# Руководство разработчика ACTIONSCRIPT® 3.0

## Юридическая информация

Юридическую информацию см. на веб-странице http://help.adobe.com/ru_RU/legalnotices/index.html.

Последнее обновление 7.10.2015.

# Содержание

**Глава 1. Работа с датами и временем**
Управление календарными датами и временем . . . . . 1
Управление временными интервалами . . . . . 4
Пример даты и времени: простые аналоговые часы . . . . . 6

**Глава 2. Работа со строками**
Основные сведения о строках . . . . . 10
Создание строк . . . . . 11
Свойство length . . . . . 12
Работа с символами в строках . . . . . 12
Сравнение строк . . . . . 13
Получение строковых представлений других объектов . . . . . 14
Сцепление строк . . . . . 14
Поиск подстрок и шаблонов в строках . . . . . 15
Перевод строк в верхний или нижний регистр . . . . . 19
Пример строк: ASCII-графика . . . . . 20

**Глава 3. Работа с массивами**
Основные сведения о массивах . . . . . 26
Индексные массивы . . . . . 28
Ассоциативные массивы . . . . . 39
Многомерные массивы . . . . . 43
Клонирование массивов . . . . . 44
Расширение класса Array . . . . . 45
Пример массивов: PlayList . . . . . 50

**Глава 4. Обработка ошибок**
Основы обработки ошибок . . . . . 55
Типы ошибок . . . . . 57
Обработка ошибок в ActionScript 3.0 . . . . . 59
Работа с отладочными версиями сред выполнения Flash . . . . . 61
Обработка синхронных ошибок в приложении . . . . . 62
Создание пользовательских классов ошибок . . . . . 67
Ответы на события ошибок и ошибки состояния . . . . . 68
Сравнение классов ошибок . . . . . 71
Пример обработки ошибок: программа CustomErrors . . . . . 75

**Глава 5. Использование регулярных выражений**
Основы регулярных выражений . . . . . 82
Синтаксис регулярных выражений . . . . . 83
Методы для использования регулярных выражений со строками . . . . . 97
Пример регулярных выражений: синтаксический анализатор Wiki . . . . . 98

**Глава 6. Работа с XML**
Основы XML . . . . . 103
Подход E4X к обработке XML . . . . . 106
Объекты XML . . . . . 108
Объекты XMLList . . . . . 110
Инициализация переменных XML . . . . . 111
Компоновка и преобразование объектов XML . . . . . 113
Пересечение XML-структур . . . . . 114
Использование пространств имен XML . . . . . 119
Преобразование типа XML . . . . . 120
Чтение внешних XML-документов . . . . . 121
Пример XML в ActionScript: загрузка данных RSS из Интернета . . . . . 122

**Глава 7. Использование встроенных функций JSON**
Обзор API-интерфейса JSON . . . . . 126
Определение пользовательского поведения JSON . . . . . 127

**Глава 8. Обработка событий**
Основы обработки событий . . . . . 134
Чем обработка событий в ActionScript 3.0 отличается от предыдущих версий . . . . . 136
Поток событий . . . . . 138
Объекты Event . . . . . 140
Прослушиватели событий . . . . . 145
Пример обработки событий: будильник . . . . . 151

**Глава 9. Работа с доменами приложений**

**Глава 10. Программирование отображаемого содержимого**
Основы программирования отображаемого содержимого . . . . . 163
Основные классы показа . . . . . 166
Преимущества подхода с использованием списка отображения . . . . . 168
Работа с экранными объектами . . . . . 171
Операции с экранными объектами . . . . . 186
Анимация объектов . . . . . 208
Ориентация рабочей области . . . . . 211
Динамическая загрузка содержимого для показа . . . . . 214
Пример экранного объекта: SpriteArranger . . . . . 220

**Глава 11. Работа с геометрией**
Основы геометрии . . . . . 227
Использование объектов Point . . . . . 228
Использование объектов Rectangle . . . . . 230
Использование объектов Matrix . . . . . 233
Пример геометрии: применение матричного преобразования к экранному объекту . . . . . 235

**Глава 12. Использование API рисования**
Основы использования API-интерфейса рисования . . . . . 239
Класс Graphics . . . . . 240

Создание линий и кривых . . . . . 240
Создание фигур с помощью встроенных методов . . . . . 243
Создание линий градиента и заливок . . . . . 244
Использование класса Math с методами рисования . . . . . 248
Анимирование с помощью API рисования . . . . . 249
Пример API рисования: наглядный алгоритмический генератор . . . . . 250
Расширенное использование API рисования . . . . . 253

**Глава 13. Работа с растровыми изображениями**
Основы работы с растровыми изображениями . . . . . 261
Классы Bitmap и BitmapData . . . . . 263
Управление пикселами . . . . . 265
Копирование данных растровых изображений . . . . . 268
Сжатие растровых данных . . . . . 269
Создание текстур с помощью функций шумов . . . . . 270
Прокрутка изображений . . . . . 272
Выполнение множественного отображения . . . . . 273
Пример растрового изображения: анимация вращающейся луны . . . . . 275
Асинхронное раскодирование растровых изображений . . . . . 285

**Глава 14. Фильтрация экранных объектов**
Основы фильтрации экранных объектов . . . . . 289
Создание и применение фильтров . . . . . 290
Доступные фильтры показа . . . . . 297
Пример фильтрации экранных объектов: Filter Workbench . . . . . 315

**Глава 15. Работа с шейдерами Pixel Bender**
Основные сведения о шейдерах Pixel Bender . . . . . 324
Загрузка или встраивание шейдера . . . . . 327
Получение метаданных шейдера . . . . . 328
Указание значений вводов и параметров шейдера . . . . . 329
Использование шейдера . . . . . 334

**Глава 16. Работа с фрагментами роликов**
Основные сведения о фрагментах роликов . . . . . 348
Работа с объектами MovieClip . . . . . 349
Управление воспроизведением фрагмента ролика . . . . . 349
Создание объектов MovieClip с помощью ActionScript . . . . . 352
Загрузка внешнего SWF-файла . . . . . 355
Пример фрагмента ролика: RuntimeAssetsExplorer . . . . . 357

**Глава 17. Работа с анимацией движения**
Основные сведения об анимации движения . . . . . 362
Копирование сценариев анимации движения в Flash . . . . . 363
Встраивание сценариев анимации движения . . . . . 364
Описание анимации . . . . . 365
Добавление фильтров . . . . . 368
Связывание анимации движения с ее экранными объектами . . . . . 370

**Глава 18. Работа с обратной кинематикой**

Основы обратной кинематики . . . . . 372
Обзор анимации каркасов OK . . . . . 373
Получение сведений о каркасе OK . . . . . 375
Создание экземпляра IK Mover и ограничение его движения . . . . . 376
Перемещение каркаса OK . . . . . 377
Использование пружин . . . . . 377
Использование событий OK . . . . . 378

**Глава 19. Работа в трех измерениях (3D)**

Основные сведения о трехмерных экранных объектах . . . . . 379
Трехмерные экранные объекты во Flash Player и среде выполнения AIR . . . . . 380
Создание и перемещение трехмерных экранных объектов . . . . . 382
Проецирование трехмерных объектов в двухмерное пространство . . . . . 384
Пример: перспективная проекция . . . . . 386
Выполнение сложных трехмерных преобразований . . . . . 389
Использование треугольников для создания трехмерных эффектов . . . . . 392

**Глава 20. Основы работы с текстом**

**Глава 21. Использование класса TextField**

Отображение текста . . . . . 403
Выделение и обработка текста . . . . . 407
Захват введенного текста . . . . . 409
Ограничение ввода текста . . . . . 410
Форматирование текста . . . . . 411
Расширенная визуализация текста . . . . . 415
Работа со статическим текстом . . . . . 417
Пример класса TextField: форматирование текста в газетном стиле . . . . . 419

**Глава 22. Использование Flash Text Engine**

создание и отображение текста . . . . . 428
Обработка событий в FTE . . . . . 433
Форматирование текста . . . . . 436
Работа со шрифтами . . . . . 441
Управление текстом . . . . . 443
Пример Flash Text Engine: макет новостей . . . . . 449

**Глава 23. Использование Text Layout Framework**

Обзор Text Layout Framework . . . . . 458
Использование Text Layout Framework . . . . . 460
Структурирование текста с использованием TLF . . . . . 465
Форматирование текста с использованием TLF . . . . . 469
Импорт и экспорт текста с использованием TLF . . . . . 470
Управление контейнерами текста с использованием TLF . . . . . 471
Включение выбора текста, редактирования и отмены с использованием TLF . . . . . 472
Обработка событий с использованием TLF . . . . . 472
Размещение изображений в тексте . . . . . 473

**Глава 24. Работа со звуком**

Основные сведения о работе со звуком . . . . . 474
Обзор архитектуры звука . . . . . 475
Загрузка внешних звуковых файлов . . . . . 477
Работа с встроенными звуками . . . . . 479
Работа с потоковыми аудиофайлами . . . . . 481
Работа с динамически создаваемым звуком . . . . . 482
Воспроизведение звуков . . . . . 484
Безопасность при загрузке и воспроизведении звуков . . . . . 488
Управление громкостью и панорамированием звука . . . . . 489
Работа с метаданными звука . . . . . 491
Получение необработанных аудиоданных . . . . . 491
Захват вводимого звука . . . . . 495
Пример звука: Podcast Player . . . . . 501

**Глава 25. Работа с видео**

Основы видео . . . . . 508
Сведения о форматах видео . . . . . 509
Понимание класса Video . . . . . 513
Загрузка видеофайлов . . . . . 514
Управление воспроизведением видео . . . . . 515
Воспроизведение видео в полноэкранном режиме . . . . . 517
Потоковые видеофайлы . . . . . 521
Понимание ключевых точек . . . . . 521
Написание методов обратного вызова для метаданных и ключевых точек . . . . . 522
Использование ключевых точек и метаданных . . . . . 528
Отслеживание активности NetStream . . . . . 538
Дополнительные темы о видеофайлах . . . . . 541
Пример видеоданных: Video Jukebox . . . . . 543
Использование класса StageVideo для визуализации с аппаратным ускорением . . . . . 549

**Глава 26. Работа с камерами**

Понимание класса Camera . . . . . 558
Отображение содержимого камеры на экране . . . . . 559
Разработка приложения для камеры . . . . . 559
Подключение к камере пользователя . . . . . 560
Проверка установки камер . . . . . 560
Определение разрешений для доступа к камере . . . . . 561
Обеспечение максимального качества видео с камеры . . . . . 563
Мониторинг состояния камеры . . . . . 564

**Глава 27. Цифровые средства защиты авторских прав (DRM)**

Описание технологического процесса защищенного содержимого . . . . . 567
Относящиеся к DRM элементы и события класса NetStream . . . . . 574
Использование класса DRMStatusEvent . . . . . 576
Использование класса DRMAuthenticateEvent . . . . . 577
Использование класса DRMErrorEvent . . . . . 581

Использование класса DRMManager . . . . . 581
Использование класса DRMContentData . . . . . 583
Обновление проигрывателя Flash Player для поддержки Adobe Access . . . . . 584
Внешние лицензии . . . . . 585
Поддержка домена . . . . . 587
Воспроизведение защищенного содержимого с помощью поддержки домена . . . . . 587
Предварительный просмотр . . . . . 588
Предоставление содержимого . . . . . 588
Среда Open Source Media Framework . . . . . 589

**Глава 28. Добавление PDF-содержимого в приложения AIR**
Определение возможности отображения PDF . . . . . 592
Загрузка PDF-содержимого . . . . . 593
Сценарии передачи PDF . . . . . 594
Известные ограничения PDF в AIR . . . . . 595

**Глава 29. Основы взаимодействия пользователя с системой**
Захват действий пользователя . . . . . 597
Управление фокусом . . . . . 598
Обнаружение типов ввода . . . . . 599

**Глава 30. Ввод с клавиатуры**
Захват действий клавиатуры . . . . . 602
Использование класса IME . . . . . 604
Виртуальная клавиатура . . . . . 609

**Глава 31. Действия мыши**
Захват действий мыши . . . . . 617
Пример действий мыши: WordSearch . . . . . 620

**Глава 32. Ввод с помощью прикосновения, нескольких прикосновений или жестов**
Основные сведения о вводе прикосновением . . . . . 625
Обнаружение поддержки прикосновения . . . . . 627
Обработка событий нажатия . . . . . 628
Прикосновение и перетаскивание . . . . . 632
Обработка событий жеста . . . . . 633
Устранение неполадок . . . . . 637

**Глава 33. Копирование и вставка**
Основы копирования и вставки . . . . . 641
Чтение из системного буфера обмена и запись в него . . . . . 642
Копирование и вставка HTML в AIR . . . . . 642
Форматы данных Clipboard . . . . . 644

**Глава 34. Применение акселерометра**
Проверка наличия поддержки акселерометра . . . . . 651
Обнаружение изменений акселерометра . . . . . 652

**Глава 35. Перетаскивание в AIR**

Основы перетаскивания в AIR . . . . . 654
Поддержка жеста «перетаскивания из» . . . . . 656
Поддержка жеста «перетаскивания в» . . . . . 659
Перетаскивание в HTML . . . . . 662
Перетаскивание данных из HTML-элемента . . . . . 666
Перетаскивание данных в HTML-элемент . . . . . 667
Пример: переопределение поведения по умолчанию при перетаскивании в HTML . . . . . 667
Обработка перемещения файлов в изолированной программной среде HTML вне приложений . . . . . 669
Перетаскивание предварительных образов файла . . . . . 671

**Глава 36. Работа с меню**

Основные сведения о меню . . . . . 679
Создание исходных меню (AIR) . . . . . 686
О контекстных меню в HTML (AIR) . . . . . 688
Отображение всплывающих исходных меню (AIR) . . . . . 689
Обработка событий меню . . . . . 690
Пример собственного меню: меню окна и приложения (AIR) . . . . . 691

**Глава 37. Значки панели задач AIR**

О значках панели задач . . . . . 695
Значки дока . . . . . 696
Значки области уведомлений . . . . . 697
Кнопки и значки окон на панели задач . . . . . 699

**Глава 38. Работа с файловой системой**

Использование класса FileReference . . . . . 701
Использование API-интерфейса файловой системы AIR . . . . . 716

**Глава 39. Хранение локальных данных**

Общие объекты . . . . . 754
Зашифрованная локальная система хранения данных . . . . . 764

**Глава 40. Работа с локальными базами данных SQL в AIR**

О локальных базах данных SQL . . . . . 768
Создание и изменение базы данных . . . . . 773
Операции над данными базы данных SQL . . . . . 780
Использование синхронных и асинхронных операций базы данных . . . . . 809
Использование шифрования с базами данных SQL . . . . . 814
Стратегии работы с базами данных SQL . . . . . 832

**Глава 41. Работа с байтовыми массивами**

Чтение и запись объектов ByteArray . . . . . 836
Пример объекта ByteArray: чтение zip-файла . . . . . 842

**Глава 42. Основные сведения о работе с сетью и обмене данными**

Сетевые интерфейсы . . . . . 850
Изменения сетевого подключения . . . . . 852
Записи системы доменных имен (DNS) . . . . . 854

**Глава 43. Сокеты**
TCP-сокеты . . . . . 856
UDP-сокеты (AIR) . . . . . 868
Адреса IPv6 . . . . . 870

**Глава 44. Связь HTTP**
Загрузка внешних данных . . . . . 872
Запросы веб-службы . . . . . 881
Открытие URL-адреса в другом приложении . . . . . 889

