши. Не принимает параметров.

Впервые появился во Flash 5.

Key.getCode

Метод, возвращающий виртуальный код последней нажатой клавиши. Не принимает параметров.

Впервые появился во Flash 5.

Key.HOME

Свойство, возвращающее код клавиши <Home> (36).

Впервые появилось во Flash 5.

Key.INSERT

Свойство, возвращающее код клавиши <Insert> (45).

Впервые появилось во Flash 5.

Key.isDown

Метод, позволяющий проверить, нажата ли данная клавиша. Формат использования:

`Key.isDown(<Виртуальный код>)`

Если клавиша, чей виртуальный код был передан в качестве параметра, нажата, возвращается `true`. В противном случае возвращается `false`.

Впервые появился во Flash 5.

Key.isToggled

Метод, позволяющий проверить, включена ли клавиша `<CapsLock>` или `<NumLock>`. Формат использования:

`Key.isToggled(<Виртуальный код>)`

Если клавиша, чей виртуальный код был передан в качестве параметра, включена, возвращается `true`. В противном случае возвращается `false`.

Виртуальный код клавиши `<CapsLock>` — 20, `<NumLock>` — 144.

Впервые появился во Flash 5.

Key.LEFT

Свойство, возвращающее код клавиши `<Стрелка влево>` (37).

Впервые появилось во Flash 5.

Key.onKeyDown

Обработчик события объекта-перехватчика событий клавиатуры, происходящего при нажатии клавиши. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Key.onKeyUp

Обработчик события объекта-перехватчика событий клавиатуры, происходящего при отпускании нажатой ранее клавиши. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Key.PGDN

Свойство, возвращающее код клавиши <PgDn> (34).

Впервые появилось во Flash 5.

Key.PGUP

Свойство, возвращающее код клавиши <PgUp> (33).

Впервые появилось во Flash 5.

Key.removeListener

Метод, удаляющий добавленный ранее объект-перехватчик. Формат использования:

`Key.removeListener(<Объект-перехватчик>)`

Возвращает `true`, если объект-перехватчик был успешно удален, и `false` в противном случае.

Впервые появился во Flash MX.

Key.RIGHT

Свойство, возвращающее код клавиши <Стрелка вправо> (39).

Впервые появилось во Flash 5.

Key.SHIFT

Свойство, возвращающее код клавиши <Shift> (16).

Впервые появилось во Flash 5.

Key.SPACE

Свойство, возвращающее код клавиши пробела (32).

Впервые появилось во Flash 5.

Key.TAB

Свойство, возвращающее код клавиши <Tab> (9).

Впервые появилось во Flash 5.

Key.UP

Свойство, возвращающее код клавиши <Стрелка вверх> (38).

Впервые появилось во Flash 5.

le

Оператор сравнения "не больше". Формат использования:

`<Аргумент 1> le <Аргумент 2>`

Возвращает `true`, если строковое представление Аргумента 1 меньше или равно строковому представлению Аргумента 2, и `false` в противном случае.

Впервые появился во Flash 4. Во Flash 5 объявлен устаревшим и нерекомендованным к использованию; вместо него рекомендуется применять оператор `<=`.

length

Функция, возвращающая длину строки. Формат использования:

`length(<Строковое выражение>)`

Впервые появилась во Flash 4. Во Flash 5 объявлена устаревшей и нерекомендованной к использованию; вместо нее рекомендуется применять свойство `length` объекта `String`.

_level

Ключевое слово, ссылающееся на клип, загруженный на определенный уровень. Формат использования:

`_level<Номер уровня>`

Уровни нумеруются, начиная с нуля. Основной фильм всегда грузится на нулевой уровень (`_level0`).

Впервые появилось во Flash 4.

loadMovie

Действие, загружающее внешний файл Shockwave/Flash или JPEG и выводящее его содержимое как встроенный клип. Формат использования:

`loadMovie("<Интернет-адрес клипа или изображения>",`

`⚡<Путь нового клипа>[, <Метод отправки данных>]);`

Первым параметром передается интернет-адрес файла в строковом виде или адрес серверного приложения, которое сформирует этот файл. Имейте в виду, что он должен находиться в том же домене, что и основной фильм. Вторым параметром передается путь к новому клипу. Необязательный третий параметр задает метод передачи данных серверному приложению. Он может принимать два значения:

□ GET — для передачи данных используется метод GET;

□ POST — для передачи данных используется метод POST.

Если третий параметр пропущен, данные не передаются.

Впервые появилось во Flash 3.

loadMovieNum

Действие, загружающее внешний файл Shockwave/Flash или JPEG и выводящее его содержимое как встроенный клип. Формат использования:

```
loadMovieNum("<Интернет-адрес клипа или изображения>",  
⚡<Уровень нового клипа>[, <Метод отправки данных>]);
```

Первым параметром передается интернет-адрес файла в строковом виде или адрес серверного приложения, которое сформирует этот файл. Имейте в виду, что он должен находиться в том же домене, что и основной фильм. Вторым параметром в виде целого числа передается уровень нового клипа. Если задан нулевой уровень, новый клип заменит основной фильм. Необязательный третий параметр задает метод передачи данных серверному приложению. Он может принимать два значения:

- GET — для передачи данных используется метод GET;
- POST — для передачи данных используется метод POST.

Если третий параметр пропущен, данные не передаются.

Впервые появилось во Flash 4.

loadVariables

Действие, загружающее данные из серверного приложения или текстового файла. Формат использования:

```
loadVariables("Интернет-адрес", "Путь клипа"[, GET|POST]);
```

Первым параметром передается интернет-адрес серверного приложения или файла в строковом виде. Вторым параметром передается путь к клипу, который примет эти данные. Необязательный третий параметр задает метод передачи аргументов серверному приложению. Он может принимать два значения:

- GET — для передачи аргументов используется метод GET;
- POST — для передачи аргументов используется метод POST.

Если третий параметр пропущен, никакие аргументы не передаются.

Загружаемые данные должны быть представлены в текстовом виде как набор пар "переменная" = "значение", разделенных знаком "коммерческое и" & (то есть, так, как кодируются данные, передаваемые методом GET).

Впервые появилось во Flash 4.

loadVariablesNum

Действие, загружающее данные из серверного приложения или текстового файла. Формат использования:

```
loadVariables("Интернет-адрес", "Уровень клипа"[, GET|POST]);
```

Первым параметром передается интернет-адрес серверного приложения или файла в строковом виде. Вторым параметром в виде целого числа передается уровень клипа, который примет эти данные. Необязательный третий параметр задает метод передачи аргументов серверному приложению. Он может принимать два значения:

□ GET — для передачи аргументов используется метод GET;

□ POST — для передачи аргументов используется метод POST.

Если третий параметр пропущен, никакие аргументы не передаются.

Загружаемые данные должны быть представлены в текстовом виде как набор пар "переменная" = "значение", разделенных знаком "коммерческое и" & (то есть так, как кодируются данные, передаваемые методом GET).

Впервые появилось во Flash 4.

LoadVars

Объект, позволяющий обмениваться данными с серверным приложением. Также позволяет загружать данные из текстового файла.

Для создания экземпляра этого объекта используется конструктор:

```
<Переменная> = new LoadVars();
```

Затем создайте в этом экземпляре необходимые свойства и присвойте им нужные значения.

```
<Экземпляр объекта LoadVars>.<Свойство> = <Значение>;
```

Тогда введенные вами данные будут переданы в виде пары "Свойство"="Значение".

```
myLoadVars = new LoadVars();  
myLoadVars.name1 = "Ivan";  
myLoadVars.name2 = "Ivanov";  
myLoadVars.password = "vanyusha";
```

Впервые появился во Flash MX.

LoadVars.contentType

Свойство, задающее тип кодирования передаваемых и принимаемых данных, так называемый тип *MIME* (Multipurpose Internet Mail Extensions —

многоцелевые расширения почты Интернета). Значение по умолчанию — `application/x-www-form-urlencoded`.

Впервые появилось во Flash MX.

LoadVars.getBytesLoaded

Метод, возвращающий число загруженных байтов данных. Не принимает параметров.

Если операция загрузки данных не была запущена или реально еще не началась, возвращается `undefined`.

Впервые появился во Flash MX.

LoadVars.getBytesTotal

Метод, возвращающий общий размер загружаемых данных в байтах. Не принимает параметров.

Если операция загрузки данных не была запущена или реально еще не началась, возвращается `undefined`.

Впервые появился во Flash MX.

LoadVars.load

Метод, загружающий данные из текстового файла или принимающий их от серверного приложения. Формат использования:

```
<Экземпляр объекта LoadVars>.load(<Интернет-адрес>);
```

В качестве единственного параметра этого метода принимается интернет-адрес серверного приложения или текстового файла.

После загрузки данные раскодируются и помещаются в соответствующие им свойства экземпляра объекта `LoadVars`.

Не возвращает значения.

Впервые появился во Flash MX.

LoadVars.loaded

Свойство, позволяющее узнать, полностью ли загружены данные. Возвращает `true`, если данные загружены полностью, `false`, если не полностью, и `undefined`, если загрузка данных не была запущена или реально не началась.

Впервые появилось во Flash MX.

LoadVars.onLoad

Обработчик события, происходящего после успешной или неуспешной загрузки данных. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события принимает единственный параметр. Если он равен `true`, то данные были приняты нормально, если `false`, то во время их загрузки произошла ошибка. Функция-обработчик не возвращает значения.

Впервые появился во Flash MX.

LoadVars.send

Метод, отправляющий данные серверному приложению. Формат использования:

```
<Экземпляр объекта LoadVars>.send("<Интернет-адрес>"[, "<Цель>",  
    <Метод передачи данных>]);
```

Первым параметром передается интернет-адрес серверного приложения, которое получит эти данные. Вторым параметром передается имя фрейма, в который будет загружена сформированная этим приложением страница, или одно из предопределенных значений:

- ❑ `_self` — страница загружается в текущий фрейм текущего окна (значение по умолчанию);
- ❑ `_blank` — страница загружается в новом окне;
- ❑ `_parent` — страница загружается во фрейм, являющийся внешним для текущего фрейма текущего окна, или в само это окно;
- ❑ `_top` — страница загружается в текущее окно, заполняя его целиком.

Третий параметр задает метод передачи данных серверному приложению. Он может принимать два значения:

- ❑ `GET` — для передачи данных используется метод `GET`;
- ❑ `POST` — для передачи данных используется метод `POST`.

Если третий параметр пропущен, данные передаются методом `POST`.

Впервые появился во Flash MX.

LoadVars.sendAndLoad

Метод, отправляющий данные серверному приложению и тотчас принимающий ответ. Формат использования:

```
<Экземпляр объекта LoadVars>.sendAndLoad("<Интернет-адрес>",  
    <Экземпляр объекта LoadVars, принимающий данные>  
    [, <Метод передачи данных>]);
```

Первым параметром передается интернет-адрес серверного приложения, которое получит эти данные. Вторым параметром передается ссылка на экземпляр объекта `LoadVars`, который получит результат; он может быть тем же экземпляром, что отправляет данные или совсем другим. Третий параметр задает метод передачи данных серверному приложению. Он принимает два значения:

- GET — для передачи данных используется метод GET;
- POST — для передачи данных используется метод POST.

Если третий параметр пропущен, данные передаются методом POST.

Впервые появился во Flash MX.

LoadVars.toString

Метод, возвращающий строку, содержащую подготовленные к передаче данные экземпляра объекта `LoadVars`. Не принимает параметров.

Впервые появился во Flash MX.

lt

Оператор сравнения "меньше". Формат использования:

<Аргумент 1> lt <Аргумент 2>

Возвращает `true`, если строковое представление Аргумента 1 меньше строкового представления Аргумента 2, и `false` в противном случае.

Впервые появился во Flash 4. Во Flash 5 объявлен устаревшим и нерекомендованным к использованию; вместо него рекомендуется применять оператор `<`.

Math

Объект, предоставляющий доступ к различным математическим функциям и константам.

Единственный экземпляр этого объекта под именем `Math` создается самим Flash.

Впервые появился во Flash 4.

Math.abs

Метод, возвращающий модуль (абсолютное значение) числа. Формат использования:

`Math.abs(<Числовое выражение>)`

Впервые появился во Flash 4.

Math.acos

Метод, возвращающий величину арккосинуса в радианах. Формат использования:

```
Math.acos(<Числовое выражение>)
```

Впервые появился во Flash 4.

Math.asin

Метод, возвращающий величину арксинуса в радианах. Формат использования:

```
Math.asin(<Числовое выражение>)
```

Впервые появился во Flash 4.

Math.atan

Метод, возвращающий величину арктангенса в радианах. Формат использования:

```
Math.atan(<Числовое выражение>)
```

Впервые появился во Flash 4.

Math.atan2

Метод, возвращающий величину арктангенса в радианах. Формат использования:

```
Math.atan(<X>, <Y>)
```

Возвращается арктангенс от x , деленного на y .

Впервые появился во Flash 4.

Math.ceil

Метод, возвращающий целое число, равное или ближайшее большее к аргументу. Формат использования:

```
Math.ceil(<Числовое выражение>)
```

Впервые появился во Flash 4.

Math.cos

Метод, возвращающий косинус угла, заданного в радианах. Формат использования:

```
Math.cos(<Числовое выражение>)
```

Впервые появился во Flash 4.

Math.E

Свойство, возвращающее значение основания натурального логарифма e (2,71828).

Впервые появилось во Flash 4.

Math.exp

Метод, возвращающий значение натурального антилогарифма (e^x). Формат использования:

`Math.exp(<Числовое выражение>)`

Впервые появился во Flash 4.

Math.floor

Метод, возвращающий целое число, равное или ближайшее меньшее к аргументу. Формат использования:

`Math.floor(<Числовое выражение>)`

Впервые появился во Flash 4.

Math.log

Метод, возвращающий значение натурального логарифма. Формат использования:

`Math.log(<Числовое выражение>)`

Впервые появился во Flash 4.

Math.LOG2E

Свойство, возвращающее значение двоичного логарифма от e (1,4427).

Впервые появилось во Flash 4.

Math.LOG10E

Свойство, возвращающее значение десятичного логарифма от e (0,43429).

Впервые появилось во Flash 4.

Math.LN2

Свойство, возвращающее значение натурального логарифма от 2 (0,69315).

Впервые появилось во Flash 4.

Math.LN10

Свойство, возвращающее значение натурального логарифма от 10 (2,30259).
Впервые появилось во Flash 4.

Math.max

Метод, возвращающий максимальное из двух переданных значений. Формат использования:

`Math.max(<Числовое выражение 1>, <Числовое выражение 2>)`

Впервые появился во Flash 4.

Math.min

Метод, возвращающий меньшее из двух переданных значений. Формат использования:

`Math.min(<Числовое выражение 1>, <Числовое выражение 2>)`

Впервые появился во Flash 4.

Math.PI

Свойство, возвращающее значение константы π (3,14159).

Впервые появилось во Flash 4.

Math.pow

Метод, возвращающий значение x^y . Формат использования:

`Math.pow(<X>, <Y>)`

Впервые появился во Flash 4.

Math.random

Метод, возвращающий псевдослучайное число в диапазоне между 0 и 1. Не принимает параметров.

Впервые появился во Flash 4.

Math.round

Метод, возвращающий значение параметра, округленное до ближайшего целого. Формат использования:

`Math.round(<Числовое выражение>)`

Впервые появился во Flash 4.

Math.sin

Метод, возвращающий синус угла, заданного в радианах. Формат использования:

```
Math.sin(<Числовое выражение>)
```

Впервые появился во Flash 4.

Math.sqrt

Метод, возвращающий значение квадратного корня. Формат использования:

```
Math.sqrt(<Числовое выражение>)
```

Значение параметра должно быть равно или больше нуля.

Впервые появился во Flash 4.

Math.SQRT1_2

Свойство, возвращающее значение квадратного корня от 0,5 (0,70711).

Впервые появилось во Flash 4.

Math.SQRT2

Свойство, возвращающее значение квадратного корня от 2 (1,41421).

Впервые появилось во Flash 4.

Math.tan

Метод, возвращающий тангенс угла, заданного в радианах. Формат использования:

```
Math.tan(<Числовое выражение>)
```

Впервые появился во Flash 4.

maxscroll

Свойство, возвращающее номер самой верхней строки, видимой в многострочном поле ввода или динамическом текстовом блоке, при условии, что дальнейшая прокрутка его содержимого невозможна. Другими словами, максимальное значение вертикальной полосы прокрутки. Формат использования:

```
<Переменная, привязанная к текстовому полю>.maxscroll
```

Доступно только для чтения.

Впервые появилось во Flash 4. Во Flash 5 объявлено устаревшим и не рекомендованным к использованию; вместо него рекомендуется применять свойство `maxscroll` объекта `TextField`.

mbchr

Функция, принимающая многобайтовый код символа ASCII/S-JIS, использующийся для представления иероглифических тестов, и возвращающая его строковое представление. Формат использования:

```
mbchr(<Код символа>)
```

Впервые появилась во Flash 4. Начиная с Flash 5, объявлена устаревшей и не рекомендована к использованию; вместо нее рекомендуется употреблять метод `fromCharCode` объекта `String`.

mblength

Функция, возвращающая длину строки многобайтовых символов (кодировка ASCII/S-JIS). Формат использования:

```
mblength(<Строковое выражение>)
```

Впервые появилась во Flash 4. Во Flash 5 объявлена устаревшей и не рекомендованной к использованию; вместо нее рекомендуется применять свойство `length` объекта `String`.

mbord

Функция, возвращающая код многобайтового символа (кодировка ASCII/S-JIS). Формат использования:

```
mbord(<Символ>)
```

Впервые появилась во Flash 4. Во Flash 5 объявлена устаревшей и не рекомендованной к использованию; вместо нее рекомендуется применять свойство `charAt` объекта `String`.

mbsubstring

Функция, возвращающая подстроку из многобайтовой строки (кодировка ASCII/S-JIS). Формат использования:

```
mbsubstring(<Строка>, <Первый символ подстроки>, <Количество символов>)
```

Первым параметром задается сама строка, из которой будет выделена подстрока. Вторым параметром передается номер первого символа подстроки. Третьим параметром задается количество символов, которые должны войти в подстроку.

Впервые появилась во Flash 4. Во Flash 5 объявлена устаревшей и не рекомендованной к использованию; вместо нее рекомендуется применять свойство `substr` объекта `String`.

method

Это действие существует только в виде пункта иерархического списка панели **Actions**. Применяется для создания метода объекта.

Впервые появилось во Flash MX.

Mouse

Объект, предоставляющий доступ к некоторым свойствам мыши.

Единственный экземпляр объекта `Mouse` под тем же именем — `Mouse` — создается самим Flash.

Впервые появился во Flash 5.

Mouse.addListener

Метод, добавляющий объект-перехватчик событий клавиатуры. Формат использования:

```
Mouse.addListener (<Объект-перехватчик>);
```

Единственным параметром этого метода передается ссылка на объект-перехватчик. Этот объект представляет собой экземпляр объекта `Object`, содержащий методы `onMouseDown`, `onMouseMove` и `onMouseUp`.

Метод `onMouseDown` вызывается, когда пользователь нажимает левую кнопку мыши, а метод `onMouseUp` — когда пользователь отпускает нажатую ранее кнопку. Метод `onMouseMove` вызывается при любом перемещении мыши пользователем. Этим методам не передаются никакие параметры.

Не возвращает значения.

Впервые появился во Flash MX.

Mouse.hide

Метод, скрывающий курсор мыши. Не принимает параметров и не возвращает значения.

Впервые появился во Flash 5.

Mouse.onMouseDown

Обработчик события объекта-перехватчика событий мыши, происходящего при нажатии ее левой кнопки. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Mouse.onMouseMove

Обработчик события объекта-перехватчика событий мыши, происходящего при любом ее перемещении. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Mouse.onMouseUp

Обработчик события объекта-перехватчика событий клавиатуры, происходящего при отпускании нажатой ранее ее левой кнопки. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Mouse.removeListener

Метод, удаляющий добавленный ранее объект-перехватчик. Формат использования:

```
Mouse.removeListener(<Объект-перехватчик>)
```

Возвращает `true`, если объект-перехватчик был успешно удален, и `false` в противном случае.

Впервые появился во Flash MX.

Mouse.show

Метод, выводящий на экран скрытый методом `hide` курсор мыши. Не принимает параметров и не возвращает значения.

Впервые появился во Flash 5.

MovieClip

Объект, представляющий основной фильм и любой встроенный клип, в том числе, компонент.

Для каждого встроенного клипа, помещенного на рабочий лист в среде Flash или из сценария, создается отдельный экземпляр объекта `MovieClip`. Этот экземпляр получает имя, заданное в редакторе свойств или одним из параметров метода `attachMovie` объекта `MovieClip`.

Объект `MovieClip` не наследуется от `Object`.

Впервые появился во Flash 4.

MovieClip._alpha

Свойство, задающее прозрачность встроенного клипа. Допускаются целые числовые значения от 0 (полная прозрачность) до 100 (полная непрозрачность). При этом клип остается активным, даже если он совершенно невидим (то есть для свойства `_alpha` было задано значение 0).

Впервые появилось во Flash 4.

MovieClip.attachMovie

Метод, создающий на рабочем листе новый клип, основанный на сценарном образце-клипе и вложенный в текущий клип или основной фильм. Формат использования:

```
<Внешний клип>.attachMovie("<Имя разделяемого образца>",  
    ⚡ "<Имя нового клипа>", <Уровень клипа>[, <Клип-инициализатор>]);
```

Первый параметр задает в строковом виде имя сценарного образца-клипа в библиотеке. Второй параметр также в строковом виде задает имя нового клипа. Третий параметр — это целое число, задающее уровень нового клипа. Последний параметр может задавать уже существующий клип, у которого новый клип "позаимствует" значения свойств.

Не возвращает значения.

```
_root.attachMovie("car", "car", 1);  
_root.car.attachMovie("wheel", "wheel1", 1);  
_root.car.attachMovie("wheel", "wheel2", 1, "wheel1");
```

Впервые появился во Flash 5.

MovieClip.beginFill

Метод, служащий для задания параметров сплошной заливки. После его вызова Flash будет применять сплошную заливку ко всем замкнутым контурам, пока вы не вызовете метод `endFill`. Формат использования:

```
<Клип>.beginFill([<Цвет линии>, [<Прозрачность заливки>]]);
```

Цвет линии задается в формате 0xRRGGBB, а Прозрачность заливки — значением от 0 до 100. Хотя оба параметра не являются обязательными, лучше задать их, иначе результат может быть непредсказуемым.

Не возвращает значения.

Впервые появился во Flash MX.

MovieClip.beginGradientFill

Метод, служащий для задания параметров градиентной заливки. После его вызова Flash будет применять градиентную — линейную или радиальную — заливку ко всем замкнутым контурам, пока вы не вызовете метод `endFill`.
Формат использования:

```
<Клип>.beginGradientFill(<Тип>, <Цвета>, <Прозрачности>, <Площади>,  
    <Преобразования>);
```

Первый параметр задает тип градиентной заливки: линейная ("linear") или радиальная ("radial").

Второй параметр представляет собой массив ключевых цветов градиента. Их должно быть минимум два.

Третий параметр представляет собой массив значений прозрачности для каждого ключевого цвета. Их должно быть столько же, сколько значений цветов.

Четвертый параметр представляет собой массив значений, задающих относительную площадь, занимаемую каждым цветом. Их должно быть столько же, сколько значений цветов. Каждое из этих значений задает позицию, где представлен соответствующий цвет в чистом виде, и должно находиться в диапазоне между 0 и 255 (0xFF). Например, для массива площадей [0, 128, 255] первый цвет находится у самого начала градиента, второй — на полпути к концу и третий — у самого конца.

Пятый параметр задает преобразования градиента. Он имеет вид экземпляра объекта `Object`, имеющего особые свойства, и бывает двух видов.

Первый вид — самый простой. Экземпляр объекта должен содержать такие свойства:

- `matrixType` — всегда равен "box";
- `x` — горизонтальная координата верхнего левого угла заливки относительно точки фиксации клипа;
- `y` — вертикальная координата верхнего левого угла заливки относительно точки фиксации клипа;
- `w` — ширина заливки;
- `h` — высота заливки;
- `r` — угол поворота заливки в радианах.

```
m = {matrixType:"box", x:100, y:100, w:200, h:200, r:(45 / 180) *  
    Math.PI};
```

Второй вид пятого параметра — сложный. Свойства экземпляра объекта задают значения матрицы преобразования градиента:

a	b	c
d	e	f
g	h	i

```
m = {a:200, b:0, c:0, d:0, e:200, f:0, g:200, h:200, i:1};
```

Не возвращает значения.

Впервые появился во Flash MX.

MovieClip.clear

Метод, удаляющий всю графику, созданную программно. Не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.createEmptyMovieClip

Метод, создающий новый пустой клип, вложенный в текущий клип или основной фильм. Формат использования:

```
<Внешний клип>.createEmptyMovieClip("<Имя нового клипа>",  
    ⚡<Уровень клипа>);
```

Первый параметр задает имя нового клипа в строковом виде. Второй параметр — это целое число, задающее уровень нового клипа.

Не возвращает значения.

Впервые появился во Flash MX.

MovieClip.createTextField

Метод, создающий поле ввода или динамический текстовый блок внутри текущего клипа или основного фильма. Формат использования:

```
<Внешний клип>.createTextField(<Имя>, <Уровень>, <X>, <Y>, <Ширина>,  
    ⚡<Высота>);
```

Первый параметр задает имя создаваемого поля ввода или динамического текстового блока в строковом виде. Второй параметр задает уровень в виде целого числа. Третий и четвертый параметры задают соответственно горизонтальную и вертикальную координаты верхнего левого угла создаваемого поля ввода (динамического текстового блока) относительно точки фиксации внешнего клипа. Последние два параметра задают размеры — ширину и высоту — поля ввода (динамического текстового блока). И координаты, и размеры задаются в пикселах.

Не возвращает значения.

Впервые появился во Flash MX.

MovieClip._currentframe

Свойство, возвращающее номер клипа, на котором стоит указатель. Доступно только для чтения.

Впервые появилось во Flash 4.

MovieClip.curveTo

Метод, рисующий кривую из текущей точки. Формат использования:

```
<Клип>.curveTo(<AX>, <AY>, <X>, <Y>);
```

Параметры *x* и *y* задают соответственно горизонтальную и вертикальную координаты конечной точки кривой, которая потом станет текущей, в пикселах. Параметры *ax* и *ay* задают горизонтальную и вертикальную координаты точки искривления. Все эти координаты отсчитываются опять же относительно точки фиксации клипа.

Не возвращает значения.

Впервые появился во Flash MX.

MovieClip._droptarget

Свойство, возвращающее путь к клипу, в который был "брошен" текущий клип после завершения операции drag'n'drop. Путь возвращается в виде "запись со слэшем"; для преобразования его в обычную "запись с точкой" следует воспользоваться функцией `eval`.

Доступно только для чтения.

Впервые появилось во Flash 4.

MovieClip.cloneMovieClip

Метод, создающий копию клипа. Формат использования:

```
<Клип>.cloneMovieClip(<Имя>, <Уровень>[, <Клип-инициализатор>]);
```

Первый параметр задает в строковом виде имя нового клипа. Второй параметр — это целое число, задающее уровень нового клипа. Третий параметр может задавать уже существующий клип, у которого новый клип "позаймствует" значения свойств.

Проигрывание нового клипа всегда начинается с первого кадра, независимо от того, на каком кадре исходного клипа находился указатель. Содержимое переменных исходного клипа не копируется в переменные нового клипа. Если исходный клип был удален, новый клип также удаляется.

Для удаления созданной копии клипа воспользуйтесь действием или методом `removeMovieClip`.

Впервые появился во Flash 5.

MovieClip.enabled

Свойство, разрешающее или запрещающее доступ к клипу. Если присвоено значение `true`, клип доступен, если `false` — недоступен и не реагирует на щелчки. По умолчанию клип доступен (значение `true`).

Впервые появилось во Flash MX.

MovieClip.endFill

Метод, завершающий рисование заливки. Не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.focusEnabled

Свойство, разрешающее или запрещающее клипу получать фокус ввода, если он не является кнопкой. Если присвоено значение `true`, клип может принимать фокус ввода, если `false` или `undefined` — не может.

Впервые появилось во Flash MX.

MovieClip._focusrect

Свойство, задающее, будет ли клип иметь желтую рамку, если на нем находится фокус ввода. Если присвоено значение `true`, то клип имеет такую рамку, если `false` — не имеет. По умолчанию это свойство имеет то же значение, что и глобальная переменная `_focusrect`.

Впервые появилось во Flash MX.

MovieClip._framesloaded

Свойство, возвращающее количество загруженных кадров клипа или основного фильма. Доступно только для чтения.

Впервые появилось во Flash 4.

MovieClip.getBounds

Метод, позволяющий узнать координаты текущего клипа относительно точки фиксации другого клипа. Формат использования:

```
<Клип>.getBounds (<Исходный клип>);
```

В качестве единственного параметра задается ссылка на клип, точка фиксации которого послужит началом координат.

Метод возвращает экземпляр объекта `Object`, содержащий следующие свойства:

- ❑ `xMin` — горизонтальная координата левой границы клипа;
- ❑ `xMax` — горизонтальная координата правой границы клипа;
- ❑ `yMin` — вертикальная координата верхней границы клипа;
- ❑ `yMax` — вертикальная координата нижней границы клипа.

Впервые появился во Flash 5.

MovieClip.getBytesLoaded

Метод, возвращающий количество загруженных байтов клипа. Не принимает параметров.

Впервые появился во Flash MX.

MovieClip.getBytesTotal

Метод, возвращающий общий размер загружаемых данных в байтах. Для основного фильма фактически возвращает размер файла Shockwave/Flash, в котором хранится этот фильм. Не принимает параметров.

Впервые появился во Flash 5.

MovieClip.getDepth

Метод, возвращающий уровень клипа в виде целого числа. Не принимает параметров.

Впервые появился во Flash MX.

MovieClip.getURL

Метод, загружающий Web-страницу и выводящий ее в окне Web-обозревателя. Может также применяться для передачи данных серверному приложению и приема от него результатов. Формат использования:

```
<Клип>.getURL(<Интернет-адрес>[, <Цель>[, <Метод передачи данных>]]);
```

Первым параметром передается интернет-адрес нужной Web-страницы или серверного приложения. Вторым параметром передается имя фрейма, в который будет загружена страница, или одно из предопределенных значений:

- ❑ `_self` — страница загружается в текущий фрейм текущего окна (значение по умолчанию);
- ❑ `_blank` — страница загружается в новом окне;

□ `_parent` — страница загружается во фрейм, являющийся внешним для текущего фрейма текущего окна, или в само это окно;

□ `_top` — страница загружается в текущее окно, заполняя его целиком.

Третий параметр задает метод передачи данных серверному приложению. Он может принимать два значения:

□ `"GET"` — для передачи данных используется метод `GET`;

□ `"POST"` — для передачи данных используется метод `POST`.

Если третий параметр пропущен, данные не передаются.

Не возвращает значения.

Впервые появился во Flash 5.

MovieClip.globalToLocal

Метод, преобразующий координаты, заданные относительно основного фильма, в координаты, заданные относительно клипа. Формат использования:

```
<Клип>.globalToLocal(<Точка>);
```

Единственный параметр этого метода представляет собой экземпляр объекта `Object`, содержащий свойства `x` и `y`. Этими свойствами и задаются нужные координаты. При вызове метода они заменяются новыми.

Не возвращает значения.

Впервые появился во Flash 5.

MovieClip.gotoAndPlay

Метод, перемещающий указатель на заданный кадр клипа и продолжающий проигрывание фильма с него. Формат использования:

```
<Клип>.gotoAndPlay(<Номер или имя кадра>);
```

Не возвращает значения.

Впервые появился во Flash 5.

MovieClip.gotoAndStop

Метод, перемещающий указатель на заданный кадр клипа и останавливающий проигрывание фильма. Формат использования:

```
<Клип>.gotoAndStop(<Номер или имя кадра>);
```

Не возвращает значения.

Впервые появился во Flash 5.

MovieClip._height

Свойство, задающее высоту клипа в пикселах.

Впервые появилось во Flash 4.

MovieClip._highquality

Свойство, задающее качество сглаживания растровой графики во всем фильме. Может принимать три числовых значения:

- 2 — растровая графика сглаживается всегда;
- 1 — растровая графика сглаживается только тогда, когда фильм не содержит анимации;
- 0 — растровая графика никогда не сглаживается.

Используйте это свойство, чтобы уменьшить количество системных ресурсов, потребляемых проигрывателем Flash. Это может понадобиться на медленных компьютерах.

Очень странно, но, судя по всему, это свойство просто ссылается на системную переменную `_highquality`, выполняющую ту же функцию. Зачем нужно было создавать еще и свойство `_highquality`, неясно.

Впервые появилось во Flash MX.

MovieClip.hitArea

Свойство, позволяющее задать другой клип в качестве "горячей" области клипа-кнопки. Для этого нужно всего лишь присвоить этому свойству ссылку на нужный клип. Если свойство равно `undefined` или `null`, в качестве "горячей" области используется сам клип-кнопка.

Впервые появилось во Flash MX.

MovieClip.hitTest

Метод, позволяющий узнать, совпадают ли два клипа, или попадает ли какая-то точка внутрь клипа. Форматы использования:

```
<Клип>.hitTest(<X>, <Y>, <Проверять контур>);
```

```
<Клип>.hitTest(<Другой клип>);
```

В первом формате параметры `x` и `y` задают соответственно горизонтальную и вертикальную координату нужной вам точки в пикселах. Третий параметр имеет логический тип и позволяет задать, что считать границами клипа: его контур (значение `true`) или окружающий его прямоугольник (значение `false`).

Во втором формате в качестве параметра передается ссылка на другой клип.

Если имеют место совпадения клипа с клипом, или точка попадает внутрь клипа, возвращается `true`. В противном случае возвращается `false`.

Впервые появился во Flash 5.

MovieClip.lineStyle

Метод, задающий стиль рисуемой линии. Формат использования:

```
<Клип>.lineStyle([<Толщина линии>, [<Цвет линии>,  
♣ [<Прозрачность линии>]]]);
```

Толщина линии задается в пикселах, от 0 до 255. Если она не задана, линия не будет нарисована. Цвет линии задается в формате 0xRRGGBB. Прозрачность линии может принимать значения от 0 (полная прозрачность) до 100 (полная непрозрачность).

Перед рисованием любых линий рекомендуется задать их стиль, иначе результаты будут непредсказуемы.

Не возвращает значения.

Впервые появился во Flash MX.

MovieClip.lineTo

Метод, рисующий прямую линию из текущей точки. Формат использования:

```
<Клип>.lineTo(<X>, <Y>);
```

Параметры `x` и `y` задают соответствующие координаты конечной точки прямой в пикселах. Эта точка потом станет текущей.

Не возвращает значения.

Впервые появился во Flash MX.

MovieClip.loadMovie

Метод, загружающий внешний файл Shockwave/Flash или JPEG и выводящий его содержимое как встроенный клип. Формат использования:

```
<Внешний клип>.loadMovie("<Интернет-адрес клипа или изображения>",  
♣ [, <Метод отправки данных>]);
```

Первым параметром передается интернет-адрес файла в строковом виде или адрес серверного приложения, которое сформирует этот файл. Имейте в виду, что он должен находиться в том же домене, что и основной фильм. Необязательный второй параметр задает метод передачи данных серверному приложению. Он может принимать два значения:

□ "GET" — для передачи данных используется метод GET;

□ "POST" — для передачи данных используется метод POST.

Если второй параметр пропущен, данные не передаются.

Не возвращает значения.

Впервые появился во Flash 5.

MovieClip.loadVariables

Метод, загружающий данные из серверного приложения или текстового файла и передающий их в текущий клип. Формат использования:

```
<Клип>.loadVariables("Интернет-адрес" [, GET|POST]);
```

Первым параметром передается интернет-адрес серверного приложения или файла в строковом виде. Необязательный второй параметр задает метод передачи аргументов серверному приложению. Он может принимать два значения:

□ "GET" — для передачи аргументов используется метод GET;

□ "POST" — для передачи аргументов используется метод POST.

Если второй параметр пропущен, никакие аргументы не передаются.

Загружаемые данные должны быть представлены в текстовом виде как набор пар "переменная" = "значение", разделенных знаком "коммерческое и" & (то есть, так, как кодируются данные, передаваемые методом GET).

Не возвращает значения.

Впервые появился во Flash 5.

MovieClip.localToGlobal

Метод, преобразующий координаты, заданные относительно клипа, в координаты, заданные относительно основного фильма. Формат использования:

```
<Клип>.localToGlobal(<Точка>);
```

Единственный параметр этого метода представляет собой экземпляр объекта Object, содержащий свойства x и y. Этими свойствами и задаются нужные координаты. При вызове метода они заменяются новыми.

Не возвращает значения.

Впервые появился во Flash 5.

MovieClip.moveTo

Метод, задающий новую текущую точку рисования. Формат использования:

```
<Клип>.moveTo(<X>, <Y>);
```

Параметры x и y задают соответствующие координаты новой текущей точки в пикселах.

Не возвращает значения.

Впервые появился во Flash MX.

MovieClip._name

Свойство, возвращающее имя клипа, заданное в редакторе свойств, в строковом виде. Доступно только для чтения.

Впервые появилось во Flash MX.

MovieClip.nextFrame

Метод, перемещающий указатель на следующий кадр и останавливающий проигрывание фильма. Не принимает параметров и не возвращает значения.

Впервые появился во Flash 5.

MovieClip.onData

Обработчик события, наступающего после того, как клип примет все данные до последней переменной. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Используйте именно это событие, чтобы обработать данные, принятые в результате выполнения методов `loadVariable` и `loadMovie`.

Впервые появился во Flash MX.

MovieClip.onDragOut

Обработчик события, наступающего, когда пользователь помещает курсор мыши над текущим клипом, нажимает левую кнопку мыши и "уводит" мышь прочь. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onDragOver

Обработчик события, наступающего, когда пользователь что-то перетаскивает над текущим клипом. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onEnterFrame

Обработчик события, постоянно происходящего с частотой, равной частоте кадров фильма. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Функция-обработчик события `onEnterFrame` выполняется перед всеми остальными сценами, привязанными к кадру.

Впервые появился во Flash MX.

MovieClip.onKeyDown

Обработчик события, наступающего, когда текущий клип имеет фокус ввода, и пользователь нажимает клавишу на клавиатуре. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onKeyUp

Обработчик события, наступающего, когда текущий клип имеет фокус ввода, и пользователь отпускает нажатую ранее клавишу. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onKillFocus

Обработчик события, наступающего, когда текущий клип теряет фокус ввода. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события принимает единственный параметр, передающий ссылку на объект, получивший фокус ввода. Если ни один объект

не получил фокус ввода, Flash присваивает этому параметру значение `null`. Функция-обработчик события не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onLoad

Обработчик события, наступающего сразу после окончания загрузки текущего клипа и появления его на рабочем листе. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onMouseDown

Обработчик события, наступающего, когда пользователь помещает над клипом курсор мыши и нажимает ее левую кнопку. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onMouseMove

Обработчик события, происходящего, когда пользователь перемещает над клипом курсор мыши. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onMouseUp

Обработчик события, наступающего, когда пользователь отпускает нажатую ранее левую кнопку мыши. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onPress

Обработчик события, наступающего, когда пользователь щелкает по текущему клипу-кнопке. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onRelease

Обработчик события, наступающего, когда пользователь "отпускает" текущий клип-кнопку после щелчка. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onReleaseOutside

Обработчик события, наступающего, когда пользователь "отпускает" текущий клип-кнопку после щелчка, причем курсор мыши находится за пределами клипа. Это может случиться, когда пользователь щелкает клип, не отпуская левой кнопки мыши, "уводит" мышь прочь и там отпускает кнопку. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметры и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onRollOut

Обработчик события, наступающего, когда пользователь "уводит" курсор мыши за пределы клипа. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onRollOver

Обработчик события, наступающего, когда пользователь помещает курсор мыши на клип. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onSetFocus

Обработчик события, наступающего, когда текущий клип получает фокус ввода. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события принимает единственный параметр, передающий ссылку на объект, потерявший фокус ввода. Если ни один объект не имел ранее фокуса ввода, Flash присваивает этому параметру значение `null`. Функция-обработчик события не возвращает значения.

Впервые появился во Flash MX.

MovieClip.onUnload

Обработчик события, наступающего после исчезновения текущего клипа с рабочего листа, в следующем же кадре. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Функция-обработчик события `onUnload` выполняется перед всеми остальными сценариями, привязанными к кадру.

Впервые появился во Flash MX.

MovieClip._parent

Свойство, возвращающее ссылку на внешний клип. Доступно только для чтения.

```
parentClip = someClip._parent;
```

Может быть использовано несколько раз:

```
parentClip = someClip._parent._parent._parent;
```

Впервые появилось во Flash MX.

MovieClip.play

Метод, запускающий проигрывание клипа. Не принимает параметров и не возвращает значения.

Впервые появился во Flash 5.

MovieClip.prevFrame

Метод, перемещающий указатель на предыдущий кадр и останавливающий проигрывание фильма. Не принимает параметров и не возвращает значения.

Впервые появился во Flash 5.

MovieClip.removeMovieClip

Метод, удаляющий клип, созданный с помощью действия или метода `duplicateMovieClip` или метода `attachMovie`. Не принимает параметров и не возвращает значения.

Впервые появился во Flash 5.

MovieClip._rotation

Свойство, задающее угол поворота клипа в градусах.

Впервые появилось во Flash MX.

MovieClip.setMask

Метод, маскирующий текущий клип другим клипом. Формат использования:

```
<Маскируемый клип>.setMask(<Клип-маска>);
```

Оба клипа должны находиться в разных слоях.

Чтобы убрать маскирование, присвойте методу `setMask` нужного клипа значение `null`.

Не возвращает значения.

Впервые появился во Flash MX.

MovieClip._soundbuftime

Свойство, задающее размер буфера звука в секундах. Этот буфер используется при загрузке потоковых звуков для того, чтобы обеспечить их плавное воспроизведение.

Это свойство просто ссылается на системную переменную `_soundbuftime`, выполняющую ту же функцию. Зачем нужно было создавать еще и свойство `_soundbuftime`, неясно.

Впервые появилось во Flash MX.

MovieClip.startDrag

Метод, разрешающий пользователю перетаскивать клип мышью. Формат использования:

```
<Клип>.startDrag([<За центр>, [<X1>, <Y1>, <X2>, <Y2>]]);
```

Первый параметр позволяет задать точку, к которой будет автоматически "приклеиваться" курсор мыши: точку, по которой пользователь щелкнул мышью (значение `false`) или центр клипа (значение `true`). Параметры `x1` и `y1` задают горизонтальную и вертикальную координаты левого верхнего угла воображаемого прямоугольника, внутри которого можно будет перетаскивать этот клип и за пределы которого он не сможет выйти. А за координаты правого нижнего угла "отвечают" параметры `x2` и `y2`. Эти координаты задаются относительно внешнего клипа в пикселах.

Не возвращает значения.

Впервые появился во Flash 5.

MovieClip.stop

Метод, останавливающий проигрывание клипа. Не принимает параметров и не возвращает значения.

Впервые появился во Flash 5.

MovieClip.stopDrag

Метод, запрещающий пользователю перетаскивать клип. Вызывается после вызова метода `startDrag`. Не принимает параметров и не возвращает значения.

Впервые появился во Flash 5.

MovieClip.swapDepth

Метод, позволяющий изменить уровень клипа. Форматы использования:

```
<Клип>.swapDepths (<Другой клип>);
```

```
<Клип>.swapDepths (<Уровень>);
```

Как видите либо просто задать новый уровень.

Не возвращает, этот метод позволяет либо "поменяться" уровнем с другим клипом, значения.

Впервые появился во Flash 5.

MovieClip.tabChildren

Свойство, позволяющее включить в порядок обхода по нажатиям клавиш `<Tab>` и `<Shift>+<Tab>` клипы, встроенные в текущий клип. Если равно `true` или `undefined`, то встроенные клипы включаются в порядок обхода, если `false` — не включаются.

Впервые появилось во Flash MX.

MovieClip.tabEnabled

Свойство, задающее, будет ли клип включен в порядок обхода по нажатиям клавиш `<Tab>` и `<Shift>+<Tab>`.

Если значение этого свойства равно `true` или `undefined` или свойству `tabIndex` присвоено целое число, то клип включается в порядок обхода. В противном случае он исключается из порядка обхода, но пользователь все еще может щелкать его мышью. Значение по умолчанию — `undefined`.

Впервые появилось во Flash MX.

MovieClip.tabIndex

Свойство, задающее место текущего клипа в порядке обхода. Может быть любым неотрицательным целым числом.

Если это свойство равно `undefined` и свойство `tabEnabled` не равно `false`, Flash сам формирует порядок обхода, зависящий от порядка, в котором кнопки и клипы были помещены на рабочий лист. Чтобы задать свой порядок обхода, присвойте всем элементам управления позиции в этом порядке, используя свойство `tabIndex`. Имейте в виду, что этот порядок не зависит от вложенности одних клипов в другие, т. е. он "плоский".

Значение по умолчанию — `undefined`.

Впервые появилось во Flash MX.

MovieClip._target

Свойство, возвращающее путь экземпляра текущего клипа в формате "запись со слэшем". Доступно только для чтения.

Впервые появилось во Flash 4.

MovieClip._totalframes

Свойство, возвращающее общее количество кадров в клипе. Доступно только для чтения.

Впервые появилось во Flash 4.

MovieClip.trackAsMenu

Свойство, задающее, будет ли клип-кнопка откликаться на события мыши, происходящие в других клипах-кнопках.

Если это свойство равно `false`, то клип-кнопка не будет откликаться на события, которые в данный момент происходят в других клипах-кнопках. Так, если вы перетаскиваете кнопку над другими кнопками, эти другие кнопки не будут реагировать на события `onRollOver` и `onRollOut`, даже не будут изменять свой вид. Если же вы установите это свойство в `true`, клип-кнопка будет реагировать на такие события во всех случаях.

Значение по умолчанию — `false`.

Это свойство можно также выставить в редакторе свойств, воспользовавшись раскрывающимся списком **Options for Buttons**. Пункт **Track as Button** эквивалентен значению `false`, а пункт **Track as Menu Item** — значению `true`.

Впервые появилось во Flash MX.

MovieClip.unloadMovie

Метод, удаляющий клип, загруженный с использованием методов `loadMovie` и `attachMovie`. Не принимает параметров и не возвращает значения.

Впервые появился во Flash 5.

MovieClip._url

Свойство, возвращающее интернет-адрес файла Shockwave/Flash, содержащего образец текущего клипа. Доступно только для чтения.

Впервые появилось во Flash 4.

MovieClip.useHandCursor

Свойство, задающее форму курсора мыши. Если равно `true`, то курсор мыши при наведении на текущий клип примет форму "перста указующего", если `false` — останется в виде стрелки. Значение по умолчанию — `true`.

Впервые появилось во Flash MX.

MovieClip._visible

Свойство, позволяющее скрыть клип. Если равно `true`, то клип виден и воспринимает щелчки, если `false` — не виден и не доступен. Значение по умолчанию — `true`.

Впервые появилось во Flash 4.

MovieClip._width

Свойство, задающее ширину клипа в пикселах.

Впервые появилось во Flash 4 в виде свойства, доступного только для чтения. Начиная с Flash 5 доступно для чтения и записи.

MovieClip._x

Свойство, задающее горизонтальную координату точки фиксации текущего клипа в пикселах. Отсчет ведется от левого верхнего угла, если клип находится в основном фильме, и от точки фиксации, если он вложен во встроенный клип.

Впервые появилось во Flash 3.

MovieClip._xmouse

Свойство, возвращающее горизонтальную координату курсора мыши относительно точки фиксации текущего клипа в пикселах. Доступно только для чтения.

Впервые появилось во Flash 5.

MovieClip._xscale

Свойство, задающее масштабирование текущего клипа по горизонтали в процентах.

Впервые появилось во Flash 4.

MovieClip._y

Свойство, задающее вертикальную координату точки фиксации текущего клипа в пикселах. Отсчет ведется от левого верхнего угла, если клип находится в основном фильме, и от точки фиксации, если он вложен во встроенный клип.

Впервые появилось во Flash 3.

MovieClip._ymouse

Свойство, возвращающее вертикальную координату курсора мыши относительно точки фиксации текущего клипа в пикселах. Доступно только для чтения.