**Глава 45. Связь с другими экземплярами Flash Player и AIR**
О классе LocalConnection . . . . . 891
Обмен сообщениями между двумя приложениями . . . . . 893
Подключение к содержимому в разных доменах и к другим приложениям AIR . . . . . 896

**Глава 46. Связь с собственными процессами в AIR**
Обзор связей с собственным процессом . . . . . 898
Запуск и закрытие собственного процесса . . . . . 899
Связь с собственным процессом . . . . . 900
Вопросы безопасности при связи с собственным процессом . . . . . 902

**Глава 47. Использование внешнего API**
Основы использования внешнего API . . . . . 904
Требования к внешнему API и преимущества . . . . . 906
Использование класса ExternalInterface . . . . . 907
Пример внешнего API: связь между ActionScript и JavaScript в веб-обозревателе . . . . . 911

**Глава 48. Проверка подписи XML в среде AIR**
Основные принципы проверки XML-подписей . . . . . 918
Об XML-подписях . . . . . 923
Реализация интерфейса IURIDereferencer . . . . . 926

**Глава 49. Среда клиентской системы**
Основные сведения о среде клиентской системы . . . . . 934
Использование класса System . . . . . 935
Использование класса Capabilities . . . . . 936
Пример возможностей: определение возможностей системы . . . . . 937

**Глава 50. Запуск и завершение приложения AIR**
Вызов приложения . . . . . 942
Захват аргументов командной строки . . . . . 944
Запуск приложения AIR по имени входа пользователя . . . . . 947
Запуск приложения AIR из браузера . . . . . 948
Завершение работы приложения . . . . . 950

**Глава 51. Работа с информацией среды выполнения AIR и операционной системы**
Управление сопоставлением файлов . . . . . 952
Получение версии среды выполнения и уровня исправлений . . . . . 953

Обнаружение возможностей AIR ..... 953
Отслеживание присутствия пользователя ..... 954

**Глава 52. Работа с собственными окнами AIR**
Основные сведения о собственных окнах в AIR ..... 955
Создание окон ..... 963
Управление окнами ..... 972
Прослушивание событий окна ..... 983
Отображение окон во весь экран ..... 984

**Глава 53. Экраны дисплея в AIR**
Основные сведения об экранах дисплея в AIR ..... 987
Перечисление экранов ..... 988

**Глава 54. Печать**
Основные сведения о печати ..... 992
Печать страницы ..... 993
Задачи среды выполнения Flash и системная печать ..... 994
Настройка размера, масштаба и ориентации страницы ..... 997
Дополнительные приемы печати ..... 999
Пример печати: печать нескольких страниц ..... 1001
Пример печати: масштабирование, обрезка и динамическая настройка ..... 1003
Пример печати: настройка параметров страницы и печати ..... 1005

**Глава 55. Geolocation**
Обнаружения изменений геопозиционирования ..... 1008

**Глава 56. Интернационализация приложений**
Основы интернационализации приложений ..... 1012
Обзор пакета flash.globalization ..... 1013
Определение локали ..... 1015
Формат чисел ..... 1016
Формат валюты ..... 1019
Формат даты и времени ..... 1021
Сортировка и сравнение строк ..... 1023
Преобразование регистра ..... 1024
Пример. Интернационализация приложения, передающего котировки ценных бумаг ..... 1025

**Глава 57. Локализация приложений**
Выбор локали ..... 1030
Локализация содержимого Flex ..... 1031
Локализация содержимого Flash ..... 1031
Локализация приложений AIR ..... 1031
Локализация даты, времени и валюты ..... 1031

**Глава 58. О среде HTML**
Обзор среды HTML ..... 1034
AIR и WebKit ..... 1037

**Глава 59. Программирование HTML и JavaScript в AIR**

О классе HTMLLoader ..... 1054
Избежание ошибок безопасности в JavaScript ..... 1056
Доступ к классам API-интерфейса AIR из JavaScript ..... 1061
Об URL-адресах в AIR ..... 1063
Открытие доступа к объектам ActionScript для JavaScript ..... 1064
Доступ к объектам DOM и JavaScript из ActionScript ..... 1065
Вложение SWF-содержимого в HTML ..... 1067
Использование библиотек ActionScript на HTML-странице ..... 1070
Преобразование объектов Date и RegExp ..... 1072
Управление таблицей стилей HTML из ActionScript ..... 1072
Перекрестное выполнение сценариев в содержимом из различных изолированных программных сред безопасности 1074

**Глава 60. Написание сценария для HTML-контейнера AIR**

Свойства отображения объектов HTMLLoader ..... 1079
Прокрутка содержимого HTML ..... 1082
Доступ к истории HTML ..... 1083
Задание агента пользователя для использования при загрузке содержимого HTML ..... 1084
Задание кодировки символов для содержимого HTML ..... 1084
Определение пользовательских интерфейсов, подобных интерфейсу обозревателя, для содержимого HTML ..... 1085
Создание подклассов класса HTMLLoader ..... 1096

**Глава 61. Обработка в AIR событий HTML**

События HTMLLoader ..... 1098
Обработка событий DOM с помощью ActionScript ..... 1099
Ответ на необрабатываемые исключения JavaScript ..... 1099
Обработка событий среды выполнения с помощью JavaScript ..... 1101

**Глава 62. Отображение HTML-содержимого в мобильных приложениях**

Объекты StageWebView ..... 1104
Содержимое ..... 1105
События навигации ..... 1106
История ..... 1107
Фокус ..... 1108
Захват растрового изображения ..... 1110

**Глава 63. Использование потоков worker для параллелизма**

Общие сведения о потоках worker и параллелизме ..... 1112
Создание потоков worker и управление ими ..... 1113
Обмен данными между потоками worker ..... 1116

**Глава 64. Безопасность**

Общие сведения о безопасности платформы Flash ..... 1121
Изолированные программные среды ..... 1123
Элементы управления разрешениями ..... 1128
Ограничение API-интерфейсов сетевого подключения ..... 1137
Безопасность в полноэкранном режиме ..... 1139

Безопасность в полноэкранном интерактивном режиме ..... 1141
Загрузка содержимого ..... 1142
Перекрестное выполнение сценариев ..... 1145
Получение данных загруженного мультимедийного содержимого ..... 1149
Загрузка данных ..... 1151
Загрузка встроенного содержимого из SWF-файлов, импортированных в домен безопасности ..... 1155
Работа с устаревшим содержимым ..... 1155
Установка прав доступа LocalConnection ..... 1156
Управление доступом к внешним URL-адресам ..... 1156
Общие объекты ..... 1158
Доступ к камере, микрофону, буферу обмена и клавиатуре ..... 1160
Безопасность AIR ..... 1160

**Глава 65. Работа с примерами ActionScript**
Типы примеров ..... 1185
Работа с примерами ActionScript 3.0 во Flash Professional ..... 1187
Работа с примерами ActionScript 3.0 в Flash Builder ..... 1188
Работа с примерами ActionScript 3.0 на мобильных устройствах ..... 1189

**Глава 66. Поддержка SQL в локальных базах данных**
Поддерживаемый синтаксис SQL ..... 1195
Поддержка типов данных ..... 1218

**Глава 67. Сообщения со сведениями об ошибках SQL, идентификаторы и аргументы**

**Глава 68. Язык AGAL (Adobe Graphics Assembly Language)**
Формат байт-кода AGAL ..... 1228

# Глава 1. Работа с датами и временем

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Исчисление времени — не главное в программном приложении, но порой это ключевой фактор его качества. Язык ActionScript 3.0 предлагает эффективные способы управления календарными датами, временем и временными интервалами. За эту функциональность отвечают два основных класса: класс Date и новый класс Timer в пакете flash.utils.

Даты и время — это тип информации, часто используемый в программах ActionScript. В качестве примера возможной функциональности вам может понадобиться знать точный день недели или определить, сколько времени пользователь тратит на той или иной экран. В языке ActionScript класс Date используется для представления отдельного момента времени, включая данные о дате и времени. В экземпляре Date содержатся отдельные значения даты и времени, включая год, месяц, число, день недели, час, минуты, секунды, миллисекунды и часовой пояс. Язык ActionScript расширяет эту функциональность с помощью класса Timer, который позволяет выполнять какое-либо действие с определенной задержкой или с заданным интервалом.

### Дополнительные разделы справки

Date

flash.utils.Timer

## Управление календарными датами и временем

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Все функции управления датами и временем в ActionScript 3.0 собраны в классе верхнего уровня Date. В классе Date содержатся методы и свойства, позволяющие обрабатывать даты и время в формате UTC или в формате местного часового пояса. UTC — это стандартное время, равное гринвичскому (GMT).

### Создание объектов Date

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Из всех ключевых классов класс Date обладает одним из наиболее гибких методов конструкторов. Его можно вызвать четырьмя способами.

Во-первых, если у конструктора `Date()` нет параметров, он возвращает объект, содержащий текущие дату и время по местному часовому поясу. Вот пример:

```
var now:Date = new Date();
```

Во-вторых, если у конструктора `Date()` только один числовой параметр, то им исчисляется количество миллисекунд с 1 января 1970 г., в результате чего метод возвращает соответствующий объект Date. Обратите внимание, что значение в миллисекундах — это количество миллисекунд с 1 января 1970 г. в формате UTC. Тем не менее, объект Date выводит значения по местному времени, кроме случаев, когда используются методы вывода значений в формате UTC. При создании нового объекта Date с помощью единственного параметра milliseconds не забывайте о разнице между своим часовым поясом и UTC. Следующие инструкции создают объект Date, установленный на полночь 1 января 1970 г. в формате UTC:

```
var millisecondsPerDay:int = 1000 * 60 * 60 * 24;
// gets a Date one day after the start date of 1/1/1970
var startTime:Date = new Date(millisecondsPerDay);
```

В-третьих, в конструктор `Date()` можно передавать несколько числовых параметров. Он распознает в этих параметрах соответственно год, месяц, число, час, минуту, секунду и миллисекунду и возвращает нужный объект Date. Эти параметры ввода считаются по местному времени, а не UTC. Следующие инструкции получают объект Date, установленный на полночь 1 января 2000 г. по местному времени:

```
var millenium:Date = new Date(2000, 0, 1, 0, 0, 0, 0);
```

В-четвертых, можно передать конструктору `Date()` единую строку-параметр. Он будет пытаться разобрать эту строку на компоненты даты и времени, а затем возвратить соответствующий объект Date. При использовании этого подхода рекомендуется включать конструктор `Date()` в блок `try..catch` для отлавливания ошибок разбора. Конструктор `Date()` принимает различные форматы строк (они перечислены в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform). Следующая инструкция инициализирует новый объект Date с помощью строкового значения:

```
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");
```

Если конструктору `Date()` не удается разобрать строку-параметр, он не генерирует исключение. Тем не менее, получившийся объект Date будет содержать неверное значение даты.

## Получение значений даты и времени

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Значение разных единиц времени можно извлечь из объекта Date с помощью свойств или методов класса Date. Каждое из следующих свойств возвращает значение единицы времени из объекта Date:

- свойство `fullYear`;
- свойство `month`, обозначаемое числами от 0 до 11, где 0 — это январь, а 11 — декабрь;
- свойство `date`, обозначаемое числом календарного дня от 1 до 31;
- свойство `day`, обозначающее день недели в числовом формате, начиная с 0 (воскресенье);
- свойство `hours` в диапазоне от 0 до 23;
- свойство `minutes`;
- свойство `seconds`;
- свойство `milliseconds`.

  В действительности класс Date предлагает несколько способов получения этих значений. Например, значение месяца из объекта Date можно получить четырьмя разными методами:

- свойство `month`;
- метод `getMonth()`;

- свойство `monthUTC`;
- метод `getMonthUTC()`.

Все четыре способа одинаково эффективны, поэтому выбор будет зависеть от конкретных целей.

Перечисленные свойства представляют компоненты значения даты в целом. Например, свойство milliseconds не может быть больше 999, потому что при достижении им значения 1000 значение seconds возрастает на 1, а свойство milliseconds сбрасывается до 0.

Если вам нужно получить значение объекта Date в миллисекундах с 1 января 1970 года (UTC), можно воспользоваться методом `getTime()`. Схожий с ним метод `setTime()` позволяет изменять значение существующего объекта Date в миллисекундах с 1 января 1970 года (UTC).

## Арифметические операции с датами и временем

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

С помощью класса Date можно складывать и вычитать даты и время. Значения дат хранятся в миллисекундах, поэтому перед сложением или вычитанием необходимо перевести остальные значения в миллисекунды.

Если в вашем приложении планируется часто выполнять арифметические действия с датами и временем, вам может быть полезно создать константы, содержащие общие значения единиц времени в миллисекундах:

```
public static const millisecondsPerMinute:int = 1000 * 60;
public static const millisecondsPerHour:int = 1000 * 60 * 60;
public static const millisecondsPerDay:int = 1000 * 60 * 60 * 24;
```

Выполнять арифметические операции с датами, используя стандартные единицы времени, будет гораздо проще. В приведенном ниже коде значение даты устанавливается на один час вперед от текущего времени с помощью методов `getTime()` и `setTime()`.

```
var oneHourFromNow:Date = new Date();
oneHourFromNow.setTime(oneHourFromNow.getTime() + millisecondsPerHour);
```

Еще один способ задать значение даты и времени — создать новый объект Date только с помощью параметра milliseconds. Например, в приведенном ниже коде для вычисления одной даты к другой прибавляется 30 дней.

```
// sets the invoice date to today's date
var invoiceDate:Date = new Date();

// adds 30 days to get the due date
var dueDate:Date = new Date(invoiceDate.getTime() + (30 * millisecondsPerDay));
```

Далее, константа `millisecondsPerDay` умножается на 30 для получения 30 дней, а результат добавляется к значению `invoiceDate` и используется для установки значения `dueDate`.

## Перевод времени между часовыми поясами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Арифметические операции с датами и временем пригодятся, если вы хотите преобразовать даты одного часового пояса в другой. Это делается методом `getTimezoneOffset()`, который возвращает значение в минутах, соответствующее разнице между часовым поясом объекта Date и UTC. Он возвращает значение в минутах, потому что не все пояса отличаются строго на час: для некоторых разница с соседними поясами составляет полчаса.

В примере ниже показано, как использовать смещение между часовыми поясами для преобразования даты из местного времени в UTC. Для преобразования сначала вычисляется значение часового пояса в миллисекундах, а потом значение Date изменяется на эту величину:

```
// creates a Date in local time
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");

// converts the Date to UTC by adding or subtracting the time zone offset
var offsetMilliseconds:Number = nextDay.getTimezoneOffset() * 60 * 1000;
nextDay.setTime(nextDay.getTime() + offsetMilliseconds);
```

# Управление временными интервалами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При разработке приложений в Adobe Flash CS4 Professional у вас есть доступ к временной шкале, с помощью которой можно отслеживать покадровое выполнение приложения. Однако в «чистых» проектах ActionScript придется полагаться на другие механизмы счисления времени.

## Циклы и таймеры

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В некоторых языках программирования необходимо задавать собственные схемы счисления времени с помощью инструкций циклов типа `for` или `do..while`.

Инструкции циклов, как правило, работают с той скоростью, какую позволяет локальный компьютер, то есть на каких-то компьютерах приложения будут выполняться быстрее, на каких-то медленнее. Если приложению требуется постоянный временной интервал, его можно привязать к календарю или к часам. Во многих приложениях: играх, анимациях, счетчиках реального времени — требуются надежные механизмы отсчета, которые будут работать одинаково вне зависимости от компьютера.

Класс ActionScript 3.0 Timer предлагает для этого удачное решение. С помощью модели событий ActionScript 3.0 класс Timer отправляет событие таймера после достижения заданного временного интервала.