Впервые появилось во Flash 5.

MovieClip._yscale

Свойство, задающее масштабирование текущего клипа по вертикали в процентах.

Впервые появилось во Flash 4.

NaN

Системная переменная, возвращающая значение NaN (Not a Number — не число). Это значение используется в некоторых случаях, например, для проверки, содержатся ли в числовом выражении ошибки. Доступна только для чтения.

Впервые появилась во Flash 5.

ne

Оператор сравнения "не равно". Формат использования:

<Аргумент 1> ne <Аргумент 2>

Возвращает true, если строковое представление Аргумента 1 не равно строковому представлению Аргумента 2, и false в противном случае.

Впервые появился во Flash 4. Во Flash 5 объявлен устаревшим и нерекомендованным к использованию; вместо него рекомендуется применять оператор !=.

new

Оператор создания нового экземпляра объекта. Формат использования:

<Переменная> = new <Конструктор объекта и список его параметров>;

Ссылка на созданный экземпляр объекта помещается в Переменную.

```
bool = new Boolean(false);
```

Впервые появился во Flash 5.

newline

Это действие существует только в виде пункта иерархического списка панели **Actions**. Применяется для того, чтобы поместить новое выражение в код ActionScript.

Впервые появилось во Flash 4.

nextFrame

Действие, перемещающее указатель на следующий кадр текущего клипа и останавливающее проигрывание фильма. Не принимает параметров.

Впервые появилось во Flash 2.

nextScene

Действие, перемещающее указатель на первый кадр следующей сцены текущего клипа и останавливающее проигрывание фильма. Не принимает параметров.

Впервые появилось во Flash 2.

not

Оператор логического отрицания (НЕ). Формат использования:

`not <Логическое выражение>`

Возвращает `true`, если значение аргумента равно `false`, и наоборот.

Впервые появился во Flash 4. Во Flash 5 объявлен устаревшим и нерекомендованным к использованию; вместо него рекомендуется применять оператор `!`.

null

Ключевое слово, обозначающее отсутствие любого значения у переменной или выражения.

Впервые появилось во Flash 2.

***Number* (функция)**

Функция, преобразующая аргумент в числовую величину. При этом она руководствуется следующими правилами:

- ☐ если аргумент — числовое выражение, возвращается его значение;
- ☐ если аргумент — логическое выражение, возвращается 1, если его значение равно `true`, и 0 в противном случае;
- ☐ если аргумент — строковое выражение, оно преобразуется в числовое;
- ☐ если аргумент равен `undefined`, возвращается 0.

Впервые появилась во Flash 4.

***Number* (объект)**

Объект, позволяющий манипулировать числовыми переменными как объектами.

Для создания экземпляра этого объекта используется конструктор:

`<Переменная> = new Number([<Числовое выражение>]);`

Если параметр конструктора пропущен, экземпляру объекта `Number` присваивается `false`.

```
num1 = new Number();  
num2 = new Number((a + 3) / (b - 1));
```

Кроме того, можно просто присвоить нужное значение переменной:

```
num2 = (a + 3) / (b - 1);
```

Впервые появился во Flash 5.

Number.MAX_VALUE

Свойство, возвращающее максимальное числовое значение, которое может быть помещено в переменную ActionScript (примерно $1,79 \times 10^{308}$). Доступно только для чтения.

Впервые появилось во Flash 5.

Number.MIN_VALUE

Свойство, возвращающее минимальное числовое значение, которое может быть помещено в переменную ActionScript (примерно 5×10^{-324}). Доступно только для чтения.

Впервые появилось во Flash 5.

Number.NaN

Свойство, возвращающее значение NaN (Not a Number — не число). Это значение используется в некоторых случаях, например, для проверки, содержатся ли в числовом выражении ошибки. Доступно только для чтения.

Впервые появилось во Flash 5.

Number.NEGATIVE_INFINITY

Свойство, возвращающее значение $-\infty$ ("минус бесконечность"). Доступно только для чтения.

Впервые появилось во Flash 5.

Number.POSITIVE_INFINITY

Свойство, возвращающее значение ∞ ("бесконечность"). Доступно только для чтения.

Впервые появилось во Flash 5.

Number.toString

Метод, возвращающий строковое представление значения числовой величины в заданной системе счисления. Формат использования:

```
<Числовая переменная>.toString([<Основание системы счисления>]);
```

Основание системы счисления может находиться в диапазоне между 2 и 36. Если оно не указано, число выводится в десятичной системе счисления. Впервые появился во Flash 5.

Number.valueOf

Метод, возвращающий значение числовой величины. Не принимает параметров.

Впервые появился во Flash 5.

Object

Объект, служащий основой для других, более сложных объектов. Предлагает минимальный набор свойств и методов.

Для создания экземпляра этого объекта используется конструктор:

```
<Переменная> = {[<Список свойств и их значений, разделенных запятыми>]};
```

Если ни одно свойство не было указано, создается "пустой" экземпляр объекта `Object`.

```
obj1 = {prop1: 1, prop2: 2, prop3: a + b};
```

```
obj2 = {};
```

```
obj2.prop1 = 1;
```

```
obj2.prop2 = 2;
```

```
obj2.prop3 = a + b;
```

Впервые появился во Flash 5.

Object.addProperty

Метод, добавляющий объекту новое динамическое свойство. Формат использования:

```
<Объект>.addProperty(<Имя>, <Функция чтения>, <Функция записи>);
```

Первым параметром передается имя создаваемого свойства в строковом виде. Вторым и третьим параметрами передаются ссылки на функции, вызываемые соответственно для чтения и записи значения свойства.

Функция чтения должна возвращать значение свойства. Она не принимает параметров.

Функция записи должна принимать единственный параметр — новое значение свойства. Она не возвращает значения.

Вы можете создавать свойства, доступные только для чтения или только для записи. В первом случае передайте значение `null` вместо функции записи, во втором — вместо функции чтения.

Не возвращает значения.

Впервые появился во Flash MX.

Object.__proto__

Свойство, возвращающее ссылку на родительский объект. Может использоваться для доступа к свойствам и методам родителя.

Впервые появилось во Flash 5.

Object.registerClass

Метод, привязывающий образец-клип к другому объекту, а не к `MovieClip`.
Формат использования:

```
Object.registerClass(<Имя образца-клипа>, <Конструктор объекта>);
```

Первым параметром в строковом виде задается имя образца-клипа. Если образец-клип с таким именем не присутствует в библиотеке, это имя "резервируется" на случай, если он все же появится. Вторым параметром передается ссылка на функцию-конструктор объекта, к которому вы хотите привязать образец.

Возвращает `true`, если привязка образца прошла успешно, и `false` в противном случае.

Этот метод часто используется для создания пользовательских компонентов.

```
#initclip
function wheelClass() {
    this._alpha = 100;
}
Object.registerClass("wheel", WheelClass);
#endinitclip
```

Приведенный выше код привязывает образец `wheel` к объекту `WheelClass`. После этого, если пользователь поместит на рабочий лист экземпляр этого образца, Flash создаст для него экземпляр объекта `WheelClass`, а не `MovieClip`.

Чтобы "отвязать" образец от объекта, вызовите этот метод еще раз, но вторым параметром передайте значение `null`.

```
Object.registerClass("wheel", null);
```

Впервые появился во Flash MX.

Object.toString

Метод, возвращающий строковое представление объекта. Вряд ли имеет какое-то практическое значение...

Впервые появился во Flash 5.

Object.unwatch

Метод, "отвязывающий" от свойства следящую функцию, привязанную ранее методом `watch`. Формат использования:

```
<Объект>.unwatch(<Имя свойства>);
```

Если "отвязка" прошла успешно, возвращается `true`, в противном случае — `false`.

Впервые появился во Flash MX.

Object.valueOf

Метод, возвращающий ссылку на сам объект. Не принимает параметров.

Впервые появился во Flash 5.

Object.watch

Метод, привязывающий к свойству следящую функцию. *Следящей* называется функция, вызываемая при любом изменении значения этого свойства. Формат использования:

```
<Объект>.watch(<Имя свойства>, <Функция>[, <Параметры>]);
```

Первым параметром передается имя свойства в строковом виде. Вторым параметром передается ссылка на следящую функцию. Третий параметр метода может задавать дополнительный параметр, передаваемый в следящую функцию.

Если привязка прошла успешно, возвращается `true`, в противном случае — `false`.

Следящая функция должна принимать четыре параметра: имя свойства, старое значение свойства, новое значение свойства и дополнительный параметр, задаваемый третьим параметром метода `watch`. Следящая функция не должна возвращать значение.

К одному и тому же свойству может быть привязана только одна следящая функция. Повторный вызов метода `watch` вызывает привязку к этому свойству другой следящей функции.

Впервые появился во Flash MX.

on

Действие, используемое для создания обработчиков событий кнопки и клипа-кнопки. Формат использования:

```
on(<Событие>) {  
    Обработчик  
}
```

Все доступные для кнопки события перечислены в табл. П1.2.

Таблица П1.2. События, принимаемые кнопкой

Событие	Описание
<code>press</code>	Наступает при нажатии левой кнопки мыши, когда ее курсор находится над текущей кнопкой Flash
<code>release</code>	Наступает при отпускании левой кнопки мыши, когда ее курсор находится над текущей кнопкой Flash
<code>releaseOutside</code>	Наступает при отпускании левой кнопки мыши, когда ее курсор находится вне текущей кнопки Flash (когда левая кнопка мыши была нажата, курсор мыши находился на текущей кнопке Flash)
<code>rollOut</code>	Наступает, когда курсор мыши "уходит" с кнопки
<code>rollOver</code>	Наступает, когда курсор мыши "заходит" на кнопку
<code>dragOut</code>	Наступает, когда курсор мыши "уходит" с кнопки Flash, причем левая кнопка мыши нажата
<code>dragOver</code>	Наступает, когда курсор мыши "заходит" на кнопку Flash, причем левая кнопка мыши нажата
<code>keyPress("<Код>")</code>	Наступает при нажатии клавиши с заданным в качестве параметра кодом

Впервые появилось во Flash 2. Полностью поддерживается, начиная с Flash 3.

onClipEvent

Действие, используемое для создания обработчиков событий клипа. Формат использования:

```
onClipEvent(<Событие>) {
    Обработчик
}
```

Все доступные для клипа события перечислены в табл. П1.3.

Таблица П1.3. События, принимаемые встроенным клипом

Событие	Описание
<code>data</code>	Наступает при приеме клипом внешних данных (если использовалось действие <code>loadVariables</code>) или загрузке очередного блока клипа (если использовалось действие <code>loadMovie</code>)

Таблица П1.3 (окончание)

Событие	Описание
enterFrame	Наступает при воспроизведении очередного кадра фильма. Предшествует всем другим событиям
keyDown	Наступает при нажатии любой клавиши на клавиатуре
keyUp	Наступает при отпускании любой клавиши на клавиатуре
load	Наступает сразу после загрузки встроенного клипа и его появления на экране
mouseDown	Наступает при нажатии левой кнопки мыши
mouseMove	Наступает при перемещении курсора мыши над встроенным клипом
mouseUp	Наступает при отпускании левой кнопки мыши
unload	Наступает после того, как клип пропадет с экрана, при воспроизведении следующего кадра (в котором уже нет этого клипа). Предшествует всем другим событиям этого кадра

Впервые появилось во Flash 5.

or

Оператор логического ИЛИ. Формат использования:

<Аргумент 1> or <Аргумент 2>

Возвращает true, если один из аргументов равен true, и false, если оба аргумента равны false.

Впервые появился во Flash 4. Во Flash 5 объявлен устаревшим и нерекомендованным к использованию; вместо него рекомендуется применять оператор ||.

ord

Функция, возвращающая код ASCII символа. Формат использования:

ord(<Символ>);

Впервые появилась во Flash 4. Во Flash 5 объявлена устаревшей и нерекомендованной к использованию; вместо нее рекомендуется применять методы объекта String.

parent

Свойство, возвращающее ссылку на внешний объект. Доступно только для чтения.

```
_parent._alpha = 50;
```

Может быть использовано несколько раз:

```
_parent._parent._parent._alpha = 50;
```

Впервые появилось во Flash 4.

parseFloat

Функция, преобразующая строку в числовое значение с плавающей запятой. Формат использования:

```
parseFloat(<Строковое выражение>);
```

Если строка не может быть преобразована в число, например, не начинается с цифры, возвращается NaN.

Впервые появилась во Flash 5.

parseInt

Функция, преобразующая строку в целочисленное значение в заданной системе счисления. Формат использования:

```
parseInt(<Строковое выражение>[, <Основание системы счисления>]);
```

Если второй параметр пропущен, возвращается число в десятичной системе счисления.

Если строка не может быть преобразована в число, например, не начинается с цифры, возвращается NaN.

Впервые появилась во Flash 5.

play

Действие, запускающее проигрывание текущего клипа. Не принимает параметров и не возвращает значения.

Впервые появилось во Flash 2.

prevFrame

Действие, перемещающее указатель на предыдущий кадр и останавливающее проигрывание фильма. Не принимает параметров и не возвращает значения.

Впервые появилось во Flash 2.

prevScene

Действие, перемещающее указатель на первый кадр предыдущей сцены текущего клипа и останавливающее проигрывание фильма. Не принимает параметров.

Впервые появилось во Flash 2.

print

Действие, печатающее клип на принтере в виде векторного изображения. Формат использования:

```
print(<Путь>[, <Область печати>]);
```

Первый параметр задает путь к нужному клипу в строковом виде. Второй параметр задает область печати фильма и может принимать три строковых значения:

- "bmovie" — кадр под именем #b задает область печати для кадров всего фильма (подробнее см. *главу 19*);
- "bmax" — кадр, чье содержимое имеет максимальные размеры, задает область печати для кадров всего фильма;
- "bframe" — каждый кадр печатается по-своему, из-за чего их размеры могут быть разными.

Как правило, изображение, печатаемое в векторном виде, выглядит лучше. Однако при этом могут быть потеряны различные цветовые эффекты, например, полупрозрачность.

Впервые появилось во Flash 4.20.

printAsBitmap

Действие, печатающее клип на принтере в виде растрового изображения. Формат использования:

```
printAsBitmap(<Путь>[, <Область печати>]);
```

Первый параметр задает путь к нужному клипу в строковом виде. Второй параметр задает область печати фильма и может принимать три строковых значения:

- "bmovie" — кадр под именем #b задает область печати для кадров всего фильма (подробнее см. *главу 19*);
- "bmax" — кадр, чье содержимое имеет максимальные размеры, задает область печати для кадров всего фильма;
- "bframe" — каждый кадр печатается по-своему, из-за чего их размеры могут быть разными.

Изображение, печатаемое в растровом виде, сохраняет все цветовые эффекты, в частности, полупрозрачность, однако выглядит хуже. Поэтому, если в изображении не используются цветовые эффекты, рекомендуется печатать его в векторном виде (см. действие `print`).

Впервые появилось во Flash 4.20.

printAsBitmapNum

Действие, печатающее клип на принтере в виде растрового изображения. Формат использования:

```
printAsBitmapNum(<Уровень>[, <Область печати>]);
```

Первый параметр задает уровень нужного клипа в числовом виде. Второй параметр задает область печати фильма и может принимать три строковых значения:

- ❑ `"bmovie"` — кадр под именем `#b` задает область печати для кадров всего фильма (подробнее см. главу 19);
- ❑ `"bmax"` — кадр, чье содержимое имеет максимальные размеры, задает область печати для кадров всего фильма;
- ❑ `"bframe"` — каждый кадр печатается по-своему, из-за чего их размеры могут быть разными.

Изображение, печатаемое в растровом виде, сохраняет все цветовые эффекты, в частности, полупрозрачность, однако выглядит хуже. Поэтому, если в изображении не используются цветовые эффекты, рекомендуется печатать его в векторном виде (см. действие `printNum`).

Впервые появилось во Flash 5.

printNum

Действие, печатающее клип на принтере в виде векторного изображения. Формат использования:

```
printNum(<Уровень>[, <Область печати>]);
```

Первый параметр задает уровень нужного клипа в числовом виде. Второй параметр задает область печати фильма и может принимать три строковых значения:

- ❑ `"bmovie"` — кадр под именем `#b` задает область печати для кадров всего фильма (подробнее см. главу 19);
- ❑ `"bmax"` — кадр, чье содержимое имеет максимальные размеры, задает область печати для кадров всего фильма;
- ❑ `"bframe"` — каждый кадр печатается по-своему, из-за чего их размеры могут быть разными.

Как правило, изображение, печатаемое в векторном виде, выглядит лучше. Однако при этом могут быть потеряны различные цветовые эффекты, например, полупрозрачность.

Впервые появилось во Flash 5.

_quality

Системная переменная, задающая качество отображаемой графики во всем фильме. Может принимать четыре строковых значения:

- ☐ "LOW" — низкое качество, ни векторная, ни растровая графика не сглаживается;
- ☐ "MEDIUM" — среднее качество, векторная графика сглаживается, растровая не сглаживается;
- ☐ "HIGH" — высокое качество, векторная графика сглаживается, если фильм не содержит анимации, растровая не сглаживается (значение по умолчанию);
- ☐ "BEST" — наилучшее качество, и векторная, и растровая графика сглаживается.

Используйте эту переменную, чтобы уменьшить количество системных ресурсов, потребляемых проигрывателем Flash. Это может понадобиться на медленных компьютерах.

Впервые появилась во Flash 5.

random

Функция, возвращающая целое псевдослучайное число от 0 и до значения переданного параметра минус 1. Формат использования:

```
random(<Верхняя граница>)
```

Впервые появилась во Flash 4. Во Flash 5 объявлена устаревшей и не рекомендованной к использованию; вместо нее рекомендуется применять метод `random` объекта `Math`.

removeMovieClip

Действие, удаляющее клип, созданный с помощью действия или метода `duplicateMovieClip` или метода `attachMovie`. Формат использования:

```
removeMovieClip(<Путь клипа>);
```

Впервые появилось во Flash 4.

return

Действие, возвращающее результат функции. Формат использования:

```
return [Выражение];
```

Если `Выражение` пропущено, возвращается `null`.

Может использоваться только в теле функции.

Впервые появилось во Flash 5.

_root

Свойство, возвращающее ссылку на корневой клип текущего уровня. Доступно только для чтения.

```
_root._alpha = 50;
```

Впервые появилось во Flash 4.

scroll

Свойство, возвращающее номер самой верхней строки, видимой в многострочном поле ввода или динамическом текстовом блоке, фактически — позицию вертикальной полосы прокрутки этого поля. Формат использования:

```
<Переменная, привязанная к текстовому полю>.scroll
```

Доступно только для чтения.

Впервые появилось во Flash 4. Во Flash 5 объявлено устаревшим и не рекомендованным к использованию; вместо него рекомендуется применять свойство `scroll` объекта `TextField`.

Selection

Объект, предоставляющий доступ к некоторым свойствам текста, находящегося в поле ввода, которое в данный момент имеет фокус.

Единственный экземпляр объекта `Selection` под тем же именем — `Selection` — создается самим Flash.

Впервые появился во Flash 5.

Selection.addListener

Метод, добавляющий объект-перехватчик событий текста. Формат использования:

```
Selection.addListener(<Объект-перехватчик>);
```

Единственным параметром этого метода передается ссылка на объект-перехватчик. Этот объект представляет собой экземпляр объекта `Object`, содержащий метод `onSetFocus`. Этот метод вызывается, когда какое-либо поле ввода получает фокус ввода.

Не возвращает значения.

Впервые появился во Flash MX.

Selection.getBeginIndex

Метод, возвращающий номер первого символа выделенного фрагмента текста. Если текст не выделен, или если в данный момент активно не поле ввода, возвращается -1. Не принимает параметров.

Впервые появился во Flash 5.

Selection.getCaretIndex

Метод, возвращающий номер символа, на котором стоит текстовый курсор. Если в данный момент активно не поле ввода, возвращается -1. Не принимает параметров.

Впервые появился во Flash 5.

Selection.getEndIndex

Метод, возвращающий номер последнего символа выделенного фрагмента текста. Если текст не выделен, или если в данный момент активно не поле ввода, возвращается -1. Не принимает параметров.

Впервые появился во Flash 5.

Selection.setFocus

Метод, возвращающий имя переменной или путь поля ввода или кнопки, имеющей в данный момент фокус ввода. Не принимает параметров.

Если поле ввода привязано к переменной, возвращается имя этой переменной в строковом виде. Если поле ввода не привязано к переменной, возвращается путь этого поля также в строковом виде. Если активна кнопка, возвращается ее путь.

Впервые появился во Flash 5. Пути полей ввода и кнопок возвращаются только во Flash MX.

Selection.onSetFocus

Обработчик события объекта-перехватчика событий текста, происходящего при перемещении фокуса ввода от одного поля ввода к другому. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события принимает два параметра: ссылки на поле ввода, потерявшее фокус, и на поле ввода, его получившее. Функция-обработчик не возвращает значения.

Впервые появился во Flash MX.

Selection.removeListener

Метод, удаляющий добавленный ранее объект-перехватчик. Формат использования:

```
Selection.removeListener(<Объект-перехватчик>)
```

Возвращает `true`, если объект-перехватчик был успешно удален, и `false` в противном случае.

Впервые появился во Flash MX.

Selection.setFocus

Метод, устанавливающий фокус ввода на поле ввода или кнопку. Формат использования:

```
Selection.setFocus(<Имя переменной>|<Путь>|<Объект>);
```

В качестве параметра может быть передано имя переменной, к которой привязано поле ввода, в строковом виде, путь к этому полю ввода в строковом же виде или ссылка на сам объект поля ввода.

Не возвращает значения.

Впервые появился во Flash 5. Пути полей ввода и кнопок возвращаются только во Flash MX.

Selection.setSelection

Метод, выделяющий фрагмент текста в поле ввода. Формат использования:

```
Selection.setSelection(<Первый символ>, <Последний символ>);
```

В качестве параметров передаются номера первого и последнего символа выделения.

Не возвращает значения.

Впервые появился во Flash 5.

set variable

Действие, присваивающее значение переменной. Формат использования:

```
set (<Переменная>, <Значение>);
```

Например:

```
set (Name, "vasya");
```

```
set (age, 25);
```

Впервые появилось во Flash 4.

setInterval

Действие, создающее таймер, вызывающий через заданный интервал времени нужную функцию или метод объекта. Форматы использования:

```
<Переменная>=setInterval(<Функция>, <Значение интервала времени>
```

```
⌘[, <Список параметров функции, разделенных запятыми>]);
```

```
<Переменная>=setInterval(<Экземпляр объекта>, <Метод объекта>,
```

```
⌘<Значение интервала времени>
```

```
⌘[, <Список параметров метода, разделенных запятыми>]);
```

Интервал времени задается в миллисекундах.

```
timerID1 = setInterval(tick, 1000);
```

```
timerID2 = setInterval(car, moveBy, 100, d);
```

Возвращает идентификатор интервала, который используется в дальнейшем для его удаления.

Впервые появилось во Flash MX.

setProperty

Действие, присваивающее значение свойству какого-либо объекта. Формат использования:

```
setProperty(<Путь объекта>, <Свойство>, <Значение>);
```

Путь объекта задается в строковом виде.

```
setProperty("_root.car", _alpha, 50);
```

Впервые появилось во Flash 4.

Sound

Объект, позволяющий управлять звуковым сопровождением какого-либо клипа.

Для создания экземпляра этого объекта используется конструктор:

```
<Переменная> = new Sound([<Клип>]);
```

В качестве единственного параметра конструктора этого объекта может быть задано имя клипа, звуковым сопровождением которого вы хотите управлять. Если параметр пропущен, полученный экземпляр будет управлять звуковым сопровождением всего фильма.

```
movieSound = new Sound();
```

```
carSound = new Sound(_root.car);
```

Впервые появился во Flash 5.

Sound.attachSound

Метод, создающий новый звук, основанный на сценарном образце-клипе, и привязывающий его к текущему экземпляру объекта `Sound`. Формат использования:

```
<Звук>.attachSound("<Имя разделяемого образца>");
```

Единственный параметр метода задает в строковом виде имя сценарного образца-клипа в библиотеке.

Не возвращает значения.

```
carSound.attachMovie("carDriving");
```

Впервые появился во Flash 5.

Sound.duration

Свойство, возвращающее продолжительность звука в миллисекундах. Доступно только для чтения.