## Класс Timer

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Предпочтительный способ обработки функций в языке ActionScript 3.0 — использование класса Timer (flash.utils.Timer), с помощью которого можно отправлять события при достижении заданного интервала.

Для запуска таймера нужно сначала создать экземпляр класса Timer, который будет сообщать, как часто генерировать событие таймера и сколько раз это нужно делать.

Например, ниже показано, как создавать экземпляр Timer, который будет отправлять событие каждую секунду в течение 60 секунд:

```
var oneMinuteTimer:Timer = new Timer(1000, 60);
```

Объект Timer отправляет объект TimerEvent каждый раз при достижении заданного интервала. Тип события объекта TimerEvent — `timer` (определяется константой `TimerEvent.TIMER`). Объект TimerEvent содержит те же свойства, что и стандартный объект Event.

Если экземпляру Timer задано фиксированное число интервалов, он также отправит событие `timerComplete` (определяемое константой `TimerEvent.TIMER_COMPLETE`) при достижении завершающего интервала.

Вот образец небольшого приложения, в котором демонстрируется работа класса Timer:

```
package
{
    import flash.display.Sprite;
    import flash.events.TimerEvent;
    import flash.utils.Timer;

    public class ShortTimer extends Sprite
    {
        public function ShortTimer()
        {
            // creates a new five-second Timer
            var minuteTimer:Timer = new Timer(1000, 5);

            // designates listeners for the interval and completion events
            minuteTimer.addEventListener(TimerEvent.TIMER, onTick);
            minuteTimer.addEventListener(TimerEvent.TIMER_COMPLETE, onTimerComplete);

            // starts the timer ticking
            minuteTimer.start();
        }

        public function onTick(event:TimerEvent):void
        {
            // displays the tick count so far
            // The target of this event is the Timer instance itself.
            trace("tick " + event.target.currentCount);
        }

        public function onTimerComplete(event:TimerEvent):void
        {
            trace("Time's Up!");
        }
    }
}
```

При создании класса ShortTimer создается экземпляр Timer, который будет отсчитывать каждую секунду в течение пяти секунд. Затем к таймеру добавляются два прослушивателя, один из которых будет прослушивать секундные отметки, а второй — событие `timerComplete`.

Затем таймер начинает отсчет, и с этого момента метод `onTick()` выполняется с интервалом в одну секунду.

Метод `onTick()` просто выводит текущий счет секунд. По прошествии пяти секунд выполняется метод `onTimerComplete()`, сообщающий, что время вышло.

При выполнении этого примера у вас в консоли или окне трассировки должны отображаться следующие строки, по одной в секунду:

```
tick 1
tick 2
tick 3
tick 4
tick 5
Time's Up!
```

## Временные функции в пакете flash.utils

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В языке ActionScript 3.0 есть несколько временных функций, схожих с функциями ActionScript 2.0. Это функции пакетного уровня в пакете flash.utils, и работают они точно так же, как и в ActionScript 2.0.

| Функция | Описание |
|---|---|
| `clearInterval(id:uint):void` | Отменяет заданный вызов метода `setInterval()`. |
| `clearTimeout(id:uint):void` | Отменяет указанный вызов `setTimeout()`. |
| `getTimer():int` | Возвращает количество миллисекунд, прошедших с момента инициализации проигрывателя Adobe® Flash® или среды Adobe® AIR™. |
| `setInterval(closure:Function, delay:Number, ... arguments):uint` | Выполняет функцию с заданным интервалом (в миллисекундах). |
| `setTimeout(closure:Function, delay:Number, ... arguments):uint` | Выполняет указанную функцию с заданной задержкой (в миллисекундах). |

Эти функции оставлены в языке ActionScript 3.0 в целях обратной совместимости. Компания Adobe не рекомендует использовать их во вновь разрабатываемых приложениях на ActionScript 3.0. В целом, использовать в приложениях класс Timer правильнее и удобнее.

# Пример даты и времени: простые аналоговые часы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Пример простых аналоговых часов иллюстрирует эти две концепции даты и времени:

- получение текущей даты и времени и извлечение значений для часов, минут и секунд;
- использовании класса Timer для задания ритма приложению.

  Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения SimpleClock находятся в папке Samples/SimpleClock. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| SimpleClockApp.mxml<br>или<br>SimpleClockApp.fla | Основной файл приложения Flash (FLA) или Flex (MXML). |
| com/example/programmingas3/simpleclock/SimpleClock.as | Основной файл приложения. |
| com/example/programmingas3/simpleclock/AnalogClockFace.as | Рисует круглый циферблат с часовой, минутной и секундной стрелками. |

## Определение класса SimpleClock

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Пример с часами очень прост, но полезно уметь грамотно работать и с простыми приложениями, чтобы в будущем понимать, как их можно усложнить и усовершенствовать. В приложении SimpleClock используется класс SimpleClock для обработки задач запуска и отсчета времени, а также еще один класс AnalogClockFace собственно для отображения времени.

Ниже приведен код для определения и инициализации класса SimpleClock (обратите внимание, что в варианте Flash класс SimpleClock является расширением класса Sprite):

```
public class SimpleClock extends UIComponent
{
    /**
     * The time display component.
     */
    private var face:AnalogClockFace;

    /**
     * The Timer that acts like a heartbeat for the application.
     */
    private var ticker:Timer;
```

У этого класса есть два важных свойства:

- свойство `face`, являющееся экземпляром класса AnalogClockFace;
- свойство `ticker`, являющееся экземпляром класса Timer.

  Класс SimpleClock использует конструктор по умолчанию. Метод `initClock()` отвечает за установку: создает циферблат и запускает экземпляр Timer.

## Создание циферблата

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В следующих строках кода SimpleClock создается циферблат для отображения времени:

```
    /**
     * Sets up a SimpleClock instance.
     */
    public function initClock(faceSize:Number = 200)
    {
        // creates the clock face and adds it to the display list
        face = new AnalogClockFace(Math.max(20, faceSize));
        face.init();
        addChild(face);

        // draws the initial clock display
        face.draw();
```

Размер циферблата можно передать в методе `initClock()`. Если значение `faceSize` не передается, то размер по умолчанию составит 200 пикселов.

Затем приложение инициализирует циферблат и добавляет его к списку отображения с помощью метода `addChild()`, наследуемого от класса DisplayObjectContainer. Далее вызывается метод `AnalogClockFace.draw()`, который один раз отображает циферблат с текущим временем.

## Запуск таймера

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

После создания циферблата метод `initClock()` настраивает таймер:

```
// creates a Timer that fires an event once per second
ticker = new Timer(1000);

// designates the onTick() method to handle Timer events
ticker.addEventListener(TimerEvent.TIMER, onTick);

// starts the clock ticking
ticker.start();
```

Сначала этот метод создает экземпляр Timer, который будет отправлять событие каждую секунду (каждые 1000 миллисекунд). Так как конструктору `Timer()` не передается второй параметр `repeatCount`, Timer будет многократно повторяться.

Метод `SimpleClock.onTick()` будет выполняться раз в секунду при получении события `timer`:

```
public function onTick(event:TimerEvent):void
{
    // updates the clock display
    face.draw();
}
```

Метод `AnalogClockFace.draw()` просто рисует циферблат и стрелки.

## Отображение текущего времени

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Подавляющая часть кода в классе AnalogClockFace отвечает за создание на циферблате отображаемых элементов. При инициализации класса AnalogClockFace рисуется круг, внутри него рядом с каждой часовой меткой пишется число, а затем создаются три объекта Shape: один для часовой стрелки, один для минутной и один для секундной.

После запуска приложения SimpleClock каждую секунду вызывается метод `AnalogClockFace.draw()`, как показано ниже:

```
/**
 * Called by the parent container when the display is being drawn.
 */
public override function draw():void
{
    // stores the current date and time in an instance variable
    currentTime = new Date();
    showTime(currentTime);
}
```

Этот метод сохраняет значение текущего времени в переменной, чтобы время не изменилось в процессе отрисовки стрелок часов. Затем вызывается метод `showTime()`, который отображает стрелки часов, как показано ниже:

```
/**
 * Displays the given Date/Time in that good old analog clock style.
 */
public function showTime(time:Date):void
{
    // gets the time values
    var seconds:uint = time.getSeconds();
    var minutes:uint = time.getMinutes();
    var hours:uint = time.getHours();

    // multiplies by 6 to get degrees
    this.secondHand.rotation = 180 + (seconds * 6);
    this.minuteHand.rotation = 180 + (minutes * 6);

    // Multiply by 30 to get basic degrees, then
    // add up to 29.5 degrees (59 * 0.5)
    // to account for the minutes.
    this.hourHand.rotation = 180 + (hours * 30) + (minutes * 0.5);
}
```

Сначала этот метод извлекает значения часов, минут и секунд текущего времени. Затем эти значения используются для расчета угла каждой из стрелок. Так как секундная стрелка делает полный оборот за 60 секунд, ее угол каждую секунду меняется на 6 градусов (360/60). Минутная стрелка каждую минуту смещается на одинаковое количество градусов.

Часовая стрелка тоже меняет свое положение каждую минуту, чтобы отражать ход времени. Каждый час она смещается на 30 градусов (360/12), а каждую минуту — на полградуса (делим 30 градусов на 60 минут).

# Глава 2. Работа со строками

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс String содержит методы, которые позволяют работать с текстовыми строками. Строки играют важную роль при работе со многими объектами. Методы, описанные здесь, полезны при работе со строками, которые используются в таких объектах, как TextField, StaticText, XML, ContextMenu и FileReference.

Строки представляют собой последовательность символов. ActionScript 3.0 поддерживает символы ASCII и «Юникод».

### Дополнительные разделы справки

String

RegExp

parseFloat()

parseInt()

## Основные сведения о строках

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В языке программистов строка обозначает текстовое значение, то есть последовательность букв, цифр или других символов, объединенных в одно значение. Например, этот код создает переменную с типом данных String и назначает ей строковый литерал.

```
var albumName:String = "Three for the money";
```

Как показывает этот пример, в языке ActionScript текст строкового значения заключается в двойные или одинарные кавычки. Ниже приведены другие примеры строк.

```
"Hello"
"555-7649"
"http://www.adobe.com/"
```

Манипулируя любым фрагментом текста в ActionScript, вы работаете со строковым значением. В ActionScript класс String является типом данных, который можно использовать для работы с текстовыми значениями. Экземпляры String часто используются в качестве свойств, параметров методов и т.д. других классов ActionScript.

**Важные понятия и термины**

Ниже приводится справочный список важных терминов, необходимых при работе со строками.

**ASCII** Система для представления текстовых знаков и символов в компьютерных программах. Система ASCII поддерживает 26-буквенный английский алфавит и ограниченный набор дополнительных символов.

**Символ** Наименьшая единица текстовых данных (одна буква или один символ).

**Сцепление** Объединение нескольких строковых значений путем добавления одной строки в конец другой, в результате чего образуется новое строковое значение.

**Пустая строка** Строка, которая не содержит текста, пробелов или других символов и выглядит как `""`. Следует отличать пустую строку от переменной String с нулевым значением. Переменная String с нулевым значением — это переменная, которой не назначен экземпляр String, в то время как пустой строке присвоен экземпляр String со значением, не содержащим символов.

**Строка** Текстовое значение (последовательность символов).

**Строковый литерал (или «литеральная строка»)** Строковое значение, которое явно записано в коде как текстовое значение, заключенное в двойные или одинарные кавычки.

**Подстрока** Строка, которая входит в состав другой строки.

**Юникод** Стандартная система для представления текстовых знаков и символов в компьютерных программах. Система «Юникод» позволяет использовать любой символ любой письменности.

# Создание строк

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс String служит для представления строковых (текстовых) данных в языке ActionScript 3.0. Строки ActionScript поддерживают символы ASCII и «Юникод». Сроку проще всего определить с помощью строкового литерала. Чтобы объявить строковый литерал, используйте прямые двойные кавычки (`"`) или одиночные кавычки (`'`). Например, следующие две строки эквивалентны друг другу:

```
var str1:String = "hello";
var str2:String = 'hello';
```

Также объявить строку можно с помощью оператора `new`, как показано ниже.

```
var str1:String = new String("hello");
var str2:String = new String(str1);
var str3:String = new String();       // str3 == ""
```

Следующие две строки эквивалентны друг другу:

```
var str1:String = "hello";
var str2:String = new String("hello");
```

Чтобы использовать одинарные кавычки (`'`) в строковом литерале, в котором в качестве разделителей выбраны одинарные кавычки (`'`), маскируйте их символом обратной косой черты (`\`). Подобным образом, чтобы использовать двойные кавычки (`"`) в строковом литерале, в котором в качестве разделителей выбраны одинарные кавычки (`"`), маскируйте их символом обратной косой черты (`\`). Следующие две строки эквивалентны друг другу:

```
var str1:String = "That's \"A-OK\"";
var str2:String = 'That\'s "A-OK"';
```

Выбирать одинарные либо двойные кавычки следует исходя из того, какие кавычки существуют в строковом литерале, как показано ниже.

```
var str1:String = "ActionScript <span class='heavy'>3.0</span>";
var str2:String = '<item id="155">banana</item>';
```

Помните, что в языке ActionScript проводится различие между прямой одинарной кавычкой (`'`) и левой или правой одинарной кавычкой (`'` или `'`). То же относится и к двойным кавычкам. Для выделения строковых литералов используются прямые кавычки. Вставляя текст из другого источника в код ActionScript, следите за тем, чтобы использовались правильные символы.

Как показано в следующей таблице, символ обратной косой черты (`\`) можно использовать для определения других символов в строковых литералах.

| **ESC-последовательность** | **Символ** |
|---|---|
| `\b` | Возврат |
| `\f` | Новая страница |
| `\n` | Новая строка |
| `\r` | Возврат каретки |
| `\t` | Табуляция |
| `\u`***nnnn*** | Символ кодировки «Юникод» с кодом, выраженным шестнадцатеричным числом *nnnn*, например `\u263a`. |
| `\\x`***nn*** | Символ ASCII с кодом, выраженным шестнадцатеричным числом *nn* |
| `\'` | Одиночная кавычка |
| `\"` | Двойная кавычка |
| `\\` | Одна обратная косая черта |

# Свойство length

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Каждая строка имеет свойство `length`, которое представляет количество символов в строке.

```
var str:String = "Adobe";
trace(str.length);            // output: 5
```

Пустая строка и нулевая строка имеют нулевую длину, как показано в следующем примере.

```
var str1:String = new String();
trace(str1.length);           // output: 0

str2:String = '';
trace(str2.length);           // output: 0
```

# Работа с символами в строках

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Каждый символ в строке имеет позицию индекса в строке (целое число). Позиция индекса первого символа — 0. Например, в следующей строке символ `y` занимает позицию 0, а символ `w` — позицию 5.

```
"yellow"
```

Анализировать отдельные символы в различных позициях строки можно с помощью метода `charAt()` и `charCodeAt()`, как показано ниже.

```
var str:String = "hello world!";
for (var i:int = 0; i < str.length; i++)
{
    trace(str.charAt(i), "-", str.charCodeAt(i));
}
```

При выполнении этого кода возвращается следующий вывод.

```
h - 104
e - 101
l - 108
l - 108
o - 111
- 32
w - 119
o - 111
r - 114
l - 108
d - 100
! - 33
```

Также можно использовать коды символов, чтобы определить строку с помощью метода `fromCharCode()`, как в следующем коде.

```
var myStr:String = String.fromCharCode(104,101,108,108,111,32,119,111,114,108,100,33);
        // Sets myStr to "hello world!"
```

# Сравнение строк

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для сравнения строк можно использовать следующие операторы: `<`, `<=`, `!=`, `==`, `=>` и `>`. Эти инструкции можно использовать с условными выражениями `if` и `while`, как показано в следующем примере.

```
var str1:String = "Apple";
var str2:String = "apple";
if (str1 < str2)
{
    trace("A < a, B < b, C < c, ...");
}
```

Когда эти операторы используются со строками, код ActionScript рассматривает значение кода каждого символа в строке, сравнивая их слева направо, как показано ниже.

```
trace("A" < "B"); // true
trace("A" < "a"); // true
trace("Ab" < "az"); // true
trace("abc" < "abza"); // true
```

Используйте операторы `==` и `!=` для сравнения строк друг с другом и с другими типами объектов, как показано в следующем примере.

```
var str1:String = "1";
var str1b:String = "1";
var str2:String = "2";
trace(str1 == str1b); // true
trace(str1 == str2); // false
var total:uint = 1;
trace(str1 == total); // true
```

# Получение строковых представлений других объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Можно получить представление String для объекта любого типа. Для этого у всех объектов есть метод `toString()`.

```
var n:Number = 99.47;
var str:String = n.toString();
    // str == "99.47"
```

При использовании оператора сцепления `+` с комбинацией объектов String и нестроковых объектов, метод `toString()` использовать не требуется. Дополнительные сведения о сцеплении см. в следующем разделе.

Глобальная функция `String()` возвращает для данного объекта то же значение, что и для объекта, вызывающего метод `toString()`.

# Сцепление строк

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Сцеплением строк называется последовательное объединение двух строк в одну. Например, для сцепления двух строк можно использовать оператор `+`.

```
var str1:String = "green";
var str2:String = "ish";
var str3:String = str1 + str2; // str3 == "greenish"
```

Также для получение того же результата можно использовать оператор `+=`, как в следующем примере.

```
var str:String = "green";
str += "ish"; // str == "greenish"
```

Кроме того, String включает метод `concat()`, который можно использовать следующим образом.

```
var str1:String = "Bonjour";
var str2:String = "from";
var str3:String = "Paris";
var str4:String = str1.concat(" ", str2, " ", str3);
// str4 == "Bonjour from Paris"
```

Если оператор `+` (или `+=`) используется для объединения объекта String и *нестрокового* объекта, ActionScript автоматически преобразует нестроковый объект в объект String, чтобы оценить выражение, как показано ниже.

```
var str:String = "Area = ";
var area:Number = Math.PI * Math.pow(3, 2);
str = str + area; // str == "Area = 28.274333882308138"
```

Однако слагаемые можно группировать с помощью круглых скобок, чтобы предоставить контекст для оператора +, как в следующем примере.

```
trace("Total: $" + 4.55 + 1.45); // output: Total: $4.551.45
trace("Total: $" + (4.55 + 1.45)); // output: Total: $6
```

# Поиск подстрок и шаблонов в строках

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Подстроки — это последовательные символы в строке. Например, строка `"abc"` имеет следующие подстроки: `""`, `"a"`, `"ab"`, `"abc"`, `"b"`, `"bc"`, `"c"`. Для нахождения подстрок в строке можно пользоваться методами ActionScript.

В ActionScript шаблоны определяются с помощью строк или регулярных выражений. Например, следующее регулярное выражение определяет конкретный шаблон: за буквами A, B и C следует цифра (косые черты являются разделителями в регулярных выражениях).

```
/ABC\d/
```

ActionScript включает методы для поиска шаблонов в строках и для замены найденных соответствий на замещающие подстроки. Эти методы описываются в последующих разделах.

Регулярные выражения позволяют определять довольно сложные шаблоны. Дополнительные сведения см. в разделе ««Использование регулярных выражений» на странице 82».

## Поиск подстроки по позиции символа

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Методы `substr()` и `substring()` имеют много общего. Они оба возвращают подстроку заданной строки. Они оба принимают два параметра. В обоих методах первый параметр указывает позицию начального символа в данной строке. Однако в методе `substr()` вторым параметром является *length*, содержащий длину возвращаемой подстроки, а в методе `substring()` — параметр *end*, содержащий позицию символа в конце подстроки (который не включен в возвращаемую строку). В этом примере демонстрируется различие этих двух методов.

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.substr(11,15)); // output: Paris, Texas!!!
trace(str.substring(11,15)); // output: Pari
```

Метод `slice()` похож на метод `substring()`. Когда в качестве параметров заданы неотрицательные целые числа, он работает практически так же. Однако метод `slice()` может принимать в качестве параметров и отрицательные числа. В этом случае позиция символа считается от конца строки, как в следующем примере.

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.slice(11,15)); // output: Pari
trace(str.slice(-3,-1)); // output: !!
trace(str.slice(-3,26)); // output: !!!
trace(str.slice(-3,str.length)); // output: !!!
trace(str.slice(-8,-3)); // output: Texas
```

В качестве параметров метода `slice()` можно указать комбинацию из отрицательного и неотрицательного чисел.

## Поиск позиции символа соответствующей подстроки

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Методы `indexOf()` и `lastIndexOf()` можно использовать для поиска соответствующих подстрок в строке, как в следующем примере.

```
var str:String = "The moon, the stars, the sea, the land";
trace(str.indexOf("the")); // output: 10
```

Обратите внимание на то, что метод `indexOf()` учитывает регистр.

Можно задать второй параметр, чтобы указать позицию индекса в строке, с которой следует начинать поиск, как показано ниже.

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.indexOf("the", 11)); // output: 21
```

Метод `lastIndexOf()` находит последнее вхождение подстроки в строке.

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the")); // output: 30
```

Если для метода `lastIndexOf()` указать второй параметр, поиск выполняется начиная с заданной позиции в обратном порядке (справа налево).

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the", 29)); // output: 21
```

## Создание массива подстрок, разбитых на основе разделителей

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

С помощью метода `split()` можно создать массив подстрок, разбитых на основе используемых символов-разделителей. Например, строку, разделенную с помощью запятых или табуляции, можно разбить на несколько подстрок.

В следующем примере показано, как можно разбить массив на подстроки, используя в качестве разделителя амперсанд (&).

```
var queryStr:String = "first=joe&last=cheng&title=manager&StartDate=3/6/65";
var params:Array = queryStr.split("&", 2); // params == ["first=joe","last=cheng"]
```

Второй, необязательный параметр метода `split()` определяет максимальный размер возвращаемого массива.

В качестве разделителя можно также использовать регулярное выражение.

```
var str:String = "Give me\t5."
var a:Array = str.split(/\s+/); // a == ["Give","me","5."]
```

Дополнительные сведения см. в разделе ««Использование регулярных выражений» на странице 82» и в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

## Поиск шаблонов в строках и замена подстрок

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс String включает следующие методы для работы с шаблонами в строках.

- Методы `match()` и `search()` служат для поиска подстрок, соответствующих шаблону.
- Метод `replace()` служит для поиска подстрок, соответствующих шаблону, и их замены заданной подстрокой.

Эти методы описываются в последующих разделах.

Для определения шаблонов, используемых в этих методах, можно использовать строки или регулярные выражения. Дополнительные сведения о регулярных выражениях см. в разделе ««Использование регулярных выражений» на странице 82».

### Поиск соответствующих подстрок

Метод `search()` возвращает позицию индекса первой подстроки, соответствующей заданному шаблону, как в следующем примере.

```
var str:String = "The more the merrier.";
// (This search is case-sensitive.)
trace(str.search("the")); // output: 9
```

Для определения шаблона соответствия можно использовать регулярные выражения, как показано ниже.

```
var pattern:RegExp = /the/i;
var str:String = "The more the merrier.";
trace(str.search(pattern)); // 0
```

Метод `trace()` выдает значение 0, так как первый символ в строе имеет позицию индекса 0. В регулярном выражении устанавливается флаг `i`, чтобы при поиске не учитывался регистр.

Метод `search()` находит только одно соответствие и возвращает его начальную позицию индекса, даже если в регулярном выражении задан флаг `g` (global).

В следующем примере показано более сложное регулярное выражение, позволяющее найти соответствие строки, заключенной в двойные кавычки.

```
var pattern:RegExp = /"[^"]*"/;
var str:String = "The \"more\" the merrier.";
trace(str.search(pattern)); // output: 4

str = "The \"more the merrier.";
trace(str.search(pattern)); // output: -1
// (Indicates no match, since there is no closing double quotation mark.)
```

Метод `match()` работает подобным образом. Он выполняет поиск соответствующей подстроки. Однако, когда в шаблоне регулярного выражения установлен флаг global, как в следующем примере, метод `match()` возвращает массив соответствующих значений.

```
var str:String = "bob@example.com, omar@example.org";
var pattern:RegExp = /\w*@\w*\.[org|com]+/g;
var results:Array = str.match(pattern);
```

Массиву `result` присваивается следующее значение:

```
["bob@example.com","omar@example.org"]
```

Дополнительные сведения о регулярных выражениях см. в разделе ««Использование регулярных выражений» на странице 82».

### Замена совпавших подстрок

С помощью метода `replace()` можно найти нужный шаблон в строке и заменить соответствия заданной замещающей строкой, как в следующем примере.

```
var str:String = "She sells seashells by the seashore.";
var pattern:RegExp = /sh/gi;
trace(str.replace(pattern, "sch")); //sche sells seaschells by the seaschore.
```

Обратите внимание, что в этом примере в совпавших строках не учитывается регистра, так как в регулярном выражении установлен флаг `i` (`ignoreCase`), и делается несколько замен, так как установлен флаг `g` (`global`). Дополнительные сведения см. в разделе ««Использование регулярных выражений» на странице 82».

В замещающую строку можно включить следующие замещающие коды с `$`. Вместо замещающего кода с `$` вставляется замещающий текст, приведенный в следующей таблице.

| Код с $ | Замещающий текст |
|---|---|
| `$$` | $ |
| `$&` | Совпавшая подстрока. |
| `$` ` | Часть строки, предшествующая совпавшей подстроке. В этом коде используется прямая левая одинарная кавычка (`), а не прямая одинарная кавычка (') или левая фигурная кавычка (‘). |
| `$'` | Часть строки, которая предшествует совпавшей подстроке. Этот код использует прямую одиночную кавычку (').. |
| $*n* | Зарегистрированное совпадение группы в скобках *n*, где n — это цифра от 1 до 9, а за $n не следует десятичное число. |
| $*nn* | Зарегистрированное совпадение группы в скобках *nn*, где *nn* десятичное двузначное число (от 01 до 99). Если запись *nn* не определена, в качестве замещающего текста используется пустая строка. |

Например, ниже показано применение замещающих кодов `$2` и `$1`, представляющих первую и вторую зарегистрированные совпавшие группы:

```
var str:String = "flip-flop";
var pattern:RegExp = /(\w+)-(\w+)/g;
trace(str.replace(pattern, "$2-$1")); // flop-flip
```

Также можно использовать функцию в качестве второго параметра метода `replace()`. Совпавший текст заменяется возвращенным значением функции.

```
var str:String = "Now only $9.95!";
var price:RegExp = /\$([\d,]+.\d+)+/i;
trace(str.replace(price, usdToEuro));

function usdToEuro(matchedSubstring:String,  capturedMatch1:String,  index:int,
str:String):String
{
    var usd:String = capturedMatch1;
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.853690;
    var euro:Number = parseFloat(usd) * exchangeRate;
    const euroSymbol:String = String.fromCharCode(8364);
    return euro.toFixed(2) + " " + euroSymbol;
}
```

Когда в качестве второго параметра метода `replace()` используется функция, ей передаются следующие аргументы:

- Совпадающая часть строки.
- Зарегистрированные совпадения группы. Количество переданных таким образом аргументов варьируется в зависимости от количества совпадений в скобках. Количество совпадений в скобках можно определить, проверив значение параметра `arguments.length - 3` в коде функции.
- Позиция в индексе в строке, где начинается совпадение.
- Строка полностью.

# Перевод строк в верхний или нижний регистр

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Как показано на следующем примере, методы`toLowerCase()` и `toUpperCase()` переводят буквенные символы строки в нижний или верхний регистр соответственно.

```
var str:String = "Dr. Bob Roberts, #9."
trace(str.toLowerCase()); // dr. bob roberts, #9.
trace(str.toUpperCase()); // DR. BOB ROBERTS, #9.
```

После выполнения этих методов исходная строка остается без изменений. Чтобы преобразовать исходную строку, используйте следующий код:

```
str = str.toUpperCase();
```

Эти методы поддерживают расширенные символы, а не только a–z и A–Z.

```
var str:String = "José Barça";
trace(str.toUpperCase(), str.toLowerCase()); // JOSÉ BARÇA josé barça
```

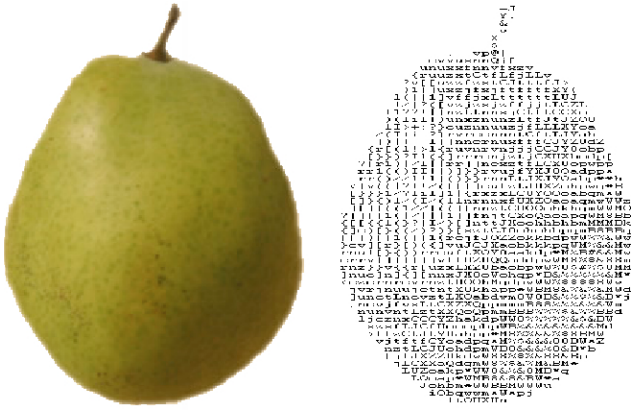
# Пример строк: ASCII-графика

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Этот пример ASCII-графики демонстрирует ряд возможностей работы с классом String в ActionScript 3.0, включая следующее.

- Метод `split()` класса String используется для получения значений из строки, разделенной символами (графическая информация в файле с разделителем-табуляцией).
- Чтобы написать с большой буквы каждое слово в названиях изображений, используется несколько приемов манипуляции со строками, включая метод `split()`, сцепление и извлечение части строки с помощью методов `substring()` и `substr()`.
- Метод `getCharAt()` используется для получения одного символа из строки (чтобы определить символ ASCII, соответствующий значению растрового изображения в оттенках серого).
- Сцепление строк используется, чтобы построить ASCII-представление изображения символ за символом.

Термином *ASCII-графика* называется текстовое представление изображения, в котором рисунок выстраивается в виде сетки символов моноширинного шрифта, такого как Courier New. Ниже приводится пример ASCII-графики, созданный приложением.



*Версия изображения, созданного с помощью ASCII-графики, находится справа.*

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения ASCIIArt находятся в папке Samples/AsciiArt. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| AsciiArtApp.mxml<br>или<br>AsciiArtApp.fla | Основной файл приложения Flash (FLA) или Flex (MXML) |
| com/example/programmingas3/asciiArt/AsciiArtBuilder.as | Класс, обеспечивающий основные функции приложения, включая извлечение метаданных изображения из текстового файла, загрузку изображений, управление процессом преобразования изображения в текст. |
| com/example/programmingas3/asciiArt/BitmapToAsciiConverter.as | Класс, обеспечивающий метод `parseBitmapData()` для преобразования данных изображения в версию класса String. |
| com/example/programmingas3/asciiArt/Image.as | Класс, представляющий загруженное растровое изображение. |
| com/example/programmingas3/asciiArt/ImageInfo.as | Класс, представляющий метаданные для изображения ASCII-графики (например, название, URL-адрес файла с изображением и т.д.). |
| image/ | Папка, содержащая изображения, используемые приложением. |
| txt/ImageData.txt | Текстовый файл с разделителем-табуляцией, содержащий сведения об изображениях, предназначенных для загрузки приложением. |

## Получение значений, разделенных табуляцией

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В этом примере используется общепринятая практика, при которой данные приложения хранятся отдельно от самого приложения. Таким образом, если данные изменяются (например, при добавлении другого изображения или при изменении названия изображения), не требуется повторно создавать SWF-файл. В этом случае метаданные изображения, включая название, URL-адрес файла с изображением и некоторые значения, используемые для манипуляций с изображением) хранятся в текстовом файле (файл txt/ImageData.txt в проекте). Ниже приводится содержимое текстового файла:

```
FILENAMETITLEWHITE_THRESHHOLDBLACK_THRESHHOLD
FruitBasket.jpgPear, apple, orange, and bananad810
Banana.jpgA picture of a bananaC820
Orange.jpgorangeFF20
Apple.jpgpicture of an apple6E10
```

Файл использует специальный формат с разделителем-табуляцией. Первая строка является строкой заголовка. Остальные строки содержат следующие денные для каждого растрового изображения, предназначенного для загрузки:

- имя файла с растровым изображением;
- отображаемое имя растрового изображения;
- пороговые значения для белого и черного цветов в растровом изображении; это шестнадцатеричные значения, выше и ниже которых пиксел считается полностью белым или черным соответственно.

Как только запускается приложение, класс AsciiArtBuilder загружает и анализирует содержимое текстового файла, чтобы создать «стопку» изображений для показа с использованием следующего кода из метода `parseImageInfo()` класса AsciiArtBuilder:

```
var lines:Array = _imageInfoLoader.data.split("\n");
var numLines:uint = lines.length;
for (var i:uint = 1; i < numLines; i++)
{
    var imageInfoRaw:String = lines[i];
    ...
    if (imageInfoRaw.length > 0)
    {
        // Create a new image info record and add it to the array of image info.
        var imageInfo:ImageInfo = new ImageInfo();

        // Split the current line into values (separated by tab (\t)
        // characters) and extract the individual properties:
        var imageProperties:Array = imageInfoRaw.split("\t");
        imageInfo.fileName = imageProperties[0];
        imageInfo.title = normalizeTitle(imageProperties[1]);
        imageInfo.whiteThreshold = parseInt(imageProperties[2], 16);
        imageInfo.blackThreshold = parseInt(imageProperties[3], 16);
        result.push(imageInfo);
    }
}
```

Все содержимое текстового файла добавлено в один экземпляр String, в свойство `_imageInfoLoader.data`. С помощью метода `split()`, в качестве параметра которого используется символ новой строки (`"\n"`) экземпляр String разбивается в массив (`lines`), элементы которого представляют собой отдельные строки текстового файла. Затем код использует цикл для обработки каждой строки (кроме первой, так как она включает только заголовки, а не содержимое). Внутри цикла еще раз используется метод `split()`, чтобы разбить содержимое одной строки с целью получения группы значений (объект Array с именем `imageProperties`). В этом случае в качестве параметра метода `split()` используется символ табуляции (`"\t"`), так как значения в каждой строке разделяются символами табуляции.

## Использование методов класса String для нормализации заголовков изображений

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Одним из решений дизайна для этого приложения является отображение названий изображений с использованием стандартного формата англоязычных заголовков, в котором каждое слово пишется с большой буквы (за исключением немногих слов, на которые это правило не распространяется). Не ожидая, что текстовый файл содержит заголовки в правильном формате, приложение форматирует их в момент получения из текстового файла.

В предыдущем коде в процессе получения метаданных отдельного изображения используется следующая строка:

```
imageInfo.title = normalizeTitle(imageProperties[1]);
```

В этом коде заголовок изображения из текстового файла передается методу `normalizeTitle()` и только после этого сохраняется в объекте ImageInfo.

```
private function normalizeTitle(title:String):String
{
    var words:Array = title.split(" ");
    var len:uint = words.length;
    for (var i:uint; i < len; i++)
    {
        words[i] = capitalizeFirstLetter(words[i]);
    }

    return words.join(" ");
}
```

Этот метод вызывает метод `split()` для деления заголовка на несколько слов (разделенных пробелом), передает каждое слово методу `capitalizeFirstLetter()`, а затем использует метод `join()` класса Array для повторного объединения отдельных слов в строку.

Метод `capitalizeFirstLetter()` делает первую букву слова заглавной.

```
    /**
     * Capitalizes the first letter of a single word, unless it's one of
     * a set of words that are normally not capitalized in English.
     */
    private function capitalizeFirstLetter(word:String):String
    {
        switch (word)
        {
            case "and":
            case "the":
            case "in":
            case "an":
            case "or":
            case "at":
            case "of":
            case "a":
                // Don't do anything to these words.
                break;
            default:
                // For any other word, capitalize the first character.
                var firstLetter:String = word.substr(0, 1);
                firstLetter = firstLetter.toUpperCase();
                var otherLetters:String = word.substring(1);
                word = firstLetter + otherLetters;
        }
        return word;
    }
```

В английском языке в заголовках *не* пишутся с заглавной буквы следующие слова: «and», «the», «in», «an», «or», «at», «of» и «a». (Это упрощенная версия правила.) Следуя этой логике, код сначала использует инструкцию `switch`, чтобы проверить, не является ли слово исключением и его не нужно писать с заглавной буквы. Если это так, код просто покидает инструкцию `switch`. С другой стороны, если слово должно быть написано с заглавной буквы, выполняются следующие действия.

1. Первая буква слова извлекается с помощью метода `substr(0, 1)`, который получает подстройку с начальным индексом символа 0 (первая буква в строке, на что указывает первый параметр `0`). Подстрока будет содержать один символ (о чем говорит второй параметр `1`).
2. Этот символ переводится в верхний регистр с помощью метода `toUpperCase()`.
3. Остальные символы исходного слова извлекаются с помощью метода `substring(1)`, который получает подстроку начиная с индекса 1 (вторая буква) до конца строки (на что указывает отсутствие второго параметра в методе `substring()`).
4. Окончательная версия слова создается путем объединения заглавной первой буквы с остальными посредством сцепления строк: `firstLetter + otherLetters`.

## Создание текста ASCII-графики

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс BitmapToAsciiConverter обеспечивает возможность преобразования растрового изображения в текстовое представление с использованием символов ASCII. Этот процесс выполняется методом `parseBitmapData()`, фрагмент которого представлен ниже.

```
var result:String = "";

// Loop through the rows of pixels top to bottom:
for (var y:uint = 0; y < _data.height; y += verticalResolution)
{
    // Within each row, loop through pixels left to right:
    for (var x:uint = 0; x < _data.width; x += horizontalResolution)
    {
        ...

        // Convert the gray value in the 0-255 range to a value
        // in the 0-64 range (since that's the number of "shades of
        // gray" in the set of available characters):
        index = Math.floor(grayVal / 4);
        result += palette.charAt(index);
    }
    result += "\n";
}
return result;
```

Сначала код определяет экземпляр String с именем `result`, который будет использоваться для построения ASCII-графики для растрового изображения. Затем выполняется циклическая обработка отдельных пикселов исходного растрового изображения. Используя несколько приемов манипуляции с цветами (опущенных здесь для краткости), код преобразует значения красного, зеленого и синего цветов отдельного пиксела в одно значение оттенка серого (число от 0 до 255). После этого код делит это значение на 4 (как показано в примере), чтобы преобразовать его в диапазон 0-63, и сохраняет полученное число в переменной `index`. (Используется диапазон от 0 до 63, так как «палитра» доступных символов ASCII, используемых этим приложением, составляет 64 значения.) Палитра символов определяется как экземпляр String в классе BitmapToAsciiConverter.

```
// The characters are in order from darkest to lightest, so that their
// position (index) in the string corresponds to a relative color value
// (0 = black).
private static const palette:String =
"@#$%&8BMW*mwqpdbkhaoQ0OZXYUJCLtfjzxnuvcr[]{}1()|/?Il!i><+_~-;,. ";
```

Переменная `index` определяет, какой символ ASCII соответствует текущему пикселу растрового изображения, и требуемый символ извлекается из экземпляра String с именем `palette` с помощью метода `charAt()`. Затем он добавляется в экземпляр String с именем `result` с помощью оператора сложения с присвоением (`+=`). Кроме того, в конце каждого ряда пикселов к экземпляру `result` добавляется символ новой строки, чтобы перенести строку и начать новый ряд символов-«пикселов».

# Глава 3. Работа с массивами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Массивы позволяют хранить множество значений в единой структуре данных. В простых индексных массивах значения сохраняются с помощью фиксированных порядковых целочисленных индексов, а в сложных ассоциативных массивах — с помощью произвольно задаваемых ключей. Массивы также могут иметь несколько измерений и содержать элементы, которые сами являются массивами. Наконец, для массива, элементы которого являются экземплярами одного типа данных, можно использовать Vector.

### Дополнительные разделы справки

Array

Vector

## Основные сведения о массивах

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В программировании нередко приходится работать не с одним объектом, а с группой элементов. Например, в приложении для воспроизведения музыки может потребоваться создать список песен, ожидающих своей очереди. Было бы затруднительно создавать отдельную переменную для каждой песни в списке. Желательно организовать все объекты Song в одну группу и работать с ними как с одним целым.

Массив — это элемент программирования, который выполняет функцию контейнера для набора элементов, такого как список песен. Чаще всего все элементы в массиве являются экземплярами одного класса, но это не является обязательным условием в ActionScript. Отдельные объекты в массиве называются его *элементами*. Массив можно представить себе как картотеку для переменных. Переменные можно добавлять в качестве элементов массива, наподобие добавления карточки в картотеку. Массив можно обрабатывать как одну переменную (как будто переставить всю картотеку в другое место). Переменные можно обрабатывать как группу (как будто искать нужную информацию, перебирая карточки). Также можно обращаться к ним в индивидуальном порядке (как будто открыть картотеку и достать из нее одну карточку).

Допустим, что вы создаете музыкальный проигрыватель, в котором пользователь может выбрать несколько песен и добавить их в список воспроизведения. В коде ActionScript есть метод `addSongsToPlaylist()`, который принимает в качестве параметра один массив. Независимо от количества песен, которые нужно добавить в список (мало, много или всего одну), можно вызвать метод `addSongsToPlaylist()` только один раз, передав ему массив с объектами Song. Внутри метода `addSongsToPlaylist()` можно использовать цикл, который будет по одному проходить элементы массива (песни) и фактически добавлять их в список воспроизведения.

Самым распространенным типом массива ActionScript является *индексный массив*. В индексном массиве каждый элемент хранится в пронумерованном слоте (так называемом *индексе*). При обращении к элементам этот номер используется в качестве адреса. Индексные массивы подходят для большинства потребностей программирования. Класс Array — это один общий класс, используемый для представления индексного массива.

Нередко индексный массив используется для хранения нескольких элементов одного типа (объектов, являющихся экземплярами одного класса). Класс Array не имеет средств для ограничения типа содержащихся в нем элементов. Класс Vector представляет собой тип индексного массива, в котором все элементы принадлежат одному типу. Используя экземпляр Vector вместо экземпляра Array, можно повысить производительность и получить другие преимущества. Класс Vector доступен для использования, начиная с Flash Player версии 10 и Adobe AIR версии 1.5.

Особым случаем использования индексного массива является *многомерный массив*. Многомерный массив — это индексный массив, элементами которого являются индексные массивы (которые, в свою очередь, содержат другие элементы).

Другим типом массива является *ассоциативный массив*, который для идентификации отдельных элементов использует строковый *ключ* а не числовой индекс. Наконец, язык ActionScript 3.0 также включает класс Dictionary, который представляет собой *словарь*. Словарь — это массив, который позволяет использовать любой тип объекта в качестве ключа для идентификации элементов.

### Важные понятия и термины

Ниже приводится список важных терминов, встречающихся в этой главе и использующихся при программировании приложений, которые оперируют массивами и векторами:

**Массив** Объект, который служит контейнером для группировки нескольких объектов.

**Оператор доступа к массиву ([])** Пара квадратных скобок, в которые заключен индекс или ключ, являющийся уникальным идентификатором элемента массива. Этот синтаксис используется после имени переменной массива, чтобы указать не весь массив, а один из его элементов.

**Ассоциативный массив** Массив, использующий строковые ключи для идентификации отдельных элементов.

**Базовый тип** Тип данных для объектов, которые могут храниться в экземпляре Vector.

**Словарь** Массив, элементы которого состоят из пар объектов «ключ-значение». Ключ используется для идентификации одного объекта вместо числового индекса.

**Элемент** Один элемент в массиве.

**Указатель** Числовой «адрес», используемый для идентификации одного элемента в индексном массиве.

**Индексный массив** Стандартный тип массива, который сохраняет каждый элемент в пронумерованной позиции и использует эти номера (индексы) для идентификации отдельных элементов.

**Клавиша** Строка или объект, используемый для идентификации одного элемента в ассоциативном массиве или словаре.

**Многомерный массив** Массив, содержащий элементы, которые являются массивами, а не одиночными значениями.

**T** Стандартное сокращение, которое в данной документации обозначает базовый тип экземпляра Vector независимо от того, чем он представлен. Сокращение T служит для представления имени класса, как показано в описании параметра Type. («T» обозначает «тип», как в словосочетании «тип данных»).

**Параметр типа** Синтаксис, который используется с именем класса Vector, чтобы указать его базовый тип (тип данных для сохраняемых объектов). Синтаксис содержит точку (`.`), затем следует имя типа данных в угловых скобках (`<>`). В целом параметр типа выглядит так: `Vector.<T>`. В данной документации класс, указанный в параметре типа, представлен в общем виде: `T`.

**Вектор** Тип массива, все элементы которого относятся к одному типу данных.

# Индексные массивы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В индексных массивах хранится последовательность из одного или более значений, организованных таким образом, чтобы каждое значение было доступно через целое число без знака. Первый индекс всегда равен 0, а затем для каждого последующего элемента массива он увеличивается на 1. В ActionScript 3.0 в качестве индексных массивов используются два класса: Array и Vector.

В индексных массивах в качестве индекса используется целое 32-разрядное число без знака. Максимальный размер индексного массива равен $2^{32}$ - 1 или 4 294 967 295. При попытке создать массив, превышающий максимально допустимый размер, произойдет ошибка при выполнении.

Для обращения к отдельному элементу индексного массива служит оператор доступа к массиву (`[]`), который позволяет указать позицию индекса для требуемого элемента. Например, следующий код представляет первый элемент (с индексом 0) в индексном массиве с именем `songTitles`.

```
songTitles[0]
```

Имя переменной массива, за которым следует индексом в квадратных скобках, выполняет роль уникального идентификатора элемента. (Другими словами, его можно использовать так же, как имя переменной.) Элементу индексного массива можно присвоить значение, используя имя и индекс в левой части инструкции присвоения.

```
songTitles[1] = "Symphony No. 5 in D minor";
```

Подобным образом можно получить значение элемента индексного массива, используя имя и индекс в правой части инструкции присвоения.

```
var nextSong:String = songTitles[2];
```

Также в квадратных скобках можно использовать переменную вместо конкретного значения. (Переменная должна содержать неотрицательное целое число, например беззнаковое целое число, положительное целое число или экземпляр Number, представленный положительным целым числом.) Этот прием обычно используется для циклического прохождения элементов индексного массива и выполнения операции над некоторыми или всеми элементами. Следующий код демонстрирует этот прием. Код создает цикл для обращения к каждому значению в объекте Array с именем `oddNumbers`. Инструкция `trace()` используется для вывода каждого значения в следующем виде: «oddNumber[*index*] = *value*».