Впервые появилось во Flash MX.

Sound.getBytesLoaded

Метод, возвращающий количество загруженных байтов звука. Не принимает параметров.

Впервые появился во Flash MX.

Sound.getBytesTotal

Метод, возвращающий общий размер загружаемых звуковых данных в байтах. Не принимает параметров.

Впервые появился во Flash MX.

Sound.getPan

Метод, возвращающий значение панорамирования звука в диапазоне от -100 (звук в левом канале) до 100 (звук в правом канале). Значение по умолчанию — 0 (звук в обоих каналах). Не принимает параметров.

Впервые появился во Flash 5.

Sound.getTransform

Метод, возвращающий экземпляр объекта `Object`, задающий трансформации звука. Не принимает параметров.

Впервые появился во Flash 5.

Sound.getVolume

Метод, возвращающий значение громкости звука в диапазоне от 0 (звук совсем не слышен) до 100 (полная громкость). Значение по умолчанию — 100. Не принимает параметров.

Впервые появился во Flash 5.

Sound.loadSound

Метод, загружающий внешний звуковой файл и привязывающий его к текущему экземпляру объекта *Sound*. Формат использования:

```
<Звук>.loadSound(<Интернет-адрес или путь звукового файла>,  
    ⚡<Потоковый звук>);
```

Первым параметром передается интернет-адрес звукового файла в строковом виде. Имейте в виду, что он должен находиться в том же домене, что и основной фильм. Второй параметр задает, станет ли загружаемый звук потоковым (значение *true*) или сигналом (значение *false*).

Не возвращает значения.

Впервые появился во Flash MX.

Sound.onLoad

Обработчик события, наступающего сразу после окончания загрузки звука, привязанного к текущему экземпляру объекта *Sound*. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Sound.onSoundComplete

Обработчик события, наступающего сразу после окончания воспроизведения звука, привязанного к текущему экземпляру объекта *Sound*. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Sound.position

Свойство, возвращающее продолжительность уже проигранной части звука в миллисекундах. Доступно только для чтения.

Впервые появилось во Flash MX.

Sound.setPan

Метод, задающий значение панорамирования звука. Формат использования:

```
<Звук>.setPan(<Панорамирование>);
```

Единственным параметром этого метода передается панорамирование. Оно может принимать значения в диапазоне от -100 (звук в левом канале) до 100 (звук в правом канале). Значение по умолчанию — 0 (звук в обоих каналах).

Не возвращает значения.

Впервые появился во Flash 5.

Sound.setTransform

Метод, задающий сложные трансформации звука. Формат использования:

```
<Звук>.setTransform(<Объект трансформаций>);
```

Единственным параметром этого метода передается ссылка на объект трансформаций. Это обычный экземпляр объекта `Object`, содержащий особые свойства, перечисленные в табл. П1.4.

Таблица П1.4. Свойства объекта трансформаций звука

Свойство	Описание
ll	Задаёт, какая часть "содержимого" левого канала будет проигрываться в левом канале в процентах, от 0 до 100
lr	Задаёт, какая часть "содержимого" левого канала будет проигрываться в правом канале в процентах, от 0 до 100
rl	Задаёт, какая часть "содержимого" правого канала будет проигрываться в левом канале в процентах, от 0 до 100
rr	Задаёт, какая часть "содержимого" правого канала будет проигрываться в правом канале в процентах, от 0 до 100

Не возвращает значения.

Впервые появился во Flash 5.

Sound.setVolume

Метод, задающий значение громкости звука. Формат использования:

```
<Звук>.setVolume(<Громкость>);
```

Единственным параметром этого метода передается громкость. Она может принимать значения в диапазоне от 0 (звук совсем не слышен) до 100 (полная громкость). Значение по умолчанию — 100.

Не возвращает значения.

Впервые появился во Flash 5.

Sound.start

Метод, запускающий проигрывание звука. Формат использования:

```
<Звук>.start([<Смещение>, <Количество повторений>]);
```

Первый параметр задает, с какой отметки будет проигран звук; отметка задается как количество секунд, которые Flash отсчитает с начала звука. Вторым параметром задает количество повторов звука. Если ни один параметр не задан, звук будет проигран один раз с начала.

Не возвращает значения.

Впервые появился во Flash 5.

Sound.stop

Метод, останавливающий проигрывание звука. Формат использования:

```
<Звук>.stop([<Имя звука>]);
```

В качестве единственного параметра этого метода может быть передано имя звука, который нужно остановить; оно передается в строковом виде. Если метод был вызван без параметра, останавливается проигрывание всех звуков.

Не возвращает значения.

Впервые появился во Flash 5.

_soundbuftime

Системная переменная, задающая размер буфера звука в секундах. Этот буфер используется при загрузке потоковых звуков для того, чтобы обеспечить их плавное воспроизведение.

Впервые появилась во Flash 4.

Stage

Объект, предоставляющий доступ к некоторым свойствам и методам окна проигрывателя Flash.

Единственный экземпляр объекта Stage под тем же именем — Stage — создается самим Flash.

Впервые появился во Flash MX.

Stage.addListener

Метод, добавляющий объект-перехватчик событий окна проигрывателя Flash. Формат использования:

```
Stage.addListener (<Объект-перехватчик>);
```

Единственным параметром этого метода передается ссылка на объект-перехватчик. Этот объект представляет собой экземпляр объекта `Object`, содержащий метод `onResize`. Данный метод вызывается при изменении размеров окна проигрывателя. Ему не передаются никакие параметры.

Не возвращает значения.

Впервые появился во Flash MX.

Stage.align

Свойство, задающее выравнивание воспроизводимого фильма в окне проигрывателя Flash. Может принимать значения, перечисленные в табл. П1.5.

Таблица П1.5. Значения, поддерживаемые свойством *align* объекта *Stage*

Значение	Выравнивание	
	по горизонтали	по вертикали
T	Верхнее	По центру
B	Нижнее	По центру
L	По центру	Левое
R	По центру	Правое
TL	Верхнее	Левое
TR	Верхнее	Правое
BL	Нижнее	Левое
BR	Нижнее	Правое

Впервые появилось во Flash MX.

Stage.height

Свойство, возвращающее высоту в пикселах окна проигрывателя Flash, если свойство `scaleMode` равно `"noScale"`, или высоту фильма, если свойство `scaleMode` не равно `"noScale"`. Доступно только для чтения.

Впервые появилось во Flash MX.

Stage.onResize

Обработчик события окна проигрывателя Flash, происходящего при изменении его размеров. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

Stage.removeListener

Метод, удаляющий добавленный ранее объект-перехватчик. Формат использования:

`Stage.removeListener(<Объект-перехватчик>)`

Возвращает `true`, если объект-перехватчик был успешно удален, и `false` в противном случае.

Впервые появился во Flash MX.

Stage.scaleMode

Свойство, задающее параметры масштабирования фильма, если он не помещается в окно проигрывателя. Это свойство может принимать следующие строковые значения:

- ❑ `"showAll"` — будет показано все изображение, для чего может быть применено масштабирование. Однако пропорции изображения искажены не будут; в результате этого вдоль горизонтальных или вертикальных сторон его могут появиться границы;
- ❑ `"noBorder"` — то же самое, что `"showAll"`, но границы появляться не будут — проигрыватель Flash обрежет изображение по горизонтали или вертикали, чтобы их избежать;
- ❑ `"exactFit"` — будет показано все изображение, для чего может быть применено масштабирование, в результате которого могут исказиться размеры изображения;
- ❑ `"noScale"` — изображение никогда не будет масштабироваться, в результате чего может оказаться обрезанным.

Впервые появилось во Flash MX.

Stage.width

Свойство, возвращающее ширину в пикселах окна проигрывателя Flash, если свойство `scaleMode` равно `"noScale"`, или ширину фильма, если свойство `scaleMode` не равно `"noScale"`. Доступно только для чтения.

Впервые появилось во Flash MX.

startDrag

Действие, разрешающее пользователю перемещать клип мышью. Формат использования:

```
startDrag(<Путь клипа>[, <За центр>, [<X1>, <Y1>, <X2>, <Y2>]]);
```

Первый (и единственный обязательный) параметр задает путь клипа, который пользователь будет иметь возможность перемещать мышью. Вторым параметром позволяет задать точку, к которой будет автоматически "приклеиваться" курсор мыши: точку, по которой пользователь щелкнул мышью (значение *false*) или центр клипа (значение *true*). Параметры *x1* и *y1* задают горизонтальную и вертикальную координаты левого верхнего угла воображаемого прямоугольника, внутри которого можно будет перемещать этот клип и за пределы которого он не сможет выйти. А за координаты правого нижнего угла "отвечают" параметры *x2* и *y2*. Эти координаты задаются относительно внешнего клипа в пикселах.

Не возвращает значения.

Впервые появилось во Flash 4.

stop

Действие, останавливающее проигрывание клипа. Не принимает параметров и не возвращает значения.

Впервые появилось во Flash 2.

stopAllSounds

Действие, останавливающее проигрывание всех звуков фильма, но не самого фильма. Не принимает параметров и не возвращает значения.

Впервые появилось во Flash 3.

stopDrag

Действие, запрещающее пользователю перетаскивать клип мышью. Вызывается после вызова действия или метода *startDrag*. Не принимает параметров и не возвращает значения.

Впервые появилось во Flash 4.

***String* (функция)**

Функция, преобразующая аргумент в строковую величину. При этом она руководствуется следующими правилами:

□ если аргумент — строковое выражение, возвращается его значение;

- ❑ если аргумент — логическое выражение, возвращается значение "true" или "false";
- ❑ если аргумент — числовое выражение, оно преобразуется в строку;
- ❑ если аргумент — объект, возвращается значение, возвращенное его методом `toString`;
- ❑ если аргумент — клип или кнопка, то возвращается его путь в формате "запись со слэшем";
- ❑ если аргумент равен `undefined`, возвращается пустая строка.

Впервые появилась во Flash 4.

***String* (объект)**

Объект, позволяющий манипулировать строковыми переменными как объектами.

Для создания экземпляра этого объекта используется конструктор:

```
<Переменная> = new String(<Строковое выражение>);
```

Примеры:

```
str = new Number("Flash MX");
```

Кроме того, можно просто присвоить нужное значение переменной:

```
str = "Flash MX";
```

Впервые появился во Flash 5.

String.charAt

Метод, возвращающий символ в заданной позиции строки. Формат использования:

```
<Строка>.charAt(<Позиция символа>);
```

Позиция символа может принимать значения от 0 (первый символ), до `<Длина строки>-1` (последний символ). Если позиция выходит за пределы этих значений, возвращается пустая строка.

Впервые появился во Flash 5.

String.charCodeAt

Метод, возвращающий код символа в заданной позиции строки в кодировке Unicode. Формат использования:

```
<Строка>.charCodeAt(<Позиция символа>);
```

Позиция символа может принимать значения от 0 (первый символ), до `<Длина строки>-1` (последний символ).

Впервые появился во Flash 5.

String.concat

Метод, объединяющий значение текущей строки со строками, переданными в качестве параметров, и возвращающий результат. Формат использования:

```
<Строка>.concat(<Список строк, разделенных запятыми>);
```

Текущее значение строки не изменяется.

Впервые появился во Flash 5.

String.fromCharCode

Метод, возвращающий строку, сформированную из символов, чьи коды ASCII были переданы в качестве параметров. Формат использования:

```
String.fromCharCode(<Список кодов символов, разделенных запятыми>);
```

Впервые появился во Flash 5.

String.indexOf

Метод, возвращающий позицию подстроки в текущей строке. Формат использования:

```
<Строка>.indexOf(<Подстрока>[, <Начальный символ>]);
```

Вторым параметром метода может быть передан номер символа, с которого начнется поиск подстроки. Если он не передан, поиск начинается с начала.

Впервые появился во Flash 5.

String.lastIndexOf

Метод, возвращающий позицию подстроки в текущей строке, причем поиск подстроки ведется с конца строки. Формат использования:

```
<Строка>.lastIndexOf(<Подстрока>[, <Начальный символ>]);
```

Вторым параметром метода может быть передан номер символа, с которого начнется поиск подстроки. Если он не передан, поиск начинается с конца.

Впервые появился во Flash 5.

String.length

Свойство, возвращающее длину строки в символах. Доступно только для чтения.

Впервые появилось во Flash 5.

String.slice

Метод, извлекающий подстроку из текущей строки и возвращающий ее в качестве значения. Сама текущая строка при этом не изменяется. Формат использования:

```
<Строка>.slice(<Начальный символ>[, <Конечный символ>]);
```

Первым параметром этого метода передается номер начального символа подстроки. Вторым параметром может быть передан номер конечного символа; если он пропущен, в подстроку попадут все символы до конца строки. Если второй параметр представляет собой отрицательное число, в подстроку также попадают все символы до конца строки.

Подстрока включает в себя первый символ, но не включает последний.

Впервые появился во Flash 5.

String.split

Метод, разбивающий текущую строку на подстроки, основываясь на переданном в параметрах символе-разделителе, и возвращающий массив полученных подстрок. Сама текущая строка при этом не изменяется. Формат использования:

```
<Строка>.split(<Символ-разделитель>[, <Лимит подстрок>]);
```

Первым параметром этого метода передается символ разделитель. Если передана пустая строка, то отдельной подстрокой станет каждый символ текущей строки. Вторым параметром может быть передано предельное количество возвращаемых подстрок. Если он пропущен, возвращаются все подстроки.

Впервые появился во Flash 5.

String.substr

Метод, извлекающий подстроку из текущей строки и возвращающий ее в качестве значения. Сама текущая строка при этом не изменяется. Формат использования:

```
<Строка>.substr(<Начальный символ>[, <Количество символов>]);
```

Первым параметром этого метода передается номер начального символа подстроки. Вторым параметром может быть передано количество символов, которые войдут в подстроку. Если оно пропущено, в подстроку попадут все символы до конца строки.

Впервые появился во Flash 5.

String.substring

Метод, извлекающий подстроку из текущей строки и возвращающий ее в качестве значения. Сама текущая строка при этом не изменяется. Формат использования:

```
<Строка>.substring(<Начальный символ>[, <Конечный символ>]);
```

Первым параметром этого метода передается номер начального символа подстроки. Вторым параметром может быть передан номер конечного символа; если он пропущен, в подстроку попадут все символы до конца строки. Если второй параметр представляет собой отрицательное число, в подстроку также попадают все символы до конца строки.

Подстрока включает в себя первый символ, но не включает последний.

Впервые появился во Flash 5.

String.toLowerCase

Метод, переводящий все символы текущей строки в нижний регистр и возвращающий результат. Сама текущая строка при этом не изменяется. Не принимает параметров.

Впервые появился во Flash 5.

String.toUpperCase

Метод, переводящий все символы текущей строки в верхний регистр и возвращающий результат. Сама текущая строка при этом не изменяется. Не принимает параметров.

Впервые появился во Flash 5.

substr

Функция, извлекающая подстроку из строки и возвращающая ее в качестве значения. Формат использования:

```
substr(<Строка>, <Начальный символ>, <Количество символов>);
```

Первым параметром этого метода передается сама строка. Вторым и третьим параметрами передаются соответственно номер начального символа подстроки и количество символов, которые войдут в подстроку. Если оно пропущено, в подстроку попадут все символы до конца строки.

Учтите, что в случае функции `substr` символы строки нумеруются, начиная с единицы, а не с нуля.

Впервые появилась во Flash 4. Во Flash 5 объявлена устаревшей и нерекондованной к использованию; вместо нее рекомендуется применять метод `substr` объекта `String`.

super

Оператор, используемый для вызова конструктора или метода родительского класса. Формат использования:

```
super.<Метод>([<Параметры>]);  
super([<Параметры>]);
```

Первый формат используется для вызова метода родительского класса, второй формат — для вызова конструктора родительского класса.

Впервые появился во Flash MX.

switch

Действие, применяемое для создания ветвлений. Формат использования:

```
switch (<Выражение>) {  
    Секции ветвления case и default  
}
```

Впервые появилось во Flash 4.

System

Объект, предоставляющий некоторые сведения о платформе клиента. В данной версии Flash содержит только внутренний объект `capabilities`.

Единственный экземпляр объекта `System` под тем же именем — `System` — создается самим Flash.

Впервые появился во Flash MX.

System.capabilities

Объект, предоставляющий некоторые сведения о платформе клиента.

Единственный экземпляр объекта `System.capabilities` под тем же именем — `System.capabilities` — создается самим Flash.

Впервые появился во Flash MX.

System.capabilities.hasAccessibility

Свойство, возвращающее `true`, если клиентский компьютер имеет средства обеспечения доступности, и `false` в противном случае. Доступно только для чтения.

Впервые появилось во Flash MX.

System.capabilities.hasAudio

Свойство, возвращающее `true`, если клиентский компьютер имеет средства воспроизведения звука, и `false` в противном случае. Доступно только для чтения.

Впервые появилось во Flash MX.

System.capabilities.hasAudioEncoder

Свойство, возвращающее ссылку на массив, установленный в системе аудиокодеков. Доступно только для чтения.

Впервые появилось во Flash MX.

System.capabilities.hasMP3

Свойство, возвращающее `true`, если клиентский компьютер имеет средства воспроизведения файлов MP3, и `false` в противном случае. Доступно только для чтения.

Впервые появилось во Flash MX.

System.capabilities.language

Свойство, возвращающее двухбуквенный строковый код языка клиентской операционной системы. Доступно только для чтения.

Все поддерживаемые Flash коды языков перечислены в табл. П1.6.

Таблица П1.6. Коды языков, поддерживаемые Flash

Язык	Код
Английский	en
Венгерский	hu
Голландский	nl
Датский	da
Испанский	es
Итальянский	it
Китайский	zh
Корейский	ko
Немецкий	de
Норвежский	no

Таблица П1.6 (окончание)

Язык	Код
Польский	pl
Португальский	pt
Русский	ru
Турецкий	tr
Финский	fi
Французский	fr
Чешский	cs
Шведский	sv
Японский	ja
Остальные языки или неизвестный язык	xu

Впервые появилось во Flash MX.

System.capabilities.manufacturer

Свойство, возвращающее имя производителя проигрывателя Flash. Может вернуть одно из трех строковых значений:

- ❑ "Macromedia Windows" — проигрыватель фирмы Macromedia для платформы Microsoft Windows;
- ❑ "Macromedia Macintosh" — проигрыватель фирмы Macromedia для платформы Apple Macintosh;
- ❑ "Macromedia Other OS" — проигрыватель фирмы Macromedia для других платформ.

Доступно только для чтения.

Впервые появилось во Flash MX.

System.capabilities.os

Свойство, возвращающее название клиентской операционной системы. Может вернуть одно из следующих строковых значений: "Windows XP", "Windows 2000", "Windows NT", "Windows 98/Me", "Windows 95", "Windows CE" и "MacOS". Доступно только для чтения.

Впервые появилось во Flash MX.

System.capabilities.pixelAspectRatio

Свойство, возвращающее частное от деления горизонтального размера пиксела на вертикальный. Доступно только для чтения.

Впервые появилось во Flash MX.

System.capabilities.screenColor

Свойство, возвращающее тип видеоподсистемы клиентского компьютера. Может вернуть одно из трех строковых значений: "color" (цветной), "gray" (черно-белый с градациями серого) и "bw" (черно-белый). Доступно только для чтения.

Впервые появилось во Flash MX.

System.capabilities.screenDPI

Свойство, возвращающее разрешающую способность экрана клиентского компьютера в пикселах на дюйм. Доступно только для чтения.

Впервые появилось во Flash MX.

System.capabilities.screenResolution.x

Свойство, возвращающее разрешение экрана клиентского компьютера по горизонтали в пикселах. Доступно только для чтения.

Впервые появилось во Flash MX.

System.capabilities.screenResolution.y

Свойство, возвращающее разрешение экрана клиентского компьютера по вертикали в пикселах. Доступно только для чтения.

Впервые появилось во Flash MX.

System.capabilities.version

Свойство, возвращающее версию проигрывателя Flash в виде целого числа. Доступно только для чтения.

Впервые появилось во Flash MX.

System.capabilities.hasVideoEncoder

Свойство, возвращающее ссылку на массив, установленный в системе видеокодеков. Доступно только для чтения.

Впервые появилось во Flash MX.

targetPath

Функция, возвращающая путь экземпляра текущего клипа в формате "запись с точкой". Формат использования:

```
targetPath(<Клип>)
```

Впервые появилась во Flash 5.

tellTarget

Действие, используемое для сокращения длины выражений доступа к свойствам и методам какого-либо объекта. Формат использования:

```
tellTarget("<Имя экземпляра объекта>") {  
    Выражения  
}
```

Имя экземпляра объекта задается в строковом виде.

```
tellTarget("someObject") {  
    prop1 = 1;  
    prop2 = 2;  
    prop3 = 3;  
    method1;  
}
```

Впервые появилось во Flash 3. Во Flash 5 объявлено устаревшим и не рекомендованным к использованию; вместо него рекомендуется применять действие `with`.

TextField

Объект, представляющий поле ввода или динамический текстовый блок.

Для каждого поля ввода или динамического текстового блока, помещенного на рабочий лист в среде Flash или из сценария, создается отдельный экземпляр объекта `TextField`. Этот экземпляр получает имя, заданное в редакторе свойств или одном из параметров метода `createTextField` объекта `MovieClip`.

Впервые появился во Flash MX.

TextField._alpha

Свойство, задающее прозрачность поля ввода или динамического текстового блока. Допускаются целые числовые значения от 0 (полная прозрачность) до 100 (полная непрозрачность). При этом поле остается активным, даже если оно совершенно невидимо (то есть, для свойства `_alpha` было задано значение 0).

Впервые появилось во Flash MX.

TextField.addListener

Метод, добавляющий объект-перехватчик событий поля ввода или динамического текстового блока. Формат использования:

```
<Поле>.addListener(<Объект-перехватчик>);
```

Единственным параметром этого метода передается ссылка на объект-перехватчик. Этот объект представляет собой экземпляр объекта `Object`, содержащий методы `onChanged` и `onScroller`.

Метод `onChanged` вызывается, когда пользователь изменяет содержимое поля ввода, а метод `onScroller` — когда пользователь прокручивает содержимое поля ввода или динамического текстового блока. Этим методам не передаются никакие параметры.

Не возвращает значения.

Впервые появился во Flash MX.

TextField.autoSize

Свойство, позволяющее сделать поле ввода "саморастягивающимся". Может принимать следующие строковые и логические значения:

- ☐ "none" или "false" — поле ввода не растягивается (значение по умолчанию);
- ☐ "left" или "true" — поле ввода растягивается вправо и вниз, а текст выравнивается по левому краю;
- ☐ "center" — поле ввода растягивается влево, вправо и вниз, а текст выравнивается по центру;
- ☐ "right" — поле ввода растягивается влево и вниз, а текст выравнивается по правому краю.

Впервые появилось во Flash MX.

TextField.background

Свойство, задающее, будет ли поле ввода иметь фон (значение `true`) или нет (значение `false`). Значение по умолчанию — `false`.

Впервые появилось во Flash MX.

TextField.backgroundColor

Свойство, задающее цвет фона поля ввода или динамического текстового блока. Значение по умолчанию — `0xFFFFFFFF` (белый цвет).

Учтите, что цветной фон будет виден только в том случае, если поле ввода или динамический текстовый блок имеет границу. В противном случае фон всегда будет белым.

Впервые появилось во Flash MX.

TextField.border

Свойство, задающее, будет ли поле ввода или динамический текстовый блок иметь границу (значение `true`) или нет (значение `false`).

Впервые появилось во Flash MX.

TextField.borderColor

Свойство, задающее цвет границы поля ввода или динамического текстового блока. Значение по умолчанию — `0x000000` (черный цвет).

Впервые появилось во Flash MX.

TextField.bottomScroll

Свойство, возвращающее номер самой нижней строки, видимой в поле ввода или динамическом текстовом блоке. Доступно только для чтения.

Строки текста нумеруются, начиная с единицы.

Впервые появилось во Flash MX.

TextField.embedFonts

Свойство, задающее, будут ли для вывода текста использоваться шрифты-псевдонимы (значение `false`) или встроенные шрифты (значение `true`).

Впервые появилось во Flash MX.

TextField._focusrect

Свойство, задающее, будет ли поле ввода иметь желтую рамку, если на нем находится фокус ввода. Если присвоено значение `true`, то поле ввода имеет такую рамку, если `false` — не имеет.