```
var oddNumbers:Array = [1, 3, 5, 7, 9, 11];
var len:uint = oddNumbers.length;
for (var i:uint = 0; i < len; i++)
{
    trace("oddNumbers[" + i.toString() + "] = " + oddNumbers[i].toString());
}
```

### Класс Array

Первый тип индексного массива представлен классом Array. Экземпляр Array может содержать значение данных любого типа. Один объект Array может содержать объекты, представленные разными типами данных. Например, один экземпляр Array может содержать значение String в позиции индекса 0, экземпляр Number в позиции 1 и объект XML в позиции 2.

**Класс Vector**

Другой тип индексного массива, поддерживаемый в ActionScript 3.0, представлен классом Vector. Экземпляр Vector — это *типизированный массив*, то есть все элементы в экземпляре Vector всегда имеют одинаковый тип данных.

***Примечание.*** *Класс Vector доступен для использования, начиная с Flash Player версии 10 и Adobe AIR версии 1.5.*

Когда объявляется переменная Vector или создается экземпляр Vector, необходимо четко указать тип данных для объектов, которые может содержать Vector. Указанный тип данных называется *базовым типом* объекта Vector. Во время выполнения и компиляции (в строгом режиме) проверяется любой код, задающий или получающий значение элемента Vector. Если тип данных добавляемого или получаемого объекта не соответствует базовому типу Vector, выдается ошибка.

Помимо ограничения, связанного с типом данных, класс Vector имеет другие ограничения, которые отличают его от класса Array:

- Вектор принадлежит к числу плотных массивов. Объект Array может иметь значения в позициях индекса 0 и 7, даже если нет значений в позициях с 1 по 6. Экземпляр Vector, напротив, должен иметь определенное значение (или `null`) в каждой позиции индекса.
- По желанию можно ограничить длину объекта Vector. Это значит, что количество элементов в массиве Vector не может изменяться.
- Доступ к элементам вектора осуществляется с контролем границ. Вам никогда не удастся прочесть значение с индексом, превышающим индекс финального элемента (`length` – 1). Вам также не удастся задать значение, индекс которого превышает текущий последний индекс (иначе говоря, значение можно присваивать только с существующим индексом или с индексом `[length]`).

В результате этих ограничений экземпляр Vector имеет три основных преимущества перед экземпляром Array, все элементы которого представляют собой экземпляры одного класса:

- Производительность: доступ к массиву и итерация осуществляются гораздо быстрее при использовании экземпляра Vector, чем при использовании экземпляра Array.
- Безопасность типа: в строгом режиме компилятор может выявлять ошибки, связанные с типом данных. К таким ошибкам относится присвоение экземпляру Vector значения неверного типа данных или попытка получить неверный тип данных при считывании значения из объекта Vector. Во время выполнения типы данных также проверяются при записи или чтении данных в объекте Vector. Однако следует учесть, что при использовании метода `push()` или `unshift()` для добавления значений в вектор типы данных для аргументов не проверяются во время компиляции. Тем не менее, при использовании этих методов значения проверяются во время выполнения.
- Надежность: проверка диапазона во время выполнения (или проверка фиксированной длины) значительно увеличивает надежность по сравнению с экземплярам Array.

Если опустить дополнительные ограничения и преимущества, класс Vector очень похож на класс Array. Свойства и методы объекта Vector похожи (и в большинстве случаев идентичны) свойствам и методам объекта Vector. В большинстве случаев, когда используется объект Array, все элементы которого относятся к одному и тому же типу данных, предпочтительным является экземпляр Vector.

## Создание массивов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Создавать экземпляры Array и Vector можно разными способами. Однако приемы, используемые для создания массива каждого типа, немного отличаются.

## Создание экземпляра Array

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы создать объект Array, нужно вызвать конструктор `Array()` или использовать синтаксис литерала Array.

Существует три способа использования функции конструктора `Array()`. Во-первых, можно вызвать конструктор без аргументов для создания пустого массива. Свойство `length` класса Array используется для проверки отсутствия элементов в массиве. Например, приведенный ниже код вызывает конструктор `Array()` без аргументов:

```
var names:Array = new Array();
trace(names.length); // output: 0
```

Во-вторых, если число является единственным параметром, который передается конструктору `Array()`, создается массив именно этой величины, а каждому элементу задается значение `undefined`. Аргументом должно быть целое число без знака от 0 до 4 294 967 295. Например, приведенный ниже код вызывает конструктор `Array()`, единственным аргументом которого является число:

```
var names:Array = new Array(3);
trace(names.length); // output: 3
trace(names[0]); // output: undefined
trace(names[1]); // output: undefined
trace(names[2]); // output: undefined
```

В-третьих, если вызвать конструктор и передать в качестве параметров список элементов, будет создан массив, элементы которого будут соответствовать этим параметрам. Приведенный ниже код передает три аргумента конструктору `Array()`:

```
var names:Array = new Array("John", "Jane", "David");
trace(names.length); // output: 3
trace(names[0]); // output: John
trace(names[1]); // output: Jane
trace(names[2]); // output: David
```

Массивы также можно создавать с помощью литералов Array. Литерал Array можно назначить непосредственно переменной массива, как показано ниже:

```
var names:Array = ["John", "Jane", "David"];
```

## Создание экземпляра Vector

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Чтобы создать экземпляр Vector, нужно вызвать конструктор `Vector.<T>()`. Также для создания объекта Vector можно вызвать глобальную функцию `Vector.<T>()`. Эта функция преобразует указанный объект в экземпляр Vector. В Flash Professional CS5 и более поздних версиях, Flash Builder 4 и более поздних версиях, а также в Flex 4 и более поздних версиях экземпляр Vector можно создавать с помощью буквенного синтаксиса Vector.

Каждый раз при объявлении переменной Vector (а также параметра метода или типа возвращаемых данных метода класса Vector) необходимо указывать ее базовый тип. Кроме того, базовый тип необходимо указывать при создании экземпляра Vector с помощью конструктора `Vector.<T>()`. Другими словами, каждый раз при использовании термина `Vector` в коде ActionScript он сопровождается указанием базового типа.

Базовый тип указывается с помощью синтаксиса параметра типа. В коде параметр типа следует непосредственно за словом `Vector`. Он состоит из точки ( . ) и имени базового класса, заключенного в угловые скобки (`<>`), как показано в следующем примере:

```
var v:Vector.<String>;
v = new Vector.<String>();
```

В первой строке примера переменная `v` объявлена как экземпляр `Vector.<String>`. Иначе говоря, он представляет индексный массив, который может содержать только экземпляры класса String. Вторая строка вызывает конструктор `Vector()` для создания экземпляра того же типа Vector (то есть массив Vector, все элементы которого являются объектами String). Этот объект присваивается переменной `v`.

### Использование глобальной функции Vector.<T>()

Если конструктор `Vector.<T>()` используется без аргументов, он создает пустой экземпляр Vector. Чтобы убедиться в том, что экземпляр Vector пуст, нужно проверить свойство `length`. Например, следующий код вызывает конструктор `Vector.<T>()` без аргументов.

```
var names:Vector.<String> = new Vector.<String>();
trace(names.length); // output: 0
```

Если заранее известно, сколько элементов изначально должен иметь экземпляр Vector, можно предварительно определить количество элементов в Vector. Чтобы создать экземпляр Vector с требуемым числом элементов, передайте это число в качестве первого параметра (свойство `length`). Так как элементы объекта Vector не могут быть пустыми, они заполняются экземплярами базового типа. Если базовый тип является ссылочным типом, который допускает значения `null`, то все элементы будут содержать значения `null`. В противном случае все элементы содержат значения по умолчанию для базового класса. Например, переменная, выраженная беззнаковым целым числом, не может иметь значение `null`. Поэтому в следующем коде экземпляр Vector с именем `ages` создается с семью элементами, которые имеют значение 0.

```
var ages:Vector.<uint> = new Vector.<uint>(7);
trace(ages); // output: 0,0,0,0,0,0,0
```

И, наконец, с помощью конструктора `Vector.<T>()` также можно создавать объекты Vector с фиксированной длиной, передав значение `true` для второго параметра (свойство `fixed`). В этом случае экземпляр Vector создается с указанным числом элементов, которое нельзя изменить. Однако следует учесть, что при этом все равно можно изменять значения элементов объекта Vector с фиксированной длиной.

### Использование конструктора Vector с буквенным синтаксисом

В Flash Professional CS5 и более поздних версиях, Flash Builder 4 и более поздних версиях, а также в Flex 4 и более поздних версиях можно передать список значений конструктору `Vector.<T>()`, чтобы задать начальные значения элементов экземпляра Vector:

```
// var v:Vector.<T> = new <T>[E0, ..., En-1 ,];
// For example:
var v:Vector.<int> = new <int>[0,1,2,];
```

К данному синтаксису относится следующая информация.

- Запятую в конце ставить не обязательно.
- Пустые элементы в массиве не поддерживаются; инструкция, такая как `var v:Vector.<int> = new <int>[0,,2,]` выдает ошибку компилятора.
- Для экземпляра Vector нельзя задать длину по умолчанию. Его длина равна числу элементов в списке инициализации.
- Для экземпляра Vector нельзя задать фиксированную длину. Вместо этого следует использовать свойство `fixed`.
- Использование значений, не соответствующих заданному типу, может привести к ошибкам или потере данных. Например:

```
var v:Vector.<int> = new <int>[4.2]; // compiler error when running in strict mode
trace(v[0]); //returns 4 when not running in strict mode
```

**Использование глобальной функции Vector.<T>()**

Помимо конструктора `Vector.<T>()` и конструкторов Vector с буквенным синтаксисом, для создания объекта Vector также можно использовать глобальную функцию `Vector.<T>()`. Глобальная функция `Vector.<T>()` представляет собой функцию преобразования. Вызывая глобальную функцию `Vector.<T>()`, нужно указать базовый тип объекта Vector, возвращаемого методом. В качестве аргумента передается один индексный массив (экземпляр Array или Vector). Затем метод возвращает объект Vector с заданным базовым типом, содержащий значения в аргументе исходного массива. В следующем коде показан синтаксис для вызова глобальной функции `Vector.<T>()`.

```
var friends:Vector.<String> = Vector.<String>(["Bob", "Larry", "Sarah"]);
```

Глобальная функция `Vector.<T>()` выполняет преобразования типа данных на двух уровнях. Во-первых, когда функции передается экземпляр Array, возвращается экземпляр Vector. Во-вторых, независимо от типа исходного массива (экземпляр Array или Vector) функция пытается преобразовать элементы исходного массива в значения базового типа. В ходе этого процесса используются стандартные правила преобразования типа данных ActionScript. Например, следующий код преобразует значения String исходного экземпляра Array в целые числа для полученного массива Vector. Десятичная часть первого значения (`"1.5"`) усекается, а нечисловое третье значение (`"Waffles"`) преобразуется в 0 в результате:

```
var numbers:Vector.<int> = Vector.<int>(["1.5", "17", "Waffles"]);
trace(numbers); // output: 1,17,0
```

Если не удается преобразовать ни один из элементов, выдается ошибка.

Если при вызове глобальной функции `Vector.<T>()` элемент в исходном массиве является экземпляром подкласса указанного базового типа, этот элемент добавляется в полученный объект Vector (ошибка не выдается). Использование глобальной функции `Vector.<T>()` фактически является единственным способом преобразования объекта Vector с базовым типом `T` в Vector с базовым типом, который является суперклассом `T`.

## Вставка элементов массива

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Самым простым способом добавления элемента в индексный массив является использование оператора доступа к массиву (`[]`). Чтобы задать значение для элемента индексного массива, используйте имя объекта Array или Vector и индекс в левой части инструкции присвоения.

```
songTitles[5] = "Happy Birthday";
```

Если в объекте Array или Vector еще нет элемента с таким индексом, он создается и используется для сохранения нового значения. Если в данной позиции индекса уже существует значение, оно заменяется новым значением.

Объект Array позволяет создавать элементы в любой позиции индекса. А в объекте Vector, напротив, значение можно присваивать только существующему индекса или следующему за ним. Следующий доступный индекс соответствует свойству `length` объекта Vector. Самым безопасным способом добавления нового элемента в объект Vector является использование кода, подобного следующему:

```
myVector[myVector.length] = valueToAdd;
```

Три метода классов Array и Vector: `push()`, `unshift()` и `splice()` — позволяют вставлять элементы в индексный массив. Метод `push()` добавляет один или несколько элементов в конец массива. Иными словами, у последнего элемента, вставленного в массив с помощью метода `push()`, будет наибольший индекс. Метод `unshift()` вставляет один или более элементов в начало массива, при этом первый индекс всегда равен 0. Метод `splice()` вставляет любое число элементов по заданному индексу в массиве.

В следующем примере показаны все три метода. Массив с именем `planets` создается для хранения названий планет в порядке удаленности от Солнца. Сначала вызывается метод `push()` и создается первый элемент `Mars`. Затем вызывается метод `unshift()` и вставляет в начало массива элемент `Mercury`. Наконец, вызывается метод `splice()` и вставляет элементы `Venus` и `Earth` после `Mercury` и перед `Mars`. Первый аргумент, передаваемый методу `splice()`, — это целое число 1. Он указывает, что вставку необходимо производить после индекса 1. Второй аргумент, передаваемый методу `splice()`, — это целое число 0. Он указывает, что никакие элементы удалять не нужно. Наконец, третий и четвертый аргументы — `Venus` и `Earth`, передаваемые методу `splice()`, — это вставляемые элементы.

```
var planets:Array = new Array();
planets.push("Mars"); // array contents: Mars
planets.unshift("Mercury"); // array contents: Mercury,Mars
planets.splice(1, 0, "Venus", "Earth");
trace(planets); // array contents: Mercury,Venus,Earth,Mars
```

Методы `push()` и `unshift()` возвращают целое число без знака, обозначающее величину измененного массива. Когда для вставки элементов используется метод `splice()`, он возвращает пустой массив. Это может показаться странным, но с учетом богатых возможностей применений метода `splice()` это вполне оправданно. Метод `splice()` можно использовать не только для вставки элементов в массив, но и для их удаления из массива. Когда метод `splice()` используется для удаления элементов, он возвращает массив удаленных элементов.

***Примечание.*** *Если свойство `fixed` объекта Vector имеет значение `true`, общее число элементов в массиве нельзя изменить. Если попытаться добавить новый элемент в объект Vector с фиксированной длиной с помощью описанных здесь приемов, произойдет ошибка.*

## Получение значений и удаление элементов массива

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Самый простой способ получить значение элемента индексного массива — использовать оператор доступа к массиву (`[]`). Чтобы получить значение для элемента индексного массива, используйте имя объекта Array или Vector и индекс в правой части инструкции присвоения.

```
var myFavoriteSong:String = songTitles[3];
```

Также можно попытаться получить значение из объекта Array или Vector, используя позицию индекса, в которой нет элемента. В этом случае объект Array возвращает значение undefined, а объект Vector выдает исключение RangeError.

Три метода классов Array и Vector: `pop()`, `shift()` и `splice()` — позволяют удалять элементы. Метод `pop()` удаляет элемент из конца массива. Иначе говоря, он удаляет элемент с наибольшим индексом. Метод `shift()` удаляет элемент из начала массива, т. е. каждый раз удаляется элемент с индексом 0. Метод `splice()`, который также может использоваться для вставки элементов, удаляет произвольное число элементов, начиная с индекса, заданного первым из передаваемых методу аргументов.

В примере ниже показано использование всех трех методов для удаления элементов из экземпляра Array. Массив Array с именем `oceans` создается для хранения названий океанов. Некоторые из названий в массиве — это озера, а не океаны, и их нужно удалить.

Во-первых, используем метод `splice()` для удаления элементов `Aral` и `Superior` и вставки элементов `Atlantic` и `Indian`. Первый аргумент, передаваемый методу `splice()`, — это целое число 2. Он указывает, что операцию необходимо начинать с третьего элемента в списке, т. е. элемента с индексом 2. Второй аргумент, 2, указывает, что нужно удалить два элемента. Остальные аргументы: `Atlantic` и `Indian` — это значения, которые нужно вставить после индекса 2.

Во-вторых, используем метод `pop()` для удаления последнего элемента массива — `Huron`. В-третьих, используем метод `shift()` для удаления первого элемента массива — `Victoria`.