Впервые появилось во Flash MX.

TextField.getDepth

Метод, возвращающий уровень поля ввода или динамического текстового блока в виде целого числа. Не принимает параметров.

Впервые появился во Flash MX.

TextField.getFontList

Метод, возвращающий массив имен всех установленных в системе клиента шрифтов. Не принимает параметров.

Массив шрифтов включает также шрифты, внедренные в файл Shockwave/Flash, и разделяемые образцы-шрифты. Имена шрифтов выводятся в строковом виде.

Впервые появился во Flash MX.

TextField.getNewTextFormat

Метод, возвращающий ссылку на экземпляр объекта `TextFormat`, который содержит параметры форматирования вновь вводимого в текущее поле ввода текста. Не принимает параметров.

Впервые появился во Flash MX.

TextField.getTextFormat

Метод, возвращающий ссылку на экземпляр объекта `TextFormat`, который содержит параметры форматирования текста в текущем поле ввода или динамическом текстовом блоке. Форматы использования:

```
<Поле ввода>.getTextFormat();
```

```
<Поле ввода>.getTextFormat(<Номер символа>);
```

```
<Поле ввода>.getTextFormat(<Начальный символ>, <Конечный символ>);
```

Первый формат используется для получения форматирования всего текста в поле ввода. Второй формат позволит узнать форматирование заданного символа текста. Третий формат позволит узнать форматирование фрагмента текста, заданного номерами начального и конечного символов.

Впервые появился во Flash MX.

TextField._height

Свойство, задающее высоту поля ввода или динамического текстового блока в пикселях.

Впервые появилось во Flash MX.

TextField._highquality

Свойство, задающее качество сглаживания растровой графики во всем фильме. Может принимать три числовых значения:

- ☐ 2 — растровая графика сглаживается всегда;
- ☐ 1 — растровая графика сглаживается только тогда, когда фильм не содержит анимации;
- ☐ 0 — растровая графика никогда не сглаживается.

Используйте это свойство, чтобы уменьшить количество системных ресурсов, потребляемых проигрывателем Flash. Это может понадобиться на медленных компьютерах.

Очень странно, но, судя по всему, это свойство просто ссылается на системную переменную `_highquality`, выполняющую ту же функцию. Зачем нужно было создавать еще и свойство `_highquality`, неясно.

Впервые появилось во Flash MX.

TextField.hscroll

Свойство, задающее позицию указателя горизонтальной полосы прокрутки. Если поле ввода или динамический текстовый блок не прокручивается по горизонтали, то возвращается 0.

Впервые появилось во Flash MX.

TextField.html

Свойство, задающее, будет ли поле ввода или динамический текстовый блок обрабатывать текст в формате HTML. Если равно `true`, то текст HTML будет обрабатываться, если `false` — не будет.

Впервые появилось во Flash MX.

TextField.htmlText

Свойство, задающее содержимое поля ввода или динамического текстового блока в формате HTML. Имеет смысл только в том случае, если свойство `html` равно `true`. Если же свойство `html` равно `false`, свойство `htmlText` содержит то же значение, что и свойство `text`.

Впервые появилось во Flash MX.

TextField.length

Свойство, возвращающее длину текста в символах. Аналогично выражению `text.length`, но работает быстрее. Табуляция считается за один символ.

Доступно только для чтения.

Впервые появилось во Flash MX.

TextField.maxChars

Свойство, задающее максимальное количество символов, которое пользователь может ввести в поле ввода. Если его значение равно `null`, лимита на количество символов нет.

Впервые появилось во Flash MX.

TextField.maxhscroll

Свойство, возвращающее максимальное значение свойства `hscroll`. Доступно только для чтения.

Впервые появилось во Flash MX.

TextField.maxscroll

Свойство, возвращающее максимальное значение свойства `scroll`. Доступно только для чтения.

Впервые появилось во Flash MX.

TextField.multiline

Свойство, задающее, будет поле ввода или динамический текстовый блок многострочным (значение `true`) или однострочным (значение `false`).

Впервые появилось во Flash MX.

TextField._name

Свойство, возвращающее имя поля ввода или динамического текстового блока, заданное в редакторе свойств, в строковом виде. Доступно только для чтения.

Впервые появилось во Flash MX.

TextField.onChanged

Обработчик события, наступающего, когда пользователь изменяет содержимое поля ввода. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

TextField.onKillFocus

Обработчик события, наступающего, когда текущее поле ввода теряет фокус ввода. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события принимает единственный параметр, передающий ссылку на объект, получивший фокус ввода. Если ни один объект не получил фокус ввода, Flash присваивает этому параметру значение `null`. Функция-обработчик события не возвращает значения.

Впервые появился во Flash MX.

TextField.onScroller

Обработчик события, наступающего, когда пользователь прокручивает содержимое поля ввода или динамического текстового блока. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

TextField.onSetFocus

Обработчик события, наступающего, когда текущее поле ввода получает фокус ввода. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события принимает единственный параметр, передающий ссылку на объект, потерявший фокус ввода. Если ни один объект не имел ранее фокуса ввода, Flash присваивает этому параметру значение `null`. Функция-обработчик события не возвращает значения.

Впервые появился во Flash MX.

TextField._parent

Свойство, возвращающее ссылку на внешний клип. Доступно только для чтения.

```
clip = txtName._parent;
```

Может быть использовано несколько раз:

```
clip = txtName._parent._parent._parent;
```

Впервые появилось во Flash MX.

TextField.password

Свойство, позволяющее превратить обычное поле ввода в поле ввода пароля. Если равно `true`, то любой текст в поле ввода будет показываться как набор звездочек. Если равно `false`, то текст будет отображаться как обычно.

Впервые появилось во Flash MX.

TextField._quality

Свойство, задающее качество отображаемой графики во всем фильме. Может принимать четыре строковых значения:

- ☐ "LOW" — низкое качество, ни векторная, ни растровая графика не сглаживается;

- ❑ "MEDIUM" — среднее качество, векторная графика сглаживается, растровая не сглаживается;
- ❑ "HIGH" — высокое качество, векторная графика сглаживается, если фильм не содержит анимации, растровая не сглаживается (значение по умолчанию);
- ❑ "BEST" — наилучшее качество, и векторная, и растровая графика сглаживается.

Используйте это свойство, чтобы уменьшить количество системных ресурсов, потребляемых проигрывателем Flash. Это может понадобиться на медленных компьютерах.

Очень странно, но, судя по всему, это свойство просто ссылается на системную переменную `_quality`, выполняющую ту же функцию. Зачем нужно было создавать еще и свойство `_quality`, неясно.

Впервые появилось во Flash MX.

TextField.removeListener

Метод, удаляющий добавленный ранее объект-перехватчик. Формат использования:

```
<Поле ввода>.removeListener(<Объект-перехватчик>)
```

Возвращает `true`, если объект-перехватчик был успешно удален, и `false` в противном случае.

Впервые появился во Flash MX.

TextField.removeTextField

Метод, удаляющий поле ввода или динамический текстовый блок, созданный с помощью метода `createTextField` объекта `MovieClip`. Не принимает параметров и не возвращает значения.

Впервые появился во Flash MX.

TextField.replaceSel

Метод, заменяющий текст, выделенный в поле ввода, на другой. Формат использования:

```
<Поле ввода>.replaceSel(<Заменяющий текст>);
```

Заменяющий текст не должен иметь тегов HTML, т. к. этот метод их не обрабатывает.

Впервые появился во Flash MX.

TextField.restrict

Свойство, позволяющее задать набор символов, которые пользователь сможет вводить в поле ввода. (Сценарий может помещать в поле ввода любые символы, вне зависимости от значения этого свойства.)

Чтобы задать набор разрешенных символов, перечислите их в строковом значении этого свойства. Приведенное ниже выражение запрещает пользователю вводить большие латинские буквы А-Н.

```
txtName.restrict = "ABCDEFGH";
```

Вы можете использовать знак —, чтобы задать сразу диапазон символов.

```
txtName.restrict = "A-D E-H";
```

Чтобы задать набор запрещенных символов, предварите его значком ^. Приведенное ниже выражение запрещает пользователю вводить любые символы, кроме больших латинских букв А-Н.

```
txtName.restrict = "^ABCDEFGH";
```

Можно также комбинировать оба этих способа задания набора символов. Например, чтобы разрешить вводить символы А-Н, но запретить вводить Е, запишите

```
txtName.restrict = "A-H ^E";
```

Чтобы разрешить или запретить пользователю вводить символы — и ^, воспользуйтесь комбинациями \— и \^. Чтобы разрешить или запретить вводить \, воспользуйтесь комбинацией \\.

```
txtName.restrict = "^\\^\\-\\\\";
```

Вы также можете задавать нужные символы их кодами ASCII. Для этого воспользуйтесь комбинацией \u<Код символа>.

```
txtName.restrict = "\u0020";
```

Впервые появилось во Flash MX.

TextField._rotation

Свойство, задающее угол поворота поля ввода или динамического текстового блока в градусах.

Впервые появилось во Flash MX.

TextField.scroll

Свойство, задающее позицию указателя вертикальной полосы прокрутки. Если поле ввода или динамический текстовый блок не прокручивается по вертикали, возвращается 0.

Впервые появилось во Flash MX.

TextField.selectable

Свойство, задающее, может ли пользователь выделять текст в поле ввода или динамическом текстовом блоке (значение `true`) или нет (значение `false`).

Впервые появилось во Flash MX.

TextField.setNewTextFormat

Метод, задающий параметры форматирования вновь вводимого в текущее поле ввода текста. Формат использования:

```
<Поле ввода>.setNewTextFormat(<Экземпляр объекта TextFormat>);
```

Впервые появился во Flash MX.

TextField.setTextFormat

Метод, задающий параметры форматирования текста в текущем поле ввода или динамическом текстовом блоке. Форматы использования:

```
<Поле ввода>.setTextFormat(<Экземпляр объекта TextFormat>);
```

```
<Поле ввода>.setTextFormat(<Номер символа>,
```

```
    &lt;Экземпляр объекта TextFormat>);
```

```
<Поле ввода>.setTextFormat(<Начальный символ>, <Конечный символ>,
```

```
    &lt;Экземпляр объекта TextFormat>);
```

Первый формат используется для задания форматирования всего текста в поле ввода. Второй формат позволит задать форматирование заданного символа текста. Третий формат позволит задать форматирование фрагмента текста, заданного номерами начального и конечного символов.

Впервые появился во Flash MX.

TextField._soundbuftime

Свойство, задающее размер буфера звука в секундах. Этот буфер используется при загрузке потоковых звуков для того, чтобы обеспечить их плавное воспроизведение.

Это свойство просто ссылается на системную переменную `_soundbuftime`, выполняющую ту же функцию. Зачем нужно было создавать еще и свойство `_soundbuftime`, да еще и принадлежащее объекту, не имеющему никакого отношения к звуку, неясно.

Впервые появилось во Flash MX.

TextField.tabEnabled

Свойство, задающее, будет ли поле ввода включено в порядок обхода по нажатиям клавиш `<Tab>` и `<Shift>+<Tab>`.

Если значение этого свойства равно `true` или `undefined` или свойству `tabIndex` присвоено целое число, то поле ввода включается в порядок обхода. В противном случае оно исключается из порядка обхода, но все еще может быть выбрано щелчком мыши. Значение по умолчанию — `undefined`.

Впервые появилось во Flash MX.

TextField.tabIndex

Свойство, задающее место текущего поля ввода в порядке обхода. Может быть любым неотрицательным целым числом.

Если это свойство равно `undefined` и свойство `tabEnabled` не равно `false`, Flash сам формирует порядок обхода, зависящий от порядка, в котором кнопки, клипы и поля ввода были помещены на рабочий лист. Чтобы задать свой порядок обхода, присвойте всем элементам управления позиции в этом порядке, используя свойство `tabIndex`. Имейте в виду, что этот порядок не зависит от вложенности одних элементов управления в другие, т. е. он "плоский".

Значение по умолчанию — `undefined`.

Впервые появилось во Flash MX.

TextField._target

Свойство, возвращающее путь текущего поля ввода или динамического текстового блока. Доступно только для чтения.

Впервые появилось во Flash MX.

TextField.text

Свойство, задающее содержимое поля ввода или динамического текстового блока. Содержит обычный текст, без тегов HTML, даже если поле ввода или динамический текстовый блок может обрабатывать HTML.

Впервые появилось во Flash MX.

TextField.textColor

Свойство, задающее цвет текста в формате `0xRRGGBB`.

Впервые появилось во Flash MX.

TextField.textHeight

Свойство, возвращающее высоту текста в пикселах. Доступно только для чтения.

Впервые появилось во Flash MX.

TextField.textWidth

Свойство, возвращающее ширину текста в пикселах. Доступно только для чтения.

Впервые появилось во Flash MX.

TextField.type

Свойство, позволяющее выбрать между полем ввода и динамическим текстовым блоком. Может принимать два строковых значения: "input" (поле ввода) и "dynamic" (динамический текстовый блок). Значение по умолчанию — "input".

Впервые появилось во Flash MX.

TextField._url

Свойство, возвращающее интернет-адрес файла Shockwave/Flash, создавшего текущее поле ввода или динамический текстовый блок. Доступно только для чтения.

Впервые появилось во Flash MX.

TextField.variable

Свойство, задающее имя переменной, к которой привязано поле ввода или динамический текстовый блок, в строковом виде.

Впервые появилось во Flash MX.

TextField._visible

Свойство, позволяющее скрыть поле ввода или динамический текстовый блок. Если равно true, то поле ввода видимо и воспринимает ввод пользователя, если false — не видимо и не доступно. Значение по умолчанию — true.

Впервые появилось во Flash MX.

TextField._width

Свойство, задающее ширину поля ввода или динамического текстового блока в пикселах.

Впервые появилось во Flash MX.

TextField.wordWrap

Свойство, разрешающее (значение `true`) или запрещающее (значение `false`) полю ввода или динамическому текстовому блоку выполнять автоматический перенос строк.

Впервые появилось во Flash MX.

TextField._x

Свойство, задающее горизонтальную координату точки фиксации текущего поля ввода или динамического текстового блока в пикселах. Отсчет ведется от левого верхнего угла, если поле ввода находится в основном фильме, и от точки фиксации, если оно вложено во встроенный клип.

Впервые появилось во Flash MX.

TextField._xmouse

Свойство, возвращающее горизонтальную координату курсора мыши относительно верхнего левого угла текущего поля ввода или динамического текстового блока в пикселах. Доступно только для чтения.

Впервые появилось во Flash MX.

TextField._xscale

Свойство, задающее масштабирование текущего поля ввода или динамического текстового блока по горизонтали в процентах.

Впервые появилось во Flash MX.

TextField._y

Свойство, задающее вертикальную координату точки фиксации текущего поля ввода или динамического текстового блока в пикселах. Отсчет ведется от левого верхнего угла, если поле ввода находится в основном фильме, и от точки фиксации, если оно вложено во встроенный клип.

Впервые появилось во Flash MX.

TextField._ymouse

Свойство, возвращающее вертикальную координату курсора мыши относительно верхнего левого угла текущего поля ввода или динамического текстового блока в пикселах. Доступно только для чтения.

Впервые появилось во Flash MX.

TextField_yscale

Свойство, задающее масштабирование текущего поля ввода или динамического текстового блока по вертикали в процентах.

Впервые появилось во Flash MX.

TextFormat

Объект, позволяющий управлять форматированием текста, находящемся в каком-либо поле ввода или динамическом текстовом блоке.

Для создания экземпляра этого объекта используется конструктор:

```
<Переменная> = new TextFormat([<Шрифт>, [<Размер>, [<Цвет>,  
[<"Насыщенность">, [<Курсив>, [<Подчеркивание>, [<Интернет-адрес>,  
[<Цель>,  
[<Выравнивание>, [<Левая граница>, [<Правая граница>,  
[<Отступ красной строки>, [<Интервал>]]]]]]]]]]];
```

Ниже перечислены параметры конструктора:

- ☐ Шрифт — имя шрифта в строковом виде;
- ☐ Размер — размер шрифта в пунктах;
- ☐ Цвет — цвет текста в формате 0xRRGGBB;
- ☐ "Насыщенность" — true, чтобы сделать шрифт полужирным, и false, чтобы сделать его обычным;
- ☐ Курсив — true, чтобы сделать шрифт курсивным, и false, чтобы сделать его обычным;
- ☐ Подчеркивание — true, чтобы сделать текст подчеркнутым, и false, чтобы сделать его обычным;
- ☐ Интернет-адрес — интернет-адрес гиперссылки или пустая строка, если нужно сделать обычный текст, а не гиперссылку;
- ☐ Цель — цель гиперссылки в строковом виде;
- ☐ Выравнивание — "left" для выравнивания по левому краю, "center" для выравнивания по центру и "right" для выравнивания по правому краю;
- ☐ Левая граница — левая граница абзаца в пикселах;
- ☐ Правая граница — правая граница абзаца в пикселах;
- ☐ Отступ красной строки — отступ красной строки абзаца в пикселах;
- ☐ Интервал — количество интервалов между строками текста.

Впервые появился во Flash MX.

TextFormat.align

Свойство, задающее выравнивание текста. Может принимать три строковых значения:

- "left" — выравнивание по левому краю;
- "center" — выравнивание по центру;
- "right" — выравнивание по правому краю.

Значение по умолчанию — `null` (выравнивание не задано).

Впервые появилось во Flash MX.

TextFormat.blockIndent

Свойство, задающее отступ слева в пикселах для всех строк абзаца. Значение по умолчанию — `null` (отступ не задан).

Впервые появилось во Flash MX.

TextFormat.bold

Свойство, позволяющее сделать шрифт текста полужирным (значение `true`) или обычным (значение `false`). Значение по умолчанию — `null` (не задано).

Впервые появилось во Flash MX.

TextFormat.bullet

Свойство, позволяющее превратить текст в маркированный список (значение `true`) или сделать его обычным текстом (значение `false`). Значение по умолчанию — `null` (не задано).

Впервые появилось во Flash MX.

TextFormat.color

Свойство, задающее цвет текста в формате `0xRRGGBB`.

Впервые появилось во Flash MX.

TextFormat.font

Свойство, задающее имя шрифта текста в строковом виде. Значение по умолчанию — `null` (шрифт не задан).

Впервые появилось во Flash MX.

TextFormat.getTextExtent

Метод, возвращающий размеры строки текста, имеющего заданные параметры форматирования. Формат использования:

```
<Экземпляр объекта TextFormat>.getTextExtent(<Строка текста>);
```

Результат возвращается в виде экземпляра объекта `Object`, имеющего свойства `height` (высота строки) и `width` (ширина строки).

Текст, передаваемый методу в качестве параметра, считается одной строкой. Все символы возврата каретки и перевода строки игнорируются. Игнорируются также теги HTML.

Впервые появился во Flash MX.

TextFormat.indent

Свойство, задающее отступ красной строки абзаца в пикселах. Значение по умолчанию — `null` (отступ не задан).

Впервые появилось во Flash MX.

TextFormat.italic

Свойство, позволяющее сделать шрифт текста курсивом (значение `true`) или обычным (значение `false`). Значение по умолчанию — `null` (не задано).

Впервые появилось во Flash MX.

TextFormat.leading

Свойство, задающее расстояние между строками текста в интервалах. Значение по умолчанию — `null` (расстояние не задано).

Впервые появилось во Flash MX.

TextFormat.leftIndent

Свойство, задающее левую границу текста в пикселах. Значение по умолчанию — `null` (отступ не задан).

Впервые появилось во Flash MX.

TextFormat.rightIndent

Свойство, задающее правую границу текста в пикселах. Значение по умолчанию — `null` (отступ не задан).

Впервые появилось во Flash MX.

TextFormat.size

Свойство, задающее размер (кегель) шрифта в пунктах. Значение по умолчанию — `null` (размер не задан).

Впервые появилось во Flash MX.

TextFormat.tabStops

Свойство, определяющее значения позиций табуляции. Оно представляет собой массив числовых значений, заданных в пунктах. Значение по умолчанию — `null` (массив позиций табуляции не задан); в этом случае все позиции табуляции отстоят друг от друга на 4 пункта.

Впервые появилось во Flash MX.

TextFormat.target

Свойство, задающее цель гиперссылки. Это может быть имя фрейма, в который будет загружена страница, или одно из предопределенных значений:

- ☐ `_self` — страница загружается в текущий фрейм текущего окна (значение по умолчанию);
- ☐ `_blank` — страница загружается в новом окне;
- ☐ `_parent` — страница загружается во фрейм, являющийся внешним для текущего фрейма текущего окна, или в само это окно;
- ☐ `_top` — страница загружается в текущее окно, заполняя его целиком.

Имеет смысл только в том случае, когда задано значение свойства `url`.

Впервые появилось во Flash MX.

TextFormat.underline

Свойство, позволяющее сделать текст подчеркнутым (значение `true`) или не подчеркнутым (значение `false`). Значение по умолчанию — `null` (не задано).

Впервые появилось во Flash MX.

TextFormat.url

Свойство, задающее интернет-адрес. Используется для создания гиперссылок. Значение по умолчанию — `null` (не гиперссылка).

Впервые появилось во Flash MX.

this

Ключевое слово, возвращающее ссылку на текущий объект.

В сценарии, привязанном к кадру, возвращает ссылку на клип. В теле метода возвращает ссылку на экземпляр объекта, в котором выполняется этот метод. В обработчике события возвращает ссылку на внешний клип.

Впервые появилось во Flash 5.

toggleHighQuality

Действие, последовательно включающее и выключающее сглаживание графики всего фильма.

Сглаживание улучшает вид графики, но отнимает много системных ресурсов. Используйте это действие для ускорения проигрывания фильма на медленных компьютерах.

Впервые появилось во Flash 2.

trace

Действие, выводящее результат вычисления выражения в окно **Output**. Формат использования:

```
trace(<Выражение>);
```

Используйте это действие для целей отладки.

Впервые появилось во Flash 4.

true

Ключевое слово, обозначающее "истину" — одно из значений, которые может принимать логическая переменная.

Впервые появилось во Flash 5.

typeof

Оператор, возвращающий тип значения выражения. Формат использования:

```
typeof <Выражение>
```

Тип выражения возвращается в виде одного из строковых значений:

- "boolean" — логический тип;
- "function" — функция;
- "movieclip" — клип;
- "object" — объект, кнопка, поле ввода или динамический текстовый блок;

- ❑ "number" — числовой тип;
- ❑ "string" — строковый тип.

Впервые появился во Flash 5.

undefined

Ключевое слово, обозначающее, что переменной не было присвоено значение.

Впервые появилось во Flash 5.

unescape

Функция, принимающая в качестве параметра строку, закодированную для передачи методом GET, и возвращающая нормальную, раскодированную строку. Формат использования:

```
unescape(<Строковое выражение>);
```

Впервые появилась во Flash 5.

unloadMovie

Действие, удаляющее клип, загруженный с использованием действия и метода `loadMovie` и `attachMovie`. Формат использования:

```
unloadMovie(<Путь клипа>);
```

Не возвращает значения.

Впервые появилось во Flash 3.

unloadMovieNum

Действие, удаляющее клип, загруженный с использованием действия `loadMovieNum`. Формат использования:

```
unloadMovieNum(<Уровень клипа>);
```

Не возвращает значения.

Впервые появилось во Flash 3.

updateAfterEvent

Действие, принудительно обновляющее изображение на рабочем листе. Применяется внутри обработчиков событий и функций, вызываемых из таймеров; если же это действие встретится в другом месте кода, то будет проигнорировано. Не принимает параметров и не возвращает значения.

Впервые появилось во Flash 5.

var

Действие, служащее для объявления локальных переменных. Формат использования:

```
var <Имя переменной>=<Значение>[, <Имя переменной>=<Значение>...];
```

Впервые появилось во Flash 5.

void

Оператор, вычисляющий выражение и возвращающий значение `undefined`. Формат использования:

```
void (<Выражение>)
```

Впервые появился во Flash 5.

while

Действие, служащее для создания циклов с постусловием. Формат использования:

```
while(<Условие>) {  
    Тело цикла  
}
```

Тело цикла выполняется, пока `Условие` истинно.

Впервые появилось во Flash 4.

with

Действие, используемое для сокращения длины выражений доступа к свойствам и методам какого-либо объекта. Формат использования:

```
with(<Экземпляр объекта>) {  
    Выражения  
}
```

Пример:

```
with(someObject){  
    prop1 = 1;  
    prop2 = 2;  
    prop3 = 3;  
    method1;  
}
```

Впервые появилось во Flash 5.

XML

Объект, позволяющий работать с целыми документами и отдельными узлами документов XML.

Для создания экземпляра этого объекта используется конструктор:

```
<Переменная> = new XML([<Исходный текст>]);
```

В качестве единственного параметра конструктора этого объекта можно задать исходный текст создаваемого документа.

```
desc1 = new XML();
```

```
desc2 = new XML("<NAME>Flash MX</NAME><DESC>Приложение для создания Web-  
❧ графики и анимации</DESC>");
```

Впервые появился во Flash 5.

XML.appendChild

Метод, добавляющий к документу XML другой документ в качестве дочернего узла. Формат использования:

```
<Документ XML>.appendChild(<Узел XML>);
```

Не возвращает значения.

```
product = new XML("<PRODUCT></PRODUCT>");
```

```
desc = new XML("<NAME>Flash MX</NAME><DESC>Приложение для создания Web-  
❧ графики и анимации</DESC>");
```

```
product.appendChild(desc);
```

Впервые появился во Flash 5.

XML.attributes

Объект, представляющий собой массив атрибутов текущего узла XML. Формат использования:

```
<Документ XML>.attributes.<Атрибут>
```

Пример:

```
product.attributes.pid = "FL";
```

Впервые появился во Flash 5.

XML.childNodes

Объект, представляющий собой массив дочерних узлов текущего узла XML. Формат использования:

```
<Документ XML>.childNodes[<Номер узла>]
```

Пример:

```
product.childNodes[0].firstChild.nodeValue = "Flash 5";
```

Впервые появился во Flash 5.

XML.cloneNode

Метод, возвращающий точную копию текущего узла. Формат использования:

```
<Документ XML>.cloneNode(<Копировать дочерние узлы>)
```

Если значение единственного параметра этого метода равно `true`, то будут скопированы также все дочерние узлы текущего узла. Если оно равно `false`, то будет скопирован только текущий узел.

Впервые появился во Flash 5.

XML.contentType

Свойство, задающее тип кодирования передаваемых и принимаемых данных, так называемый тип *MIME* (Multipurpose Internet Mail Extensions — многоцелевые расширения почты Интернета). Значение по умолчанию — `application/x-www-form-urlencoded`.

Впервые появилось во Flash MX.

XML.createElement

Метод, создающий новый узел XML и возвращающий ссылку на него. Формат использования:

```
<Документ XML>.createElement(<Имя тега>);
```

В качестве параметра передается имя тега XML в строковом виде.

Возвращается экземпляр объекта XML, представляющий созданный узел. Изначально этот узел не имеет ни родителя, ни потомков.

```
prName = prod.createElement("NAME");
```

Впервые появился во Flash 5.

XML.createTextNode

Метод, создающий новый текстовый узел и возвращающий ссылку на него. Формат использования:

```
<Документ XML>.createTextNode(<Содержимое текстового узла>);
```

Возвращается экземпляр объекта XML, представляющий созданный узел. Изначально этот узел не имеет ни родителя, ни потомков.

```
prNameValue = prod.createTextNode("Flash MX");
```

Впервые появился во Flash 5.

XML.docTypeDecl

Свойство, задающее объявление типа документа XML, так называемое *DTD* (Document Type Declaration — объявление типа документа).

```
prod.docTypeDecl = "<!DOCTYPE greeting SYSTEM \"hello.dtd\">";
```

Впервые появилось во Flash 5.

XML.firstChild

Свойство, возвращающее ссылку на первый дочерний узел текущего узла XML. Доступно только для чтения.

Впервые появилось во Flash 5.

XML.getBytesLoaded

Метод, возвращающий количество загруженных байтов данных. Не принимает параметров.

Если операция загрузки данных не была запущена или реально еще не началась, возвращается *undefined*.

Впервые появился во Flash MX.

XML.getBytesTotal

Метод, возвращающий общий размер загружаемых данных в байтах. Не принимает параметров.

Если операция загрузки данных не была запущена или реально еще не началась, возвращается *undefined*.

Впервые появился во Flash MX.

XML.hasChildNodes

Свойство, возвращающее *true*, если текущий узел XML содержит дочерние узлы, и *false*, если не содержит. Доступно только для чтения.

Впервые появилось во Flash 5.

XML.ignoreWhite

Свойство, позволяющее исключить узлы, содержащие только пробелы. Если равно *true*, то узлы, содержащие только пробелы, будут исключены при обработке документа XML. Если равно *false*, то такие узлы останутся в документе. Значение по умолчанию — *false*.

Впервые появилось во Flash 5.

XML.insertBefore

Метод, вставляющий в документ XML новый узел. Формат использования:

```
<Документ XML>.insertBefore(<Вставляемый узел>,  
❏<Узел, перед которым он будет вставлен>);
```

Пример:

```
prName = prod.createElement("NAME");  
prDesc = prod.createElement("DESC");  
prod.appendChild(prDesc);  
prod.insertBefore(prName, prDesc);
```

Впервые появился во Flash 5.

XML.lastChild

Свойство, возвращающее ссылку на последний дочерний узел текущего узла XML. Доступно только для чтения.

Впервые появилось во Flash 5.

XML.load

Метод, загружающий в документ XML данные из текстового файла или принимающий их от серверного приложения. Формат использования:

```
<Документ XML>.load(<Интернет-адрес>);
```

В качестве единственного параметра этого метода принимается интернет-адрес серверного приложения или текстового файла.

После загрузки данные раскодируются, и формируются соответствующие узлы документа XML и их атрибуты.

Не возвращает значения.

Впервые появился во Flash 5.

XML.loaded

Свойство, позволяющее узнать, полностью ли загружены данные. Возвращает `true`, если данные загружены полностью, `false`, если не полностью, и `undefined`, если загрузка данных не была запущена или реально не началась.

Впервые появилось во Flash 5.

XML.nextSibling

Свойство, возвращающее ссылку на следующий по порядку узел XML той же степени вложенности, что и текущий. Если узлов той же степени вложенности больше нет, возвращается `null`. Доступно только для чтения.

Впервые появилось во Flash 5.

XML.nodeName

Свойство, задающее имя текущего узла (а именно, тег XML). Для текстовых узлов всегда равно `null`.

Впервые появилось во Flash 5.

XML.nodeType

Свойство, возвращающее тип текущего узла. Если это узел XML, возвращается 1, если текстовый — 3. Доступно только для чтения.

Впервые появилось во Flash 5.

XML.nodeValue

Свойство, задающее содержимое (текст) текстового узла. Для узлов XML оно всегда равно `null` и доступно только для чтения.

Впервые появилось во Flash 5.

XML.onData

Обработчик события, происходящего после успешной или неуспешной загрузки данных. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Изначально при наступлении этого события Flash автоматически обрабатывает полученный код XML и вызывает обработчик события `onLoad` (если, конечно, он задан). Вы можете создать собственный обработчик события `onData`, например, для особой обработки полученного кода XML.

Функция-обработчик события принимает единственный параметр — полученный код XML. Если во время приема произошла ошибка, этот параметр равен `undefined`. Функция-обработчик не возвращает значения.

Впервые появился во Flash 5.

XML.onLoad

Обработчик события, происходящего после успешной или неуспешной загрузки данных. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события принимает единственный параметр. Если он равен `true`, то данные были приняты нормально, если `false`, то во время их загрузки произошла ошибка. Функция-обработчик не возвращает значения.

Впервые появился во Flash 5.

XML.parentNode

Свойство, возвращающее ссылку на узел XML, являющийся родителем текущего узла. Если родителя у узла нет, возвращается `null`. Доступно только для чтения.

Впервые появилось во Flash 5.

XML.parseXML

Метод, обрабатывающий переданный в качестве параметра код XML и заполняющий полученными данными текущий экземпляр объекта XML. Все содержимое текущего документа XML будет уничтожено. Формат использования:

```
<Документ XML>.parseXML(<Код XML>):
```

Не возвращает значения.

Впервые появился во Flash 5.

XML.previousSibling

Свойство, возвращающее ссылку на предыдущий по порядку узел XML той же степени вложенности, что и текущий. Если узлов той же степени вложенности больше нет, возвращается `null`. Доступно только для чтения.

Впервые появилось во Flash 5.

XML.removeNode

Метод, удаляющий текущий узел из документа. Не принимает параметров и не возвращает значения.

```
prod.childNodes[1].removeNode();
```

Впервые появился во Flash 5.

XML.send

Метод, отправляющий документ XML серверному приложению. Формат использования:

```
<Документ XML>.send("<Интернет-адрес>",[,"<Цель>"]);
```

Первым параметром передается интернет-адрес серверного приложения, которое получит эти данные. Вторым параметром передается имя фрейма, в который будет загружена сформированная этим приложением страница, или одно из предопределенных значений:

- `_self` — страница загружается в текущий фрейм текущего окна (значение по умолчанию);
- `_blank` — страница загружается в новом окне;
- `_parent` — страница загружается во фрейм, являющийся внешним для текущего фрейма текущего окна, или в само это окно;
- `_top` — страница загружается в текущее окно, заполняя его целиком.

Данные всегда передаются методом `POST`.

Впервые появился во Flash 5.

XML.sendAndLoad

Метод, отправляющий документ XML серверному приложению и тут же принимающий от него результат. Формат использования:

```
<Документ XML>.sendAndLoad("<Интернет-адрес>",
```

```
❧ <Документ XML, принимающий данные>);
```

Первым параметром опять же передается интернет-адрес серверного приложения, которое получит эти данные. Вторым параметром передается ссылка на экземпляр объекта XML, который получит результат.

Данные всегда передаются методом `POST`.

Впервые появился во Flash 5.

XML.status

Свойство, служащее для задания и проверки кода ошибки обработки документа XML. Может принимать следующие числовые значения:

- 0 — документ был успешно обработан, ошибок не возникло;
- -2 — ошибка в секции `CDATA`;
- -3 — ошибка в объявлении XML;
- -4 — ошибка в объявлении `DOCTYPE`;
- -5 — ошибка в комментарии;
- -6 — документ XML некорректно отформатирован;
- -7 — не хватает памяти;
- -8 — ошибка в значении атрибута;

- -9 — начальный тег не совпадает с конечным;
- -10 — найден конечный тег без парного ему начального тега.

Впервые появилось во Flash 5.

XML.toString

Метод, возвращающий код XML, содержащийся в текущем документе, в строковом виде. Не принимает параметров.

Впервые появился во Flash 5.

XML.xmlDecl

Свойство, задающее объявление XML.

```
prod.xmlDecl = "<?xml version=\"1.0\" ?>";
```

Впервые появилось во Flash 5.

XMLSocket

Объект, позволяющий открывать постоянные соединения между приложением Flash и серверной программой и передавать по ним данные в формате XML.

Для создания экземпляра этого объекта используется конструктор:

```
<Переменная> = new XMLSocket();
```

Впервые появился во Flash 5.

XMLSocket.close

Метод, закрывающий текущее постоянное соединение. Не принимает параметров и не возвращает значения.

Впервые появился во Flash 5.

XMLSocket.connect

Метод, устанавливающий постоянное соединение с серверной программой. Формат использования:

```
<Экземпляр объекта XMLSocket>.connect(<Интернет-адрес>, <Порт>);
```

Первым параметром в строковом виде передается интернет-адрес — внимание! — сервера, на котором работает серверное приложение. Вы также можете передать в качестве параметра `null` для соединения с сервером, откуда был загружен файл с приложением Flash. Вторым параметром передается номер порта, по которому будут передаваться данные. Для создания посто-

янных соединений можно использовать только порты с номерами 1024 и выше.

Возвращает `true`, если создание постоянного соединения успешно начато. В противном случае возвращается `false`.

Впервые появился во Flash 5.

XMLSocket.onClose

Обработчик события, наступающего после закрытия текущего постоянного соединения серверным приложением (не приложением Flash). Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события не принимает параметров и не возвращает значения.

Впервые появился во Flash 5.

XMLSocket.onConnect

Обработчик события, происходящего после успешного или неуспешного установления постоянного соединения с серверным приложением. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события принимает единственный параметр. Если он равен `true`, то соединение было установлено нормально, если `false`, то во время установления соединения произошла ошибка. Функция-обработчик не возвращает значения.

Впервые появился во Flash 5.

XMLSocket.onData

Обработчик события, происходящего после успешной или неуспешной загрузки данных. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Изначально при наступлении этого события Flash автоматически обрабатывает полученный код XML и вызывает обработчик события `onXML` (если, конечно, он задан). Вы можете создать собственный обработчик события `onData`, например, для особой обработки полученного кода XML.

Функция-обработчик события принимает единственный параметр — полученный код XML. Если во время приема произошла ошибка, этот параметр равен `undefined`. Функция-обработчик не возвращает значения.

Впервые появился во Flash 5.

XMLSocket.onXML

Обработчик события, происходящего после успешной или неуспешной загрузки данных. Представляет собой свойство, которому нужно присвоить функцию, обрабатывающую это событие.

Функция-обработчик события принимает единственный параметр — ссылку на экземпляр объекта XML, содержащий принятые данные. Функция-обработчик не возвращает значения.

Впервые появился во Flash 5.

XMLSocket.send

Метод, передающий данные серверной программе по постоянному соединению. Формат использования:

```
<Экземпляр объекта XMLSocket>.send(<Экземпляр объекта XML>);
```

Единственным параметром этого метода передается ссылка на экземпляр объекта XML, содержащий передаваемые данные.

Не возвращает значения.

Впервые появился во Flash 5.

Коды символов ASCII

Все коды символов ASCII, имеющих на клавиатуре, приведены в табл. П1.7. Коды остальных символов вы можете узнать, запустив утилиту Таблица символов, поставляющуюся в составе Windows.

Таблица П1.7. Коды символов ASCII

Символ	Код	Символ	Код	Символ	Код	Символ	Код
Пробел	32	*	42	4	52	>	62
!	33	+	43	5	53	?	63
"	34	,	44	6	54	@	64
#	35	-	45	7	55	A	65
\$	36	.	46	8	56	B	66
%	37	/	47	9	57	C	67
&	38	0	48	:	58	D	68
'	39	1	49	;	59	E	69
(40	2	50	<	60	F	70
)	41	3	51	=	61	G	71

Таблица П1.7 (окончание)

Символ	Код	Символ	Код	Символ	Код	Символ	Код
H	72	g	103	Д	196	в	226
I	73	h	104	Е	197	г	227
J	74	i	105	Ж	198	д	228
K	75	j	106	З	199	е	229
L	76	k	107	И	200	ж	230
M	77	l	108	Й	201	з	231
N	78	m	109	К	202	и	232
O	79	n	110	Л	203	й	233
P	80	o	111	М	204	к	234
Q	81	p	112	Н	205	л	235
R	82	q	113	О	206	м	236
S	83	r	114	П	207	н	237
T	84	s	115	Р	208	о	238
U	85	t	116	С	209	п	239
V	86	u	117	Т	210	р	240
W	87	v	118	У	211	с	241
X	88	w	119	Ф	212	т	242
Y	89	x	120	Х	213	у	243
Z	90	y	121	Ц	214	ф	244
[91	z	122	Ч	215	х	245
\	92	{	123	Ш	216	ц	246
]	93		124	Щ	217	ч	247
^	94	}	125	Ъ	218	ш	248
_	95	~	126	Ы	219	щ	249
'	96	Е	168	Ь	220	ъ	250
a	97	ё	184	Э	221	ы	251
b	98	А	192	Ю	222	ь	252
c	99	Б	193	Я	223	э	253
d	100	В	194	а	224	ю	254
e	101	Г	195	б	225	я	255
f	102						

Виртуальные коды клавиш

В табл. П1.8 перечислены виртуальные коды всех клавиш клавиатуры.

Таблица П1.8. Виртуальные коды клавиш

Клавиша	Код	Клавиша	Код	Клавиша	Код	Клавиша	Код
Backspace	8	6	54	V	86	F3	114
Tab	9	7	55	W	87	F4	115
Enter	13	8	56	X	88	F5	116
Shift	16	9	57	Y	89	F6	117
Ctrl	17	A	65	Z	90	F7	118
Alt	18	B	66	Windows	91	F8	119
CapsLock	20	C	67	Context menu	93	F9	120
Esc	27	D	68	NumPad* 0	96	F10	121
Пробел	32	E	69	NumPad 1	97	F11	122
PgUp	33	F	70	NumPad 2	98	F12	123
PgDn	34	G	71	NumPad 3	99	F13	124
End	35	H	72	NumPad 4	100	F14	125
Home	36	I	73	NumPad 5	101	F15	126
←	37	J	74	NumPad 6	102	NumLock	144
↑	38	K	75	NumPad 7	103	::	186
→	39	L	76	NumPad 8	104	=+	187
↓	40	M	77	NumPad 9	105	-_	189
Ins	45	N	78	NumPad *	106	/?	191
Del	46	O	79	NumPad +	107	'~	192
0	48	P	80	NumPad Enter	108	[{	219
1	49	Q	81	NumPad -	109	\	220
2	50	R	82	NumPad .	110	}]	221
3	51	S	83	NumPad /	111	"	222
4	52	T	84	F1	112		
5	53	U	85	F2	113		

* Так обозначаются клавиши, расположенные на дополнительной цифровой клавиатуре.

Сообщения об ошибках

В табл. П1.9 перечислены все сообщения об ошибках, выдаваемые Flash, и их разъяснение.

Таблица П1.9. Сообщения об ошибках

Сообщение	Разъяснение
<Ключевое слово> Expected	Должно быть Ключевое слово
Case statements can only be used inside a switch statement	Действие case может быть использовано только в выражении ветвления
Case statements must be terminated with a ':'	Действие case должно завершаться знаком двоеточия
Case-insensitive identifier <Идентификатор> will obscure built-in object <Имя встроенного объекта>	Идентификатор совпал с Именем встроенного объекта
Clip events are permitted only for movie clip instances	Обработчики событий клипа можно привязывать только к клипам
Duplicate in event list	Встретился еще один обработчик того же события, что недопустимо
'else' encountered without matching 'if'	Обнаружено else без соответствующего ему if
Error opening include file:file not found	Ошибка открытия внешнего файла со сценариями, включенного в код с помощью действия #include. Файл не найден
Event expected	Должно быть событие
Expected a field name after '.' operator	Неправильная запись доступа к свойству или методу объекта
Function declaration not permitted here	В этом случае задавать имя функции недопустимо
Function name expected	Требуется имя функции
Hexadecimal digits expected after 0x	Шестнадцатеричное число должно начинаться с 0x
Identifier expected	Должно быть имя переменной или свойства
Initializer list must be terminated by <Закрывающий символ>	Пропущен Закрывающий символ, например, квадратная или фигурная скобка

Таблица П1.9 (продолжение)

Сообщение	Разъяснение
Internal error	Внутренняя ошибка Flash. Разработчики фирмы Macromedia предлагают при возникновении этой ошибки послать им файл документа с подробным описанием условий, приведших к возникновению этой ошибки
Invalid key code	Неверный или неподдерживаемый код клавиши
Invalid mouse event specified	Неверное или неподдерживаемое событие мыши
Invalid movie clip event specified	Неверное событие для обработчика onClipEvent
Key code identifier expected	Должен быть код клавиши
Left side of assignment operator must be variable or property	Слева от оператора присваивания должно быть имя переменной или свойства
Malformed #include directive	Ошибка в написании действия #include
Mouse events are permitted only for button instances	Обработчики событий кнопки можно привязывать только к кнопкам
Multi-line comment was not terminated	Найден символ конца комментария */, но символ начала комментария /* не найден
'on' handlers may not nest within other 'on' handlers	Один обработчик события был вложен внутрь другого, что недопустимо
'onClipEvent' handlers may not nest within other 'onClipEvent' handlers	Один обработчик события был вложен внутрь другого, что недопустимо
Operator <Оператор> must be followed by an operand	Оператор без операнда
Parameter <Параметр> cannot be declared multiple times	Параметр встретился в объявлении функции несколько раз
Parameter name expected	Требуется имя параметра (внутри объявления функции)
Property <Свойство> does not exist	Свойство у данного объекта не существует
Property name expected in getProperty	Требуется имя свойства
Scene name must be a quoted string	Имя сцены должно быть строковой величиной, а значит, помещаться в кавычках
Statement block must be terminated by '}'	Пропущена закрывающая фигурная скобка

Таблица П1.9 (окончание)

Сообщение	Разъяснение
Statement must appear within on handler	Это выражение допустимо только в теле обработчика события
Statement must appear within on or onClipEvent handler	Это выражение допустимо только в теле обработчика события
Statement must appear within onClipEvent handler	Это выражение допустимо только в теле обработчика события
String literal was not properly terminated	Не хватает одной из кавычек строки
Syntax error	Синтаксическая ошибка
The JavaScript '<Выражение>' construct is not supported	Выражение языка JavaScript не поддерживается в ActionScript
Trailing garbage found	В конце выражения проставлены ненужные символы (хотя само выражение правильное)
Variable <Переменная> cannot be declared multiple times	Переменная была объявлена несколько раз
Wrong number of parameters; <Функция> requires between <От> and <До>	Неверное количество параметров Функции

"Запись со слэшем" и "Запись с точкой"

Осталось поговорить о так называемой "записи со слэшем". Этот способ записи выражений доступа к свойствам и методам объектов применялся в старых версиях Flash — 3 и 4. Начиная с Flash 5, его сменила привычная "запись с точкой".

Что же такое "запись со слэшем"? Давайте выясним, сравнив ее с "записью с точкой".

Предположим, нам нужно изменить прозрачность одного из колес автомобиля. Для этого мы используем выражение

```
car.wheel2._alpha = 50;
```

Это "запись с точкой". В ней каждый элемент пути отделяется точкой.

В "записи со слэшем" каждый элемент пути, кроме свойства или метода, отделяется друг от друга знаком косой черты (слэша). Свойство или метод отделяется двоеточием. Запомните это.

```
car/wheel2:_alpha = 50;
```

Некоторые действия, функции и методы в двух последних версиях Flash (5 и MX) возвращают путь объекта в "записи со слэшем". Чтобы перевести его в "запись с точкой", используйте функцию `eval`.

Приложение 2

Внедрение фильмов Flash в Web-страницы

Flash не зря называют пакетом Web-графики и анимации. Большинство фильмов и приложений, созданных во Flash, распространяются именно по Всемирной Сети. Это всевозможные анимационные ролики, баннеры, элементы оформления, игрушки и пр. Существуют даже целые Web-сайты, построенные на Flash. А большинство современных программ Web-обозревателей поставляются с проигрывателями Flash. А как же — стандарт де-факто!

Можно сказать, что графика в WWW после появления Flash переживает свое второе рождение. В самом деле, если фильм AVI или QuickTime занимает несколько мегабайт, то Flash-ролик "весит" на порядок меньше. А значит, его проще выложить в Интернет для всеобщего доступа. И прославиться...

В этом приложении мы поговорим о тщеславии и способах помещения графики Flash на Web-страницы.

Как графика Flash помещается на Web-страницы

Действительно, как это делается?

При всех своих неоспоримых достоинствах, язык HTML имеет одно существенное ограничение. Он может описывать только текстовые данные. (За очень небольшим исключением: горизонтальные линии, таблицы и т. п.) Это означает, что с его помощью можно отформатировать некий текст, разбить его на абзацы, оформить заголовки, раскрасить разными цветами, задать отступы — но не более. Пользуясь только HTML, вы не можете создать на странице рисунок. Для этого вам придется рисовать его отдельно и оформлять в виде внедренного элемента.

Внедренные элементы всегда хранятся в отдельных файлах и помещаются на Web-страницу с помощью особых тегов HTML. Когда Web-обозреватель встретит такой тег, он извлекает из его *атрибутов* имя файла, где хранится

внедренный элемент, загружает его и выводит на экран в нужном месте страницы.

Если вы знакомы с языком HTML, то, наверняка, знаете, что для помещения рисунков на Web-страницу служит тег ``.

```
<IMG SRC="logo.jpg">
```

Атрибут SRC этого тега задает имя файла.

Фильмы Flash, хранящиеся в файлах Shockwave/Flash, являются такими же внедренными элементами, как растровое изображение JPEG, которое мы только что поместили на Web-страницу с помощью тега ``. Чтобы поместить на страницу фильм Flash, используются теги `<OBJECT>` и `<EMBED>`. Сейчас мы их рассмотрим.