```
var oceans:Array = ["Victoria", "Pacific", "Aral", "Superior", "Indian", "Huron"];
oceans.splice(2, 2, "Arctic", "Atlantic"); // replaces Aral and Superior
oceans.pop(); // removes Huron
oceans.shift(); // removes Victoria
trace(oceans);// output: Pacific,Arctic,Atlantic,Indian
```

Методы `pop()` и `shift()` возвращают элемент, который был удален. Для объекта Array, типом данных возвращаемого значения является Object, так как массивы могут содержать данные любого типа. Для объекта Vector типом данных возвращаемого значения является его базовый класс. Метод `splice()` возвращает объект Array или Vector с удаленными значениями. Можно изменить пример с массивом `oceans` так, чтобы вызов метода `splice()` назначал возвращенный объект Array новой переменной Array, как показано ниже:

```
var lakes:Array = oceans.splice(2, 2, "Arctic", "Atlantic");
trace(lakes); // output: Aral,Superior
```

Вам может встретиться код, применяющий оператор `delete` к элементу объекта Array. Оператор `delete` задает элементу объекта Array значение `undefined`, но не удаляет его из массива. Например, в приведенном ниже коде оператор `delete` применяется к третьему элементу массива `oceans`, но длина объекта Array по-прежнему остается равной 5.

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Indian", "Atlantic"];
delete oceans[2];
trace(oceans);// output: Arctic,Pacific,,Indian,Atlantic
trace(oceans[2]); // output: undefined
trace(oceans.length); // output: 5
```

Можно усечь экземпляр Array или Vector с помощью свойства `length`. Если задать свойству `length` индексного массива значение меньше текущей длины, массив будет усечен, и все элементы с индексами больше нового значения `length` минус 1 будут удалены. Например, если массив `oceans` отсортировать так, чтобы все нужные значения были в начале массива, то с помощью свойства `length` можно удалить записи в конце массива, как показано ниже:

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Aral", "Superior"];
oceans.length = 2;
trace(oceans); // output: Arctic,Pacific
```

***Примечание.*** *Если свойство `fixed` объекта Vector имеет значение `true`, общее число элементов в массиве нельзя изменить. Если попытаться удалить элемент или усечь экземпляр Vector с фиксированной длиной с помощью описанных здесь приемов, произойдет ошибка.*

## Сортировка массива

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Существует три метода: `reverse()`, `sort()` и `sortOn()`, — позволяющих изменить порядок элементов в индексном массиве путем сортировки или задания обратного порядка. Все они изменяют имеющийся массив. В следующей таблице приводятся общие сведения об этих методах и их поведении для объектов Array и Vector.

| Метод | Поведение Array | Поведение Vector |
|---|---|---|
| `reverse()` | Изменяет порядок элементов таким образом, что последний элемент становится первым, предпоследний — вторым и так далее. | То же, что для объекта Array |
| `sort()` | Позволяет сортировать элементы объекта Array разными предварительно определенными способами, например в алфавитном порядке или по номерам. Также можно задать пользовательский алгоритм сортировки | Сортирует элементы в соответствии с алгоритмом сортировки, определенным пользователем. |
| `sortOn()` | Позволяет сортировать объекты, которые имеют одно или несколько общих свойств, указав одно или более свойств в качестве критериев сортировки. | Не доступен для класса Vector |

### Метод reverse()

Метод `reverse()` не принимает никаких параметров и не возвращает значений, но позволяет переключать порядок массива с прямого на обратный. В следующем примере порядок океанов в массиве `oceans` меняется с прямого на обратный:

```
var oceans:Array = ["Arctic", "Atlantic", "Indian", "Pacific"];
oceans.reverse();
trace(oceans); // output: Pacific,Indian,Atlantic,Arctic
```

### Базовая сортировка с помощью метода sort() (только для класса Array)

Для класса Array метод `sort()` меняет порядок элементов в массиве с помощью *порядка сортировки по умолчанию*. Порядок сортировки по умолчанию обладает определенными характеристиками. Они перечислены ниже.

- Сортировка чувствительна к регистру, поэтому заглавные буквы всегда имеют приоритет перед строчными. Например, буква «D» предшествует букве «b».
- Сортировка выполняется по возрастанию, т.е. символы с меньшей величиной кода (скажем, «A») предшествуют символам с большей (скажем, «B»).
- Одинаковые значения располагаются друг за другом, но определенного порядка внутри них самих нет.
- В основе сортировки лежат строки, так как элементы перед сравнением преобразуются в строки (например, 10 предшествует 3, потому что строка `"1"` имеет меньшую величину кода символа, чем `"3"`).

Вам может потребоваться отсортировать массив без учета регистра или в порядке убывания, а может быть, массив содержит числа, которые вы хотите отсортировать по величине, а не в алфавитном порядке. У метода `sort()` класса Array есть параметр `options`, позволяющий изменять каждую из характеристик порядка сортировки по умолчанию. Характеристики определяются набором статических переменных в классе Array, как показано ниже.

- `Array.CASEINSENSITIVE` отменяет чувствительность к регистру. В этом случае, например, строчная буква «b» будет предшествовать заглавной «D».

- `Array.DESCENDING` изменяет возрастающий порядок сортировки по умолчанию на убывающий. Например, буква «B» предшествует букве «A».
- `Array.UNIQUESORT` останавливает сортировку, если обнаруживаются два одинаковых значения.
- `Array.NUMERIC` позволяет выполнять сортировку чисел по величине так, чтобы число 3 стояло перед числом 10.

В примере ниже некоторые из этих параметров освещены подробнее. Создается объект Array с именем `poets`, который сортируется с помощью нескольких разных параметров.

```
var poets:Array = ["Blake", "cummings", "Angelou", "Dante"];
poets.sort(); // default sort
trace(poets); // output: Angelou,Blake,Dante,cummings

poets.sort(Array.CASEINSENSITIVE);
trace(poets); // output: Angelou,Blake,cummings,Dante

poets.sort(Array.DESCENDING);
trace(poets); // output: cummings,Dante,Blake,Angelou

poets.sort(Array.DESCENDING | Array.CASEINSENSITIVE); // use two options
trace(poets); // output: Dante,cummings,Blake,Angelou
```

**Пользовательская сортировка с помощью метода sort() (для классов Array и Vector)**

Помимо базовой сортировки, доступной для объектов Array, также можно определить пользовательское правило сортировки. Этот прием является единственной формой использования метода `sort()` для класса Vector. Чтобы определить собственный порядок сортировки, нужно написать пользовательскую функцию сортировки и передать ее в качестве аргумента методу `sort()`.

Например, если у вас есть список имен, каждый из элементов которого состоит из имени и фамилии, а вы хотите отсортировать его по фамилиям, используйте собственную функцию сортировки для разбора каждого элемента и выполнения сортировки по фамилии. В примере ниже показано, как это сделать с помощью собственной функции, которая является параметром метода `Array.sort()`:

```
var names:Array = new Array("John Q. Smith", "Jane Doe", "Mike Jones");
function orderLastName(a, b):int
{
    var lastName:RegExp = /\b\S+$/;
    var name1 = a.match(lastName);
    var name2 = b.match(lastName);
    if (name1 < name2)
    {
        return -1;
    }
    else if (name1 > name2)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
trace(names); // output: John Q. Smith,Jane Doe,Mike Jones
names.sort(orderLastName);
trace(names); // output: Jane Doe,Mike Jones,John Q. Smith
```

Персонализированная функция сортировки `orderLastName()` использует регулярное выражение для извлечения фамилии из каждого элемента и сравнения значений по фамилиям. Идентификатор функции `orderLastName` используется как самостоятельный параметр при вызове метода `sort()` в массиве `names`. Функция сортировки принимает два параметра, `a` и `b`, потому что она обрабатывает два элемента одновременно. Значение, возвращаемое функцией сортировки, указывает, каким образом сортировать элементы:

- Возвращенное значение -1 означает, что первый параметр (`a`) стоит перед вторым (`b`).
- Возвращенное значение 1 указывает, что второй параметр (`b`) стоит перед первым (`a`).
- Возвращенное значение 0 говорит о том, что элементы равны в контексте сортировки.

### Метод sortOn() (только для класса Array)

Метод `sortOn()` разработан для объектов Array с элементами, содержащими объекты. У этих объектов должно быть хотя бы одно общее свойство, которое можно использовать в качестве ключа сортировки. Использование метода `sortOn()` в массивах любого другого типа возвращает непредсказуемые результаты.

***Примечание.*** *Класс Vector не имеет метода `sortOn()`. Он доступен только для объектов Array.*

В примере ниже объект Array с именем `poets` преобразуется, и каждый элемент превращается из строки в объект. Каждый объект содержит фамилию поэта и год его рождения.

```
var poets:Array = new Array();
poets.push({name:"Angelou", born:"1928"});
poets.push({name:"Blake", born:"1757"});
poets.push({name:"cummings", born:"1894"});
poets.push({name:"Dante", born:"1265"});
poets.push({name:"Wang", born:"701"});
```

Метод `sortOn()` можно использовать для сортировки объекта Array по свойству `born`. Метод `sortOn()` определяет два параметра: `fieldName` и `options`. Аргумент `fieldName` должен быть строкой. В примере ниже вызывается метод `sortOn()` с двумя аргументами — `"born"` и `Array.NUMERIC`. Аргумент `Array.NUMERIC` используется для гарантии того, что сортировка выполняется по величине чисел, а не по алфавиту. Следует придерживаться этого способа, даже если все числа одного порядка, потому что, если добавятся числа меньшего или большего порядков, вы будете точно знать, что сортировка будет выполняться правильно.

```
poets.sortOn("born", Array.NUMERIC);
for (var i:int = 0; i < poets.length; ++i)
{
    trace(poets[i].name, poets[i].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

**Сортировка без изменения исходного массива (только для класса Array)**

Как правило, методы `sort()` и `sortOn()` вносят изменения в объект Array. Если требуется выполнить сортировку без изменения существующего объекта Array, передайте константу `Array.RETURNINDEXEDARRAY` как часть параметра `options`. Этот параметр указывает методам, что нужно возвратить новый объект Array с сортировкой и оставить существующий массив без изменений. Методы возвращают простой массив индексов, отражающий новый порядок сортировки и не содержащий элементов исходного объекта Array. Например, для сортировки массива `poets` по году рождения без изменения самого объекта Array константа `Array.RETURNINDEXEDARRAY` включается как часть аргумента, передаваемого для параметра `options`.

В примере ниже возвращенная информация об индексах сохраняется в объекте Array с именем `indices`, а затем массив `indices` используется вместе с исходным массивом `poets`, чтобы вывести имена поэтов в порядке их годов рождения.

```
var indices:Array;
indices = poets.sortOn("born", Array.NUMERIC | Array.RETURNINDEXEDARRAY);
for (var i:int = 0; i < indices.length; ++i)
{
    var index:int = indices[i];
    trace(poets[index].name, poets[index].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

## Запросы в массив

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Четыре метода классов Array и Vector: `concat()`, `join()`, `slice()`, `toString()` — отправляют запросы в массив, но не изменяют его. Методы `concat()` и `slice()` возвращают новые массивы, а методы `join()` и `toString()` возвращают строки. Метод `concat()` принимает в качестве аргументов новый массив или список элементов и совмещает его с предыдущим массивом для создания нового. У метода `slice()` есть два параметра с говорящими названиями `startIndex` и `endIndex`. Этот метод возвращает новый массив, содержащий копии элементов, «вырезанных» из существующего массива. Вырезаемый фрагмент начинается с элемента с индексом `startIndex` и заканчивается перед элементом `endIndex`. Еще раз отметим, что элемент с индексом `endIndex` не включается в возвращаемые значения.

В следующем примере методы `concat()` и `slice()` используются для создания новых массивов с помощью элементов других массивов:

```
var array1:Array = ["alpha", "beta"];
var array2:Array = array1.concat("gamma", "delta");
trace(array2); // output: alpha,beta,gamma,delta

var array3:Array = array1.concat(array2);
trace(array3); // output: alpha,beta,alpha,beta,gamma,delta

var array4:Array = array3.slice(2,5);
trace(array4); // output: alpha,beta,gamma
```

Методы `join()` и `toString()` можно использовать для отправки запросов в массив и возвращения его содержимого в виде строки. Если в методе `join()` не используются никакие параметры, оба метода ведут себя одинаково и возвращают строку, содержащую все элементы массива, разделенные запятыми. Метод `join()`, в отличие от метода `toString()`, принимает параметр `delimiter`, позволяющий выбрать знак для отделения элементов возвращаемой строки друг от друга.

В примере ниже создается объект Array с именем `rivers`, затем вызываются методы `join()` и `toString()` для возвращения значений массива в виде строки. Метод `toString()` возвращает значения, разделенные запятой (`riverCSV`), в то время как метод `join()` возвращает значения, разделенные знаком `+`.

```
var rivers:Array = ["Nile", "Amazon", "Yangtze", "Mississippi"];
var riverCSV:String = rivers.toString();
trace(riverCSV); // output: Nile,Amazon,Yangtze,Mississippi
var riverPSV:String = rivers.join("+");
trace(riverPSV); // output: Nile+Amazon+Yangtze+Mississippi
```

При работе с методом `join()` необходимо помнить, что он всегда возвращает вложенные экземпляры Array или Vector со значениями, разделенными запятой, вне зависимости от того, какой разделительный знак вы задали для основных элементов массива. Это проиллюстрировано в примере:

```
var nested:Array = ["b","c","d"];
var letters:Array = ["a",nested,"e"];
var joined:String = letters.join("+");
trace(joined); // output: a+b,c,d+e
```

# Ассоциативные массивы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Ассоциативный массив, иногда называемый *хеш-таблицей* или *сопоставлением*, использует для организации хранимых значений *ключи* вместо числовых индексов. Каждый ключ в ассоциативном массиве — это уникальная строка, используемая для доступа к хранимому значению. Ассоциативный массив — это экземпляр класса Object, то есть каждый ключ соответствует имени свойства. Ассоциативные массивы — это неупорядоченные наборы пар «ключ-значение». Ключи в ассоциативном массиве не подчиняются какому-либо порядку.

В ActionScript 3.0 также предусмотрен расширенный тип ассоциативного массива под названием *словарь*. Словари являются экземплярами класса Dictionary в пакете flash.utils и используют ключи, которые могут быть любым типом данных. Иными словами, ключи словаря не ограничиваются значениями типа String.

## Ассоциативные массивы со строками-ключами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В ActionScript 3.0 ассоциативные массивы можно создавать двумя способами. Во-первых, можно использовать экземпляр Object. С помощью экземпляра Object можно инициализировать массив, используя литерал объекта. Экземпляр класса Object, также называемый *родовым объектом*, по своим функциям идентичен ассоциативному массиву. Имя каждого из свойств родового объекта служит ключом доступа к сохраненному значению.

В примере ниже показано, как создавать ассоциативный массив `monitorInfo` с помощью литерала объекта, чтобы инициализировать массив с двумя парами «ключ-значение».