Теги `<OBJECT>` и `<EMBED>`

Здесь пойдет рассказ об этих тегах и их атрибутах.

Тег `<OBJECT>`

Парный тег `<OBJECT>` в общем случае служит для помещения на Web-страницу компонентов ActiveX. В том числе, с его помощью можно поместить на Web-страницу проигрыватель Flash, оформленный в виде компонента ActiveX, и загрузить в него нужный фильм.

Пример использования тега `<OBJECT>`:

```
<OBJECT CLASSID="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" WIDTH="100"  
  ⚡HEIGHT="100" CODEBASE="http://active.macromedia.com/flash6/cabs/  
  ⚡swflash.cab#version=6,0,0,0">
```

```
<PARAM NAME="MOVIE" VALUE="movie.swf">
```

```
<PARAM NAME="PLAY" VALUE="true">
```

```
<PARAM NAME="LOOP" VALUE="true">
```

```
<PARAM NAME="QUALITY" VALUE="high">
```

```
</OBJECT>
```

Тег `<OBJECT>` содержит довольно много атрибутов. С их помощью задаются, в частности, размеры внедренного компонента ActiveX на Web-странице (атрибуты `WIDTH` и `HEIGHT`) и интернет-адрес его дистрибутива (атрибут `CODEBASE`). Таким образом, если нужный компонент на клиентском компьютере не установлен, Web-обозреватель может самостоятельно его загрузить и установить.

Как видите, внутри тега `<OBJECT>` помещается набор тегов `<PARAM>`, с помощью которых задаются различные параметры компонента ActiveX. Для каждого компонента этот набор параметров свой. В случае проигрывателя Flash мы можем задать имя загружаемого файла Shockwave/Flash (параметр "MOVIE") и качество графики (параметр "QUALITY"). Имя параметра задается атрибутом NAME тега `<PARAM>`, а значение — атрибутом VALUE.

Тег `<OBJECT>` начал поддерживаться Web-обозревателями Microsoft Internet Explorer версии 3.0 и Netscape Navigator 4.0. Более старыми версиями этих программ он не поддерживается.

Тег `<OBJECT>` стандартизирован и рекомендован к использованию комитетом WWWW.

Тег `<EMBED>`

Парный тег `<EMBED>` в общем случае служит для помещения на Web-страницу модулей расширения Web-обозревателя. В том числе, с его помощью можно поместить на Web-страницу проигрыватель Flash, оформленный в виде модуля расширения.

Пример использования тега `<EMBED>`:

```
<EMBED SRC="movie.swf" WIDTH="100" HEIGHT="100" PLAY="true" LOOP="true"  
QUALITY="high"  
PLUGINSPAGE="http://www.macromedia.com/shockwave/download/index.cgi?  
P1_Prod_Version=ShockwaveFlash"  
>
```

В отличие от тега `<OBJECT>`, здесь все необходимые параметры задаются в самом теге `<EMBED>`. Здесь указываются размеры внедренного модуля на Web-странице (атрибуты WIDTH и HEIGHT), имя проигрываемого файла Shockwave/Flash (атрибут SRC), путь к дистрибутиву модуля расширения (атрибут PLUGINSPAGE) и качество воспроизведения (атрибут QUALITY). Никакие дополнительные теги в этом случае не нужны.

Тег `<EMBED>` начал поддерживаться Web-обозревателями Microsoft Internet Explorer версии 3.0 и Netscape Navigator 2.0. Хотя он не стандартизирован и не рекомендован к использованию комитетом WWWW, но поддерживается до сих пор. Однако во вновь созданных Web-страницах рекомендуется использовать тег `<OBJECT>`.

Использование тегов `<OBJECT>` и `<EMBED>`

На практике, для решения проблем с совместимостью между разными программами Web-обозревателей теги `<OBJECT>` и `<EMBED>` используются совместно.

```
<OBJECT CLASSID="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" WIDTH="100"  
⚡HEIGHT="100" CODEBASE="http://active.macromedia.com/flash6/cabs/  
⚡swflash.cab#version=6,0,0,0">  
  
<PARAM NAME="MOVIE" VALUE="movie.swf">  
<PARAM NAME="PLAY" VALUE="true">  
<PARAM NAME="LOOP" VALUE="true">  
<PARAM NAME="QUALITY" VALUE="high">  
  
<EMBED SRC="movie.swf" WIDTH="100" HEIGHT="100" PLAY="true" LOOP="true"  
⚡QUALITY="high"  
⚡PLUGINSPAGE="http://www.macromedia.com/shockwave/download/index.cgi?  
⚡P1_Prod_Version=ShockwaveFlash">  
</EMBED>  
  
</OBJECT>
```

Здесь тег `<EMBED>` вложен внутри тега `<OBJECT>`, вместе с набором тегов `<PARAM>`.

Что же происходит, если клиент пытается просмотреть Web-страницу, содержащую такой код? Давайте рассмотрим эту ситуацию для разных Web-обозревателей.

Если клиент использует современный Web-обозреватель, а именно, Microsoft Internet Explorer 3.0 или Netscape Navigator 4.0 или более новые их версии, происходит следующее. Web-обозреватель считывает тег `<OBJECT>`, затем — теги `<PARAM>`, загружает компонент ActiveX, файл фильма и выводит его на экран. Тег `<EMBED>`, вложенный внутри тега `<OBJECT>`, в этом случае игнорируется.

Если же клиент использует более старые версии программ Web-обозревателей, все происходит по-другому. Так как теги `<OBJECT>` и `<PARAM>` не известны старым программам, они их игнорируют — это стандартное поведение Web-обозревателя, встретившего незнакомый тег. В этом случае обрабатывается тег `<EMBED>`, пользователи, так или иначе, увидят на Web-странице фильм Flash.

Есть, правда, и третий вариант. Клиент может использовать Microsoft Internet Explorer 2.0 — первую версию Web-обозревателя фирмы Microsoft, не поддерживающую ни `<OBJECT>`, ни `<EMBED>`. Но вряд ли в мире найдется много пользователей этой курьезной программы, так что их не стоит принимать в расчет.

Атрибуты тегов **<OBJECT>** и **<EMBED>**

Здесь мы опишем все атрибуты тегов **<OBJECT>** и **<EMBED>**, которые могут нам пригодиться.

ALIGN

Задаёт выравнивание окна проигрывателя Flash на Web-странице. Может принимать одно из четырёх значений:

- ☐ "left" — выравнивание по левой границе Web-страницы;
- ☐ "top" — выравнивание по верхней границе Web-страницы;
- ☐ "right" — выравнивание по правой границе Web-страницы;
- ☐ "bottom" — выравнивание по нижней границе Web-страницы.

Если окно Web-обозревателя меньше, чем нужно, три другие границы фильма будут обрезаны.

Применим к тегам **<OBJECT>** и **<EMBED>**. Также применим к тегу ****, используемому для внедрения графических изображений.

Если атрибут пропущен, окна проигрывателя располагается в центре Web-страницы.

Поддерживается, начиная с Microsoft Internet Explorer 3.0 и Netscape Navigator 2.0.

CLASSID

Задаёт идентификатор компонента ActiveX. В случае проигрывателя Flash должен быть равен "clsid:D27CDB6E-AE6D-11cf-96B8-444553540000". Не ошибитесь, набирая это значение!

Применим к тегу **<OBJECT>**.

Поддерживается, начиная с Microsoft Internet Explorer 3.0 и Netscape Navigator 4.0.

CODEBASE

Задаёт интернет-адрес дистрибутива компонента ActiveX. В случае проигрывателя Flash должен быть равен

"http://active.macromedia.com/flash6/cabs/swflash.cab#version=6,0,0,0". Не ошибитесь, набирая это значение!

Применим к тегу **<OBJECT>**.

Поддерживается, начиная с Microsoft Internet Explorer 3.0 и Netscape Navigator 4.0.

HEIGHT

Задаёт высоту окна проигрывателя Flash в пикселах или процентах от высоты окна Web-обозревателя. Формат использования:

HEIGHT="<Высота в пикселах>|<Процент от высоты>%"

Применим к тегам <OBJECT> и <EMBED>.

В теге <OBJECT> поддерживается, начиная с Microsoft Internet Explorer 3.0 и Netscape Navigator 4.0. В теге <EMBED> поддерживается, начиная с Microsoft Internet Explorer 3.0 и Netscape Navigator 2.0.

PLUGINSPAGE

Задаёт интернет-адрес дистрибутива модуля расширения. В случае проигрывателя Flash должен быть равен

"http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlash". Не ошибитесь, набирая это значение!

Применим к тегу <EMBED>.

Поддерживается, начиная с Netscape Navigator 2.0. Microsoft Internet Explorer не поддерживается.

SRC

Задаёт имя файла Shockwave/Flash, в котором сохранён фильм.

Применим к тегу <EMBED>.

Поддерживается, начиная с Microsoft Internet Explorer 3.0 и Netscape Navigator 2.0.

SWLIVECONNECT

Необязательный атрибут, задающий, должен ли Web-обозреватель загружать исполняющую подсистему JavaScript. Если равно false, подсистема JavaScript выгружается, когда фильм Flash запускается на воспроизведение. Если равно true, подсистема JavaScript не выгружается. Значение по умолчанию — false.

Используйте этот параметр, чтобы сэкономить память клиентского компьютера, выгрузив подсистему JavaScript. Однако, если вы используете действие FSCommand для вызова кода, написанного на JavaScript, вам будет нужно установить значение этого параметра, равным true.

Применим к тегу <EMBED>.

Поддерживается, начиная с Netscape Navigator 2.0. Microsoft Internet Explorer не поддерживается.

WIDTH

Задаёт ширину окна проигрывателя Flash в пикселах или процентах от ширины окна Web-обозревателя. Формат использования:

`WIDTH="<Ширина в пикселах>|<Процент от ширины>%"`

Применим к тегам `<OBJECT>` и `<EMBED>`.

В теге `<OBJECT>` поддерживается, начиная с Microsoft Internet Explorer 3.0 и Netscape Navigator 4.0. В теге `<EMBED>` поддерживается, начиная с Microsoft Internet Explorer 3.0 и Netscape Navigator 2.0.

Параметры проигрывателя Flash, реализованного в виде компонента ActiveX

Здесь будут описаны все параметры проигрывателя Flash, реализованного в виде компонента ActiveX, которые могут вам пригодиться. Эти параметры применимы только к тегу `<OBJECT>`, поддерживаемому, начиная с Microsoft Internet Explorer 3.0 и Netscape Navigator 4.0.

BASE

Необязательный параметр, задающий базовый интернет-адрес, используемый как точка для отсчёта всех относительных адресов.

BGCOLOR

Необязательный параметр, задающий цвет фона фильма в формате `0xRRGGBB`. Изменяет цвет фона, заданный при настройке параметров фильма в среде Flash.

DEVICEFONTS

Необязательный параметр, разрешающий или запрещающий проигрывателю Flash подставлять шрифты-псевдонимы вместо шрифтов, не установленных на клиентском компьютере. Может принимать одно из двух значений:

- ☐ `true` — подстановка разрешена;
- ☐ `false` — подстановка запрещена (значение по умолчанию).

Имейте в виду, что такая подстановка работает только в обычных текстовых блоках.

LOOP

Необязательный параметр. Если равен `true`, то фильм будет воспроизводиться бесконечно, "зациклится". Если равен `false`, то проигрывание фильма остановится после его завершения. Значение по умолчанию — `true`.

MENU

Необязательный параметр, разрешающий или запрещающий проигрывателю Flash выводить контекстное меню при щелчке правой кнопкой мыши. Может принимать одно из двух значений:

- ☐ `true` — при щелчке правой кнопкой мыши выводится полноразмерное контекстное меню (значение по умолчанию);
- ☐ `false` — контекстное меню будет содержать только пункт **About Macromedia Flash Player**.

MOVIE

Задаёт имя файла Shockwave/Flash, в котором сохранён фильм.

PLAY

Необязательный параметр. Если равен `true`, то фильм будет запущен на воспроизведение сразу же после загрузки. Если равен `false`, то фильм не будет запущен после загрузки; вам будет нужно сделать это самостоятельно из сценария на JavaScript. Значение по умолчанию — `true`.

QUALITY

Необязательный параметр, задающий качество графики. Может принимать одно из шести значений:

- ☐ `"low"` — скорость воспроизведения имеет приоритет перед качеством графики, сглаживание не используется. Самое низкое качество;
- ☐ `"autolow"` — изначально качество графики низкое, сглаживание не используется. Потом, если компьютер окажется достаточно мощным, Flash повысит качество графики и включит сглаживание;
- ☐ `"autohigh"` — изначально качество графики высокое, используется сглаживание. Потом, если компьютер окажется недостаточно мощным, Flash понизит качество графики и отключит сглаживание;
- ☐ `"medium"` — используется сглаживание для векторной, но не для растровой графики. Среднее качество;
- ☐ `"high"` — качество графики имеет приоритет над скоростью воспроизведения, используется сглаживание векторной графики, а если нет анимации, то и растровой графики. Это значение по умолчанию;
- ☐ `"best"` — используется сглаживание и векторной, и растровой графики. Самое высокое качество, которое не уменьшается ни в каких случаях.

Значение по умолчанию — `"high"`.

SALIGN

Необязательный параметр, задающий выравнивание фильма в окне проигрывателя Flash. Может принимать одно из восьми значений:

- ☐ "l" — выравнивание по левому краю;
- ☐ "t" — выравнивание по верхнему краю;
- ☐ "r" — выравнивание по правому краю;
- ☐ "b" — выравнивание по нижнему краю;
- ☐ "tl" — выравнивание по верхнему и левому краю;
- ☐ "tr" — выравнивание по верхнему и правому краю;
- ☐ "br" — выравнивание по нижнему и правому краю;
- ☐ "bl" — выравнивание по нижнему и левому краю.

Если параметр пропущен, фильм располагается в центре окна проигрывателя.

SCALE

Необязательный параметр, задающий параметры масштабирования фильма. Может принимать одно из трех значений:

- ☐ "showall" — будет показано все изображение, для чего может быть применено масштабирование. Однако пропорции изображения искажены не будут, в результате этого вдоль горизонтальных или вертикальных сторон его могут появиться границы;
- ☐ "noborder" — то же самое, что "showall", но границы появляться не будут — Flash обрежет изображение по горизонтали или вертикали, чтобы их избежать;
- ☐ "exactfit" — будет показано все изображение, для чего может быть применено масштабирование, в результате которого могут исказиться размеры изображения.

Значение по умолчанию — "showall".

Этот параметр имеет смысл задавать только тогда, когда значения атрибутов HEIGHT и WIDTH заданы в процентах.

WMODE

Необязательный параметр, задающий вид окна проигрывателя на Web-странице. Может принимать три значения:

- ☐ "window" — фильм отображается на Web-странице в собственном "окошке". Самая высокая скорость воспроизведения. Это значение по умолчанию;
- ☐ "opaque" — фильм не будет отображаться в собственном "окошке". Элементы, находящиеся на Web-странице "ниже" этого фильма, не будут видны;

- "transparent" — фильм не будет отображаться в собственном "окошке". Элементы, находящиеся на Web-странице "ниже" этого фильма, будут видны. Учтите, что в этом случае может ухудшиться качество проигрывания фильма.

Шаблоны HTML

А теперь поговорим о том, как Flash создает Web-страницы, необходимые для публикации фильмов в Интернете.

Макросы

В *главе 19*, посвященной публикации и экспорту, говорилось, что для создания Web-страниц с внедренными фильмами Flash используются шаблоны HTML. Это особые Web-страницы, часть HTML-кода которых заменена так называемыми *макросами*. При форматировании нужной Web-страницы Flash заменяет эти макросы реальным кодом HTML.

Flash поставляется с довольно большим набором шаблонов, которых вам должно хватить в большинстве случаев. (Все эти шаблоны перечислены в *главе 19*.) Но вы можете легко создать свои и изменить существующие шаблоны. Все файлы HTML, в которых хранятся шаблоны, находятся в каталоге Application Data/Macromedia/Flash MX/Configuration/HTML. Если вы работаете в Windows 95/98/Me, то вам нужно искать его в каталоге, где установлена Windows. Если же вы предпочитаете Windows NT/2000/XP, то найдете его в каталоге вашего пользовательского профиля.

Имена всех макросов, используемых в коде шаблонов, начинаются со знака доллара \$. Если вам нужно вставить в код HTML знак доллара, воспользуйтесь комбинацией символов /\$.

Рассмотрим несколько примеров макросов.

В начале кода каждого шаблона HTML находятся макросы, задающие его имя и текстовое описание. Это имя и описание появляются в диалоговом окне **Htm1 Template Info**. Имя шаблона задает макрос \$TT, а описание — пара макросов \$DS и \$DF.

```
$TTFlash Only
```

```
$DS
```

```
Display Macromedia Flash Movie in HTML.
```

```
$DF
```

Приведенный выше код задает имя и описание шаблона **Flash Only**, используемого для создания простых Web-страниц с внедренным фильмом Flash. Хорошо видно, что имя шаблона следует сразу за макросом \$TT, а описание находится между макросами \$DS и \$DF.

```
<EMBED $PE WIDTH="$SWI" HEIGHT="$HE" NAME="$TI" ALIGN="$HA"
  TYPE="application/x-shockwave-flash"
  PLUGINSOURCE="http://www.macromedia.com/go/getflashplayer">
</EMBED>
```

Этот код, как вы поняли, задает параметры модуля расширения. Значения ширины и высоты задаются соответственно макросами \$SWI и \$HE, имя — макросом \$TI, а выравнивание — макросом \$HA. Обратите внимание, что атрибуты TYPE и PLUGINSOURCE заданы не макросами, а их реальными значениями, которые во всех случаях будут одними и теми же.

А что задает макрос \$PE? Все остальные атрибуты тега <EMBED>, если они должны иметь значения, отличные от значений по умолчанию. Это атрибуты SRC и SWLIVECONNECT.

```
<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="$SWI" HEIGHT="$HE" id="$TI"
  ALIGN="$HA">
$PO
</OBJECT>
```

А этот код задает параметры внедренного компонента ActiveX. В нем используются те же макросы, что и для задания параметров модуля расширения, за одним исключением — вместо макроса \$PE используется макрос \$PO.

Все поддерживаемые Flash макросы HTML перечислены в табл. П2.1.

Таблица П2.1. Макросы HTML, поддерживаемые Flash

Атрибут/параметр	Макрос
Имя шаблона	\$TT
Описание шаблона (начало)	\$DS
Описание шаблона (конец)	\$DE
Имя фильма (совпадает с именем файла Shockwave/Flash без расширения)	\$TI
Ширина (атрибут WIDTH)	\$SWI
Высота (атрибут HEIGHT)	\$HE
Имя файла Shockwave/Flash, в котором хранится фильм (параметр MOVIE и атрибут SRC)	\$MO
Местоположение окна проигрывателя на Web-странице (атрибут ALIGN)	\$HA

Таблица П2.1 (продолжение)

Атрибут/параметр	Макрос
Защелкивание фильма (параметр <code>LOOP</code>)	<code>\$LO</code>
Атрибуты тега <code><OBJECT></code> , чьи значения отличаются от значений по умолчанию	<code>\$PO</code>
Атрибуты тега <code><EMBED></code> , чьи значения отличаются от значений по умолчанию	<code>\$PE</code>
Будет ли фильм проигран сразу же после загрузки (параметр <code>PLAY</code>)	<code>\$PL</code>
Качество графики (параметр <code>QUALITY</code>)	<code>\$QU</code>
Масштабирование фильма в окне проигрывателя Flash (параметр <code>SCALE</code>)	<code>\$SC</code>
Выравнивание фильма в окне проигрывателя Flash (параметр <code>SALIGN</code>)	<code>\$SA</code>
Вид окна проигрывателя Flash на Web-странице (параметр <code>WMODE</code>)	<code>\$WM</code>
Подстановка шрифтов-псевдонимов (параметр <code>DEVICEFONTS</code>)	<code>\$DE</code>
Цвет фона фильма (параметр <code>BGCOLOR</code>)	<code>\$BG</code>
Весь текст, содержащийся в фильме	<code>\$MT</code>
Все гиперссылки, содержащиеся в фильме	<code>\$MU</code>
Ширина замещающего фильм изображения (формат не задан)	<code>\$IW</code>
Высота замещающего фильм изображения (формат не задан)	<code>\$IH</code>
Имя файла замещающего фильм изображения (формат не задан)	<code>\$IS</code>
Имя карты-изображения	<code>\$IU</code>
Список областей карты-изображения	<code>\$IM</code>
Ширина изображения или фильма QuickTime, замещающего фильм Flash	<code>\$QW</code>
Высота изображения или фильма QuickTime, замещающего фильм Flash	<code>\$QH</code>
Имя файла изображения или фильма QuickTime, замещающего фильм Flash	<code>\$QN</code>
Ширина изображения GIF, замещающего фильм Flash	<code>\$GW</code>

Таблица П2.1 (окончание)

Атрибут/параметр	Макрос
Высота изображения GIF, замещающего фильм Flash	\$GH
Имя файла изображения GIF, замещающего фильм Flash	\$GN
Ширина изображения JPEG, замещающего фильм Flash	\$JW
Высота изображения JPEG, замещающего фильм Flash	\$JH
Имя файла изображения JPEG, замещающего фильм Flash	\$JN
Ширина изображения PNG, замещающего фильм Flash	\$PW
Высота изображения PNG, замещающего фильм Flash	\$PH
Имя файла изображения PNG, замещающего фильм Flash	\$PN

Примеры использования макросов

Осталось привести несколько примеров использования описанных выше макросов.

Карты-изображения

Карта-изображение — это особое графическое изображение, разбитое на независимые *области*, к каждой из которых привязана гиперссылка. Когда пользователь щелкает по такой зоне, происходит переход по адресу, указанному в этой гиперссылке.

Чтобы создать карту-изображение, нужно иметь:

- ☐ графическое изображение, отображаемое на экране;
- ☐ список областей, описывающий их геометрическую форму, размеры и гиперссылку, привязанную к каждой из них. Эти области потом "накладываются" на графическое изображение.

Используемый для создания карт-изображений код показан ниже.

```
<MAP NAME="mymap">
```

```
<AREA COORDS="130,116,214,182" HREF="http://www.macromedia.com/flash/">
```

```
<AREA COORDS="180,244,252,300"
```

```
⌘ HREF="http://www.macromedia.com/dreamweaver/">
<AREA COORDS="280,34,376,52" HREF="http://www.macromedia.com/fireworks/">
</MAP>
<IMG SRC="mymap.gif" USEMAP="#mymap" WIDTH=550 HEIGHT=400 BORDER=0>
```

Список областей находится внутри парного тега `<MAP>`. Графическое изображение задается, как обычно, тегом ``. Обратите внимание, что для привязки списка областей к изображению используется атрибут `USEMAP` тега ``. И еще: список областей должен предшествовать изображению.

Обычно карты-изображения создаются, как правило, с помощью специальных утилит. Эти утилиты позволяют загрузить исходное графическое изображение и прямо на нем "нарисовать" нужные области. После этого остается задать для каждой области гиперссылку и вставить сформированный утилитой код HTML в Web-страницу. "Вручную" же создавать карты-изображения очень трудоемко.

Однако, если у вас есть Macromedia Flash, никакие дополнительные утилиты вам не нужны.

Создайте новый документ Flash и сформируйте в нем два ключевых кадра. Эти два кадра будут содержать соответственно графическое изображение и набор областей.

Набор областей представляет собой набор обычных кнопок. К каждой из этих кнопок вы должны будете привязать сценарий, обрабатывающий событие `press` или `release` и содержащий действие `getURL`. Например, такой:

```
on (release) {
    getURL("http://www.macromedia.com");
}
```

Присвойте кадру, содержащему набор кнопок, имя `#Map`. Если вы этого не сделаете, Flash в качестве набора кнопок использует последний кадр фильма.

Теперь приступайте к созданию изображения. После того, как вы его нарисуете, присвойте ему имя `#Static`. Если вы этого не сделаете, Flash в качестве набора кнопок использует тот кадр, на котором во время публикации стоял указатель.

Теперь остается создать необходимые файлы. Откройте окно **Publish Settings**. Включите создание файлов Shockwave/Flash, HTML и нужного графического формата (GIF, JPEG или PNG). Обязательно проверьте, не содержат ли имена этих файлов русские буквы; исправьте их, если содержат.

Переключитесь на вкладку **HTML** и выберите в раскрывающемся списке **Template** пункт **Image Map**. Задайте другие параметры экспорта, если хотите. И нажмите кнопку **Publish**.

Вам необязательно использовать именно шаблон **Image Map**. Вы можете использовать свой собственный шаблон. Поместите в его код макрос `$IM`, за-

дающий список областей, и тег ``. В качестве значения атрибута `USEMAP` тега `` задайте макрос `$IU`, а в качестве значения атрибута `SRC` — макрос `$IS` или аналогичный ему.

Код формирования карты-изображения может выглядеть так:

```
$IM
```

```
<IMG SRC=$IS WIDTH="$IW" HEIGHT="$IH" USEMAP=$IU>
```

Но проще всего, конечно, просто позаимствовать нужный код из файла `ImageMap.html`. Благо лицензионное соглашение этого не запрещает.

Замещающие изображения

Конечно, хорошо, если на компьютере клиента установлен проигрыватель Flash. Но что делать, если его там нет?

В этом случае на помощь могут прийти *замещающие изображения*. Это обычные изображения GIF, JPEG или PNG, которые отображаются вместо фильма Flash. Может быть, формат Shockwave/Flash поддерживают не все Web-обозреватели (не сами, конечно, а с помощью дополнительных программ), а уж GIF и JPEG знакомы всем.

Замещающие изображения формируются самим Flash при публикации фильма на основе его первого кадра. Вы можете задать для создания замещающего изображения любой другой кадр фильма, просто дав ему имя `#Static`.

Замещающие изображения помещаются на Web-страницу с помощью уже знакомого вам тега ``. Нужный для этого код может выглядеть следующим образом:

```
<IMG SRC=$IS WIDTH="$IW" HEIGHT="$IH">
```

Заметьте, что мы задали имя файла изображения макросом `$IS`. Вы также можете воспользоваться макросами `$GN`, `$JN` или `$PN`.

Если вы создаете замещающее изображение в формате GIF, то можете сделать его анимированным. В этом случае Flash просто сохранит фильм в формате "анимированный GIF". Задать первый и последний кадр анимационной последовательности, которая будет экспортирована в файл GIF, можно просто назвав соответственно первый и последний ее кадры `#First` и `#Last`.

Текст фильма и список гиперссылок

Вместо замещающего изображения или вместе с ним вы можете вывести на Web-страницу весь текст, присутствующий в фильме. Для этого поместите в код шаблона макрос `$MT`. После этого пользователь, на чьем компьютере не установлен проигрыватель Flash, хотя бы сможет прочитать текстовую информацию.

Кроме того, вы можете поместить на страницу все гиперссылки, которые встретились в вашем фильме. Для этого поместите в код шаблона макрос `$MU`.

Предметный указатель

A

ActionScript 419

D

DOM 601

Drag and drop 500

◊ цель 501

W

Web 19

Web-обозреватель 20, 21, 101, 104, 175

Web-сервер 250

Web-страница 19, 101

WWW 19

WWWC 19

Z

z-координата 205, 489

A

Альфа-канал 75, 95, 96, 266, 273, 275

Анимация:

◊ "бесконечная" 316

◊ вложенная 329

◊ второго уровня 331

◊ многоуровневая 329

◊ первого уровня 329

◊ покадровая 281, 286, 295

◊ трансформационная 284, 286

◊ фон 338

Апплет 23

Атрибут HTML 814

Б

Баннер 24, 95

Библиотека 124, 193, 202, 203, 220

◊ кода 527

◊ общего использования 254

◊ разделяемая 247

Блокировка 219

Буфер обмена 74, 76, 102, 127, 172, 180,
218, 257, 301

В

Видеодорожка 407

Видимость 446, 465

Возврат 145

Выделение 45, 71, 72, 119, 132, 172

◊ окружность 164

◊ перетаскиванием 133

▫ мыши 121

(окончание рубрики см. на с. 830)

Выделение (окончание):

- ◇ прямоугольник 161, 162, 211, 213, 216, 217
 - ◇ сложное 133
- Выравнивание 205, 206, 208
- Выражение 441
- ◇ блок 457
 - ◇ выбора 459
 - ◇ сложное 457
 - ◇ условное 457

Г

Гиперссылка 175

- ◇ цель 175

График 379

Графика:

- ◇ векторизация 92, 197
- ◇ векторная 76, 90, 92, 93, 99, 101, 103, 128, 193, 260, 285
- ◇ вращение 213, 217
- ◇ гибридная 90, 92, 101
- ◇ деформация 215
- ◇ заливка 117
- ◇ зеркальное отражение 210
- ◇ изменение размеров 208, 210, 212, 217
- ◇ изменение формы 129
- ◇ искажение формы 214
- ◇ масштабирование 213
- ◇ перемещение 126, 208
- ◇ полноцветная 88, 98
- ◇ разбиение 199
- ◇ растрезация 86, 91, 92, 194
- ◇ растровая 74, 75, 86, 88, 89, 92—94, 101, 103, 128, 160, 192, 193, 198, 199, 202, 237, 238, 260, 282
- ◇ сброс преобразований 218
- ◇ сдвиг 213, 217
- ◇ созданная программно 504
- ◇ стирание 136
- ◇ трансформация свободная 217
- ◇ удаление 126
- ◇ фотореалистичная 88
- ◇ чередование 95—97, 263, 266, 267, 272, 274, 275

Группа 72

- ◇ разбиение 125
- ◇ фрагментов 125, 128, 205, 215, 226, 313

Группировка 125, 196, 205, 219

Д

Данное 441

Действие 441, 456

Декордер 358

Дискретизация 384

Документ:

- ◇ возврат к последнему сохранению 59
- ◇ закрытие 59
- ◇ открытие 59
- ◇ отправка по почте 59
- ◇ параметры 56
- ◇ пересохранение 59
- ◇ печать 61
- ◇ создание 55, 57
- ◇ сохранение 59

Дорожка:

- ◇ видео 400
- ◇ звуковая 376

Доступность 386

Е

Единица измерения 56

З

Заголовок панели 36, 37

Заливка 106, 112, 116, 119, 120, 129, 132, 138, 142, 143, 152, 162, 167, 168, 197

- ◇ градиентная 76, 118, 157, 159, 161, 165, 169, 195, 215, 264, 268
 - линейная 157, 163
 - радиальная 164

◇ графическая 118, 160, 161, 165, 200

◇ многоцветная 159

◇ прозрачная 156

◇ радиальная 157

◇ сплошная 156, 264

◇ фиксированная 165

Звук 238, 254, 258, 370

◇ гибридный 372

◇ команды воспроизведения 371, 375

◇ панорамирование 377

◇ потоковый 378

◇ сигнал 378

◇ синтез частотный 371

И

Изображение:

- ◇ заливка 161, 163

- ◇ замещающее 828
- ◇ маска 509
- ◇ маскируемое 351
- Импорт 77, 160, 192, 203, 221, 226, 238
- Имя 181, 182, 202, 223, 237, 307, 318, 335
- ◇ альтернативное 395
- ◇ доменное 493
- Инициализатор 472
- Инструмент 35, 45, 371
 - ◇ банк 371
 - ◇ ведро с краской 117, 166, 167, 200
 - ◇ карандаш 74, 109, 110, 139
 - ◇ кисть 114, 139, 166
 - ◇ лассо 134, 135, 199
 - ◇ ластик 136
 - ◇ линия 107
 - ◇ лупа 46
 - ◇ перо 73, 111, 117, 132, 133, 148
 - ◇ пипетка 167, 200
 - ◇ прямоугольник 108, 117, 148
 - ◇ пузырек с чернилами 156, 167
 - ◇ рука 47
 - ◇ стрелка выделения 116, 119, 126, 127, 129, 133, 134, 138, 139, 142, 171, 173, 181, 184, 186, 214
 - ◇ стрелка выделения дополнительная 131, 132
 - ◇ текст 171, 172
 - ◇ текстовый блок 178
 - ◇ трансформатор 210, 213, 215, 217, 229
 - ◇ трансформатор заливки 161
 - ◇ эллипс 108, 117, 148
- Инструментарий 36, 39, 45
 - ◇ вспомогательный 39, 42
 - общего назначения 42
 - управления проигрыванием 42, 298
 - ◇ главный 35, 40
 - ◇ область 35
- Интернет 19
- Интерфейс:
 - ◇ многодокументный 34
 - ◇ одноктокументный 34

К

- Кадр 44, 72, 77, 257, 258, 281, 283, 294, 302, 306, 307, 310
- ◇ диапазон 304, 305

- ◇ ключевой 196, 230, 285, 295, 296, 300, 301, 361
- ◇ печатаемый 416
- ◇ промежуточный 71, 285, 297, 300, 361
- ◇ растянутый 71, 297, 300, 301
- ◇ текущий 294, 296, 301, 303, 308
- ◇ указатель 296, 298, 305
- ◇ частота 56, 359
- Карта-изображение 826
 - ◇ область 826
- Качество графики 48
- Клавиатурная комбинация 79
 - ◇ набор 79
- Клавиша:
 - ◇ код виртуальный 510
 - ◇ служебная 510
- Клиент 588
- Клип 194, 258, 428
 - ◇ "живого просмотра" 579
 - ◇ внедренный 355
 - ◇ импортированный 258
 - ◇ накладывающийся 489
 - ◇ связанный 355
 - ◇ текущий 490
- Ключевое слово 441
- Кнопка 254, 258, 529
 - ◇ переключатель 47
- Кодек 358
- Кодер 358
- Команда 441
- Комментарий 456
- Компилятор 21
- Компиляция 21
- Компонент 530, 544
 - ◇ ActiveX 23
 - ◇ "шкура" 565
 - ◇ полоса прокрутки 185
 - ◇ стиль 563
- Константа 441, 444
- Конструктор 472
- Контейнер 171
- Контур 116, 117, 120, 129, 132, 138, 143, 152, 162, 197, 198, 201
- Конфликт имен 243, 251

Л

- Линейка координатная 49
- Линия 73, 106, 116, 129, 138, 142, 152, 167
- (окончание рубрики см. на с. 832)*

Линия (окончание):

- ◇ касательная 113, 133
 - ◇ кривая 74, 113, 139, 140, 144
 - ◇ кривая Безье 111
 - ◇ ломаная 111, 112
 - ◇ прямая 74, 107, 110—112, 140, 505
 - ◇ свободной формы 109
 - ◇ стиль 152, 156
 - ◇ толщина 152
- Лист рабочий 44, 119, 122, 137, 208, 302, 306, 308

М

Макрос 823

Маркер 162, 216, 217

- ◇ диапазона 304
- ◇ изменения:
 - радиуса 165
 - размера 162, 164, 173, 185, 211
- ◇ конечный 297
- ◇ конца звука 381
- ◇ начала звука 381
- ◇ огибающей 381
- ◇ поворота 163, 165, 213
- ◇ сдвига 163, 213
- ◇ смещения 162, 164
- ◇ трансформации 324

Маска 351

Массив 87, 88, 468

- ◇ вложенный 469
- ◇ индекс 469
- ◇ размер 469
- ◇ элемент 468

Масштаб 44, 46

Масштабирование 89

Метод 470

- ◇ GET 595
- ◇ POST 596
- ◇ передачи данных 595
- ◇ проигрывателя Flash 592

Модификатор 36, 441

- ◇ волшебная палочка 199, 200
- ◇ вращение и сдвиг 213
- ◇ изменение размера 211
- ◇ искажение формы 214, 229
- ◇ кран 139
- ◇ огибающая 215, 229
- ◇ полигон 135
- ◇ притягивание 74, 107, 127

- ◇ размер кисти 115
- ◇ размер разрыва 118
- ◇ режим закрашки 116
- ◇ режим карандаша 74
- ◇ сглаживание 139
- ◇ сглаживание линий 110
- ◇ скругленные углы 108
- ◇ спрямление 140
- ◇ стирание 138
- ◇ увеличение 47
- ◇ уменьшение 47
- ◇ фиксация заливки 166, 167
- ◇ форма кисти 115
- ◇ форма ластика 137

Морфинг 312, 320

Мультимедиа 279

Н

Направляющая 49, 127

Наследование 481

О

Область печати 416

Область рабочая 44, 46

Обработчик 429

- ◇ заголовок 429
- ◇ тело 430

Образец 124, 193, 202, 203, 205, 220, 256, 257

- ◇ внешний 235
- ◇ графический 222, 258
- ◇ дублирование 234
- ◇ звук 222, 258, 376
- ◇ импортированный 223, 256
- ◇ импортированный клип 222, 258
- ◇ источник 246
- ◇ клип 222, 258
- ◇ кнопка 222, 258
- ◇ компонент 573
- ◇ обновляемый 246
- ◇ разделяемый 247, 249, 250, 252, 254, 528
- ◇ растровое изображение 222, 258
- ◇ редактирование 232, 256
 - в отдельном окне 232, 233, 256
 - "на месте" 232, 233, 256

- ◇ создание 223
- ◇ сценарный 497
- ◇ тип 222, 224
- ◇ шрифт 223, 252, 256
- Объект 470
 - ◇ вложенный 471
 - ◇ внешний 483
 - ◇ внутренний 471
 - ◇ встроенный 473
 - ◇ дочерний 481
 - ◇ перехватчик 512
 - ◇ пользовательский 478
 - ◇ поставщик данных 558
 - ◇ потомок 481
 - ◇ предок 481
 - ◇ прототип 480
 - ◇ родитель 481
- Огибающая 381
- Окно:
 - ◇ Output 611
 - ◇ библиотеки 224, 235, 245
 - ◇ главное 33, 36, 37
 - ◇ документа 34, 36, 40, 43, 308
 - ◇ отладчика 614
 - ◇ приглашение 34
 - ◇ Проводника 256
 - ◇ родительское 34
- Оператор 441
 - ◇ typeof 453
 - ◇ арифметический 448
 - ◇ бинарный 448
 - ◇ двоичный 449
 - ◇ декремента 448
 - ◇ инкремента 448
 - ◇ комментария 456
 - ◇ логический 452
 - ◇ объединения строк 449
 - ◇ приоритет 454
 - ◇ присваивания 450
 - сложного 450
 - ◇ сравнения 451
 - строгого 452
 - ◇ унарный 448
 - ◇ условный 459
- Оптимизация 139, 140, 259, 263, 267, 386, 391
 - ◇ многопроходная 141
- Откат 70, 145
- Отладка 610
 - ◇ удаленная 621
- Отладчик 611

- Ошибка 609
 - ◇ логическая 610
 - ◇ синтаксическая 609

П

- Палитра 147, 149, 152, 167, 168, 263, 265, 267, 268
 - ◇ адаптивная 265, 268, 269, 275
 - ◇ безопасная 169, 264, 265, 268, 272
 - ◇ по умолчанию 169
- Панель 36, 39, 71
 - ◇ Actions 77, 421
 - ◇ Align 206
 - ◇ Answers 66
 - ◇ Color Swatches 167
 - ◇ Component Parameters 547
 - ◇ Components 545
 - ◇ Info 209
 - ◇ Reference 67
 - ◇ Scene 307, 309
 - ◇ Transform 212, 213, 217
 - ◇ группа 37
 - ◇ закрытие 39, 40
 - ◇ кнопка закрытия 39
 - ◇ кнопка раскрытия 38
 - ◇ контекстное меню 42
 - ◇ меню дополнительное 38
 - ◇ параметров 581
 - ◇ развертывание 37
 - ◇ раскладка 40
 - ◇ ручка 36
 - ◇ сжатие 37
 - ◇ сккрытие 40
- Папка 240, 242
- Переключатель 236
 - ◇ точки отсчета 209
- Перекрытие 204
 - ◇ порядок 205
- Переменная 182, 184, 441, 445
 - ◇ глобальная 446
 - ◇ локальная 446
 - ◇ объявление 445
 - ◇ системная 447
 - ◇ уровня клипа 446
- Перо 505
- Поддомен 493
- Подсказка:
 - ◇ всплывающая 435
 - ◇ список 435
- Поле ввода 180

Порт 606
Приложение:
◊ клиентское 588
◊ серверное 588
Примечание 319
Примитив графический 45, 90, 92, 105, 194, 284
Притягивание 126
Проводник 255, 310, 313, 353, 439
Проигрыватель 21, 32, 103, 104, 176
Протокол 594
◊ HTTP 594
Профилировщик загрузки 388
Публикация 30, 48, 259, 398, 439
Пункт меню:
◊ выключатель 40
◊ переключатель 48
Путь 349, 471, 486
◊ абсолютный 487
◊ относительный 487

Р

Рабочий лист 45
Разрешение 75, 88, 91
Распределение 206, 207, 339
Растр 86, 90
Растровая графика 258
Расширение 625
Регулятор 151
Редактор свойств 38, 129, 152, 156, 173, 178, 181, 208, 227, 229
Рекурсия 467
◊ бесконечная 467

С

Свойство 470
◊ динамическое 480
Сглаживание 48, 74, 76, 111, 139, 141, 143, 201, 202, 264, 267, 272, 273, 275
◊ степень 141
Селектор цвета 51, 147, 150, 156, 158, 159
Селектор цветов 157, 167
Сетка координатная 52, 127
Сжатие 90, 202, 268, 275, 371
◊ ADPCM 383
◊ Deflate 96, 98
◊ DivX 281, 288, 289
◊ Intel Indeo 288

◊ JPEG 97, 98, 202, 276
◊ LZW 95, 96, 98
◊ MPEG 1 280, 288
◊ MPEG 1 level 3 288
◊ MPEG 2 281, 288, 289
◊ MPEG 4 281, 288, 289
◊ QuickTime 287
◊ RealMedia 290
◊ RLE 94
◊ ROB 95
◊ Sorenson Spark 358
◊ алгоритм 202
◊ без потерь 97, 203, 371
◊ коэффициент 104
◊ с потерями 97, 280, 283, 371

Символ:

◊ ASCII/S-JIS 725
◊ Unicode 442
◊ код 442
◊ ASCII 442
◊ кодировка 442
◊ специальный 442
Слияние 123, 124, 195
Слой 44, 125, 194, 196, 197, 257, 258, 313, 335
◊ блокировка 343
◊ маскируемый 351
◊ маскирующий 351
◊ направляющая 349
◊ папка 345
◊ тип 344
Событие 419, 463
◊ обработчик 464
Справка 64
Ссылка 471
Строка статуса 42, 45, 295, 303, 305
Сцена 72, 196, 307, 309, 310, 388
◊ текущая 308, 310
Сценарий 20, 182, 257, 258, 307, 419, 441
Сэмпл 371

Т

Таймер 524
◊ идентификатор 524
Тег:
◊ HTML 182
◊ MP3 373
◊ PNG 193
◊ TIFF 98
◊ XML 600

Текст 73, 103, 128, 170, 173, 194
◇ абзац 178
Текстовый блок 76, 128, 171—173, 178,
181, 184, 194, 196, 215, 257, 313
◇ вертикальный 179
◇ динамический 184
◇ прокручиваемый 185
◇ с фиксированной высотой 172, 173
◇ с фиксированной шириной 172, 173
◇ статический 184

Тип данных 442

◇ NaN 748
◇ null 444
◇ undefined 444
◇ клип 444, 485
◇ логический 444
◇ объект 444, 469, 472
◇ преобразование 453, 468
◇ совместимость 453
◇ строковый 442
◇ функция 444, 467
◇ числовой 443

Точка:

◇ искривления 73, 133
◇ ключевая 73
◇ останова 611
◇ притягивания 127
◇ угловая 73, 129, 130, 132, 133
◇ фиксации 213, 224—226, 233

Трансформация:

◇ движения 313
◇ формы 320

Трассировка 611

Трекер 372

У

Узел 601

◇ дочерний 601
◇ родительский 601
◇ тип 601

Уровень 489

Ф

Фильм:

◇ импортированный 239
◇ основной 485
◇ AutoCAD 273
◇ AVI 23, 31, 280, 287, 289, 290

◇ BMP 23, 85, 94, 104, 260, 272
◇ CorelDRAW 101
◇ DIB 94
◇ Encapsulated PostScript 101, 104
◇ Enhanced Windows Metafile 100
◇ EPS 195
◇ GIF 23, 31, 86, 95—97, 104, 169, 260,
262, 271
◇ GIF анимированный 95, 263, 291
◇ GIF89A 95
◇ JPEG 23, 31, 97, 104, 261, 265, 273
◇ JPEG прогрессивный 97
◇ Macromedia Flash Video 367
◇ MIDI 375
◇ MIDI General 375
◇ MP3 373
◇ MPEG 23, 288, 289
◇ PCX 94
◇ PDF 102
◇ PICT 98, 104
◇ PNG 96, 97, 104, 193, 260, 266, 274
◇ QuickTime 23, 31, 280, 287, 291
◇ RealAudio 374
◇ RealMedia 290, 374
◇ Shockwave/Flash 21, 31, 99, 104, 167,
176, 177, 183, 186, 195, 198, 247, 276,
291, 299, 307
◇ TIFF 98, 104
◇ VML 101
◇ VRML 103
◇ WAV 372
◇ Windows Metafile 100
◇ WMA 290, 374
◇ WMV 290
◇ документа Flash 30, 99, 187
◇ модуль трекерный 372, 375
Формат файла 85
◇ Adobe Illustrator 100, 104, 271
Фрагмент графический 46, 120, 121, 186
Фрагментация 122, 124, 195
Фрейм 175
Функция 464
◇ аргумент 465
 ▫ фактический 466
 ▫ формальный 465
◇ встроенная 468
◇ вызов 466
 ▫ вложенный 618
◇ глобальная 466
записи 480
(окончание рубрики см. на с. 836)

Функция (окончание):

- ◊ объявление 465
- ◊ результат 465
- ◊ тело 465
- ◊ уровня клипа 466
- ◊ чтения 480

Ц

Цвет 87, 106, 110, 113—115, 118, 119, 129, 147, 149, 157, 168, 210

- ◊ CMYK 195
- ◊ HSB 151
- ◊ RGB 151, 195
- ◊ ключевой 157—159, 264
- ◊ код 147, 151
- ◊ отключение 148
- ◊ режим задания 151
- ◊ смеситель 149, 150, 157—159, 161
- ◊ составной 264, 267, 268, 272, 275

Цветовой режим 75, 273, 275

- ◊ HiColor 48, 75, 87
- ◊ TrueColor 48, 75, 87, 88, 95, 97, 266, 273, 275

Цикл 460

- ◊ перезапуск 463
- ◊ прерывание 463
- ◊ просмотра 473
- ◊ с постусловием 462
- ◊ с предусловием 462
- ◊ со счетчиком 461
- ◊ счетчик 461
- ◊ тело 461
- ◊ условие 461

Ч

Частота кадров 279

Ш

Шаблон 57

- ◊ HTML 403, 823
- ◊ создание 60

Ширина потока данных 359, 384

Шкала временная 44, 46, 71, 293, 302, 303, 305, 306, 308

Шрифт 76, 174, 177, 183

- ◊ TrueType 91, 174, 176, 187
- ◊ Type 1 174, 176, 187
- ◊ базовая линия 175
- ◊ внедрение 176, 183
- ◊ кегль 174
- ◊ кернинг 73, 175
- ◊ подстановка 72, 187
- ◊ псевдоним 177, 179
- ◊ размер 174
- ◊ растровый 176
- ◊ трекинг 174

Э

Экземпляр 72, 193, 199, 205, 215, 221, 257, 301, 313

- ◊ графический 258
- ◊ изменение цвета 227
- ◊ импортированный 257
- ◊ объекта 470
- ◊ прозрачность 227
- ◊ смена 229
- ◊ создание 226
- ◊ тип 222, 229

Экспорт 21, 176, 183, 186, 259, 269, 398

- ◊ внедренный 814
- ◊ встроенный 544
- ◊ графический 46, 256
- ◊ надпись 394
- ◊ пользовательский 530, 544
- ◊ порядок обхода 533
- ◊ управление 254, 530, 544

Я

Язык программирования:

- ◊ ActionScript 22
 - ◊ Curl 24
 - ◊ HTML 19, 22, 101, 103
 - ◊ Java 23
 - ◊ JavaScript 20, 22
 - ◊ VRML 103
 - ◊ XML 575, 600
 - ◊ интерпретируемый 21
 - ◊ компилируемый 20, 21
- Якорь именованный 7