```
var monitorInfo:Object = {type:"Flat Panel", resolution:"1600 x 1200"};
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

Если вам не нужно инициализировать массив во время объявления, можно использовать для создания массива конструктор Object, как показано ниже:

```
var monitorInfo:Object = new Object();
```

После создания массива с помощью литерала объекта или конструктора класса Object можно добавить к массиву новые значения либо с помощью оператора доступа к массиву ( `[]` ), либо с помощью оператора «точка» ( `.` ). В примере ниже показано, как добавить два новых значения в массив `monitorArray`:

```
monitorInfo["aspect ratio"] = "16:10"; // bad form, do not use spaces
monitorInfo.colors = "16.7 million";
trace(monitorInfo["aspect ratio"], monitorInfo.colors);
// output: 16:10 16.7 million
```

Обратите внимание, что ключ `aspect ratio` содержит символ пробела. При использовании с оператором доступа к массиву ( `[]` ) это допустимо, но при использовании с оператором «точка» приведет к ошибке. Использовать пробелы в именах ключей не рекомендуется.

Второй способ создать ассоциативный массив — использовать конструктор Array (или конструктор любого динамического класса), а затем использовать оператор доступа к массиву ( `[]` ) или «точка» ( `.` ) для добавления пар «ключ-значение» в массив. Если вы объявляете ассоциативный массив как массив любого типа класса Array, нельзя использовать для инициализации массива литерал объекта. В примере ниже показано, как создать ассоциативный массив `monitorInfo` с помощью конструктора Array и добавить ключ `type` и ключ `resolution` вместе со значениями:

```
var monitorInfo:Array = new Array();
monitorInfo["type"] = "Flat Panel";
monitorInfo["resolution"] = "1600 x 1200";
trace(monitorInfo["type"], monitorInfo["resolution"]);
// output: Flat Panel 1600 x 1200
```

Использование конструктора Array для создания ассоциативного массива не дает никаких особых преимуществ. С ассоциативными массивами нельзя использовать свойство `Array.length` или любой из методов класса Array, даже если вы используете конструктор Array или тип данных Array. Конструктор Array лучше всего подходит для создания индексных массивов.

## Ассоциативные массивы с объектами-ключами (Словари)

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Можно использовать класс Dictionary для создания ассоциативного массива, использующего в качестве ключей объекты, а не строки. Такие массивы иногда называют словарями, хеш-таблицами или преобразованиями. Например, рассмотрим приложение, определяющее расположение объекта Sprite на основании его связи с определенным контейнером. Можно использовать объект Dictionary, чтобы сопоставить каждый объект Sprite с контейнером.

В примере ниже создаются три экземпляра класса Sprite, которые служат ключами для объекта Dictionary. Каждому ключу задается значение `GroupA` или `GroupB`. Значения могут быть любого типа данных, но в этом примере и `GroupA`, и `GroupB` являются экземплярами класса Object. Следовательно, доступ к значениям, связанным с каждым из ключей с оператором доступа к массиву ( `[]` ), осуществляется способом, показанным ниже:

```
import flash.display.Sprite;
import flash.utils.Dictionary;

var groupMap:Dictionary = new Dictionary();

// objects to use as keys
var spr1:Sprite = new Sprite();
var spr2:Sprite = new Sprite();
var spr3:Sprite = new Sprite();

// objects to use as values
var groupA:Object = new Object();
var groupB:Object = new Object();

// Create new key-value pairs in dictionary.
groupMap[spr1] = groupA;
groupMap[spr2] = groupB;
groupMap[spr3] = groupB;

if (groupMap[spr1] == groupA)
{
    trace("spr1 is in groupA");
}
if (groupMap[spr2] == groupB)
{
    trace("spr2 is in groupB");
}
if (groupMap[spr3] == groupB)
{
    trace("spr3 is in groupB");
}
```

**Итерации с объектами-ключами**

Можно выполнять итерацию в объекте Dictionary с помощью цикла `for..in` или `for each..in`. Цикл `for..in` позволяет выполнять итерацию на основе ключей, а цикл `for each..in` — на основе значений, связанных с ключами.

Используйте цикл `for..in` для прямого доступа к объектам-ключам объекта Dictionary. Доступ к значениям объекта Dictionary возможен и без оператора доступа к массиву ( `[]` ). В приведенном ниже коде словарь `groupMap` из предыдущего примера используется для выполнения итерации в объекте Dictionary с помощью цикла `for..in`:

```
for (var key:Object in groupMap)
{
    trace(key, groupMap[key]);
}
/* output:
[object Sprite] [object Object]
[object Sprite] [object Object]
[object Sprite] [object Object]
*/
```

Используйте цикл `for each..in` для прямого доступа к значениям объекта Dictionary. В приведенном ниже коде словарь `groupMap` используется для выполнения итерации в объекте Dictionary с помощью цикла `for each..in`:

```
for each (var item:Object in groupMap)
{
    trace(item);
}
/* output:
[object Object]
[object Object]
[object Object]
*/
```

**Объекты-ключи и управление памятью**

Проигрыватель Adobe® Flash® Player и среда Adobe® AIR™ используют систему сборки мусора для восстановления памяти, которая больше не используется. Когда на объект не ведет ни одна ссылка, он становится кандидатом на сборку в мусор, поэтому при следующей сборке мусора занимаемая им память высвобождается. Например, в приведенном ниже коде создается новый объект, а переменной `myObject` назначается ссылка на объект:

```
var myObject:Object = new Object();
```

Покуда на объект указывает хотя бы одна ссылка, сборщик мусора не будет высвобождать занимаемую им память. Если значение `myObject` меняется и теперь указывает на другой объект или становится `null`, то память, занимаемая исходным объектом, подлежит сборке в мусор, но лишь в том случае, если на исходный объект не ведут другие ссылки.

Если `myObject` используется как ключ в объекте Dictionary, создается еще одна ссылка на исходный объект. Например, в приведенном ниже коде создаются две ссылки на объект — переменную `myObject` и ключ в объекте `myMap`:

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary();
myMap[myObject] = "foo";
```

Чтобы объект, на который ссылается `myObject`, мог быть собран в мусор, необходимо удалить все ссылки на него. В этом случае нужно изменить значение `myObject` и удалить ключ `myObject` из `myMap`, как показано ниже:

```
myObject = null;
delete myMap[myObject];
```

Вы также можете использовать параметр `useWeakReference` конструктора Dictionary, чтобы превратить все ключи словаря в *слабые ссылки*. Сборщик мусора игнорирует слабые ссылки, т. е. объект, на который ведут только слабые ссылки, подлежит сборке в мусор. Например, в приведенном ниже коде не нужно удалять ключ `myObject` из `myMap`, чтобы объект можно было собрать в мусор:

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary(true);
myMap[myObject] = "foo";
myObject = null; // Make object eligible for garbage collection.
```

# Многомерные массивы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Элементами многомерных массивов являются другие массивы. Например, рассмотрим план на неделю, сохраненный в виде индексного массива строк:

```
var tasks:Array = ["wash dishes", "take out trash"];
```

Если нужно сохранить отдельный список дел для каждого дня недели, вы можете создать многомерный массив, где каждый день недели будет элементом. Каждый элемент содержит индексный массив, похожий на массив `tasks`, в котором хранится список задач. В многомерных массивах можно использовать любую комбинацию ассоциативных массивов. В примерах из следующих разделов используются два индексных массива или ассоциативный массив индексных массивов. Вы можете попробовать другие сочетания.

## Два индексных массива

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При использовании двух индексных массивов можно визуально представить результат как таблицу. Элементы первого массива будут строками таблицы, а элементы второго — столбцами.

Например, в следующем многомерном массиве два индексных массива используются для отслеживания списков дел для каждого дня недели. Первый массив, `masterTaskList`, создается с помощью конструктора класса Array. Каждый элемент массива представляет один из дней недели, где 0 — это понедельник, а 6 — воскресенье. Эти элементы будут рядами таблицы. Задача для каждого дня создается путем назначения литерала массива каждому из семи элементов, созданных в массиве `masterTaskList`. Литералы массива представляют столбцы таблицы.

```
var masterTaskList:Array = new Array();
masterTaskList[0] = ["wash dishes", "take out trash"];
masterTaskList[1] = ["wash dishes", "pay bills"];
masterTaskList[2] = ["wash dishes", "dentist", "wash dog"];
masterTaskList[3] = ["wash dishes"];
masterTaskList[4] = ["wash dishes", "clean house"];
masterTaskList[5] = ["wash dishes", "wash car", "pay rent"];
masterTaskList[6] = ["mow lawn", "fix chair"];
```

Обращаться к отдельным элементам в любых списках задач можно с помощью оператора доступа к массиву (`[]`). Первая пара квадратных скобок представляет день недели, а вторая — список дел для этого дня. Например, для получения второй задачи из списка дел на среду нужно сначала использовать индекс 2 для среды, а потом индекс 1 для второй задачи в списке.

```
trace(masterTaskList[2][1]); // output: dentist
```

Чтобы получить первую задачу из списка дел на воскресенье, используйте индекс 6 для вызова воскресенья, а затем индекс 0 для первой задачи.

```
trace(masterTaskList[6][0]); // output: mow lawn
```

## Ассоциативный массив с индексным массивом

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы упростить доступ к отдельным массивам, используйте ассоциативный массив для дней недели и индексный массив для списков задач. Использование ассоциативного массива позволяет использовать синтаксис записи через точку при обращении к определенному дню недели, но обращение к каждому элементу ассоциативного массива потребует дополнительного времени обработки. В примере ниже показано, как использовать для списка задач ассоциативный массив с парами «ключ-значение» для каждого из дней недели:

```
var masterTaskList:Object = new Object();
masterTaskList["Monday"] = ["wash dishes", "take out trash"];
masterTaskList["Tuesday"] = ["wash dishes", "pay bills"];
masterTaskList["Wednesday"] = ["wash dishes", "dentist", "wash dog"];
masterTaskList["Thursday"] = ["wash dishes"];
masterTaskList["Friday"] = ["wash dishes", "clean house"];
masterTaskList["Saturday"] = ["wash dishes", "wash car", "pay rent"];
masterTaskList["Sunday"] = ["mow lawn", "fix chair"];
```

Синтаксис записи через точку повышает удобочитаемость кода, избавляя его от многочисленных квадратных скобок.

```
trace(masterTaskList.Wednesday[1]); // output: dentist
trace(masterTaskList.Sunday[0]);// output: mow lawn
```

Можно выполнять итерацию в списке задач с помощью цикла `for..in`, но тогда вместо синтаксиса записи через точку нужно использовать оператор доступа к массиву (`[]`) для получения значений, связанных с ключами. Так как `masterTaskList` является ассоциативным массивом, его элементы не обязательно будут вызываться по порядку, что демонстрируется в примере ниже:

```
for (var day:String in masterTaskList)
{
    trace(day + ": " + masterTaskList[day])
}
/* output:
Sunday: mow lawn,fix chair
Wednesday: wash dishes,dentist,wash dog
Friday: wash dishes,clean house
Thursday: wash dishes
Monday: wash dishes,take out trash
Saturday: wash dishes,wash car,pay rent
Tuesday: wash dishes,pay bills
*/
```

# Клонирование массивов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

У класса Array нет встроенного метода для создания копий массивов. Можно создать *поверхностнуюкопию* массива путем вызова метода `concat()` или `slice()` без аргументов. Если элементами исходного массива являются объекты, то поверхностная копия будет содержать только ссылки на объекты, а не сами объекты. Копия указывает на те же объекты, что и оригинал. Любые изменения объектов отражаются в обоих массивах.

В *глубокой копии* все объекты, обнаруженные в исходном массиве, копируются, так что новый массив не указывает на те же объекты, что исходный. При глубоком копировании требуется более одной строки кода, для чего обычно нужно создать функцию. Можно создать глобальную вспомогательную функцию или метод подкласса Array.

В примере ниже определяется функция `clone()`, которая выполняет глубокое копирование. Алгоритм заимствован у широко используемой техники языка Java. Функция создает глубокую копию, упорядочивая массив и превращая его в экземпляр класса ByteArray, а затем заново считывая его в новый массив. Эта функция принимает объект, чтобы его можно было использовать как с индексными массивами, так и с ассоциативными, как показано в коде ниже:

```
import flash.utils.ByteArray;

function clone(source:Object):*
{
    var myBA:ByteArray = new ByteArray();
    myBA.writeObject(source);
    myBA.position = 0;
    return(myBA.readObject());
}
```

# Расширение класса Array

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс Array — это один из нескольких ключевых нефинальных классов, то есть он позволяет создавать собственные подклассы класса Array. В этом разделе приводится пример того, как создавать подкласс класса Array и рассказывается о некоторых проблемах, которые могут возникнуть при выполнении этой операции.

Как было сказано выше, массивы в языке ActionScript не типизированные, но можно создать подкласс класса Array, который будет принимать элементы только определенного типа данных. В примере в последующих разделах определяется подкласс класса Array с именем TypedArray, в первом параметре которого накладывается ограничение на тип данных элементов. Класс TypedArray представлен исключительно в качестве примера того, как расширить класс Array, и может не подойти для практического применения по ряду причин. Во-первых, проверка типа производится во время выполнения, а не при компиляции. Во-вторых, когда метод TypedArray сталкивается с несовпадением, он игнорирует его, а исключение при этом не генерируется, хотя можно легко модифицировать методы, чтобы они генерировали исключения. В-третьих, класс не может предотвратить использование оператора доступа к массиву для вставки значений какого-либо типа в массив. В-четвертых, в силу стиля кода предпочтение отдается простоте, а не оптимизации.

***Примечание.*** *Описанный здесь прием можно использовать для создания типизированного массива. Однако для этого лучше использовать объект Vector. Экземпляр Vector является настоящим типизированным массивом и превосходит класс Array и его подклассы по производительности и другим показателям. В целях данного обсуждения демонстрируется создание подкласса Array.*

### Объявление подкласса

Используйте ключевое слово `extends` для обозначения того, что класс является подклассом класса Array. Подкласс класса должен использовать атрибут `dynamic`, как и класс Array. В противном случае подкласс будет работать некорректно.

В приведенном ниже коде дается определение класса TypedArray, содержащего константу для хранения типа данных, метод конструктора и четыре метода для добавления элементов в массив. В этом примере не приводится код для каждого из методов, но он подробным образом освещается в последующих разделах:

```
public dynamic class TypedArray extends Array
{
    private const dataType:Class;

    public function TypedArray(...args) {}

    AS3 override function concat(...args):Array {}

    AS3 override function push(...args):uint {}

    AS3 override function splice(...args) {}

    AS3 override function unshift(...args):uint {}
}
```

Четыре переопределенных метода используют пространство имен AS3 вместо атрибута `public`, потому что в данном примере мы исходим из того, что параметр компилятора `-as3 = true`, а параметр компилятора `-es = false`. Это настройки по умолчанию для Adobe Flash Builder и AdobeFlashProfessional.

*Если вы опытный разработчик и предпочитаете использовать прототипное наследование, то можете внести две небольших правки в класс TypedArray, чтобы компиляция выполнялась с помощью параметра компиляции `-es` со значением `true`. Во-первых, удалите все вхождения атрибута `override` и замените пространство имен AS3 атрибутом `public`. Во-вторых, замените все четыре вхождения `super` на `Array.prototypesuper`.*

### Конструктор TypedArray

Конструктор подкласса ставит перед разработчиком интересную задачу, так как конструктор должен принимать список аргументов любой длины. Сложность состоит в том, как передавать аргументы суперконструктору для создания массива. Если список аргументов передается как массив, суперконструктор расценивает его как единственный аргумент типа Array, и итоговый массив всегда состоит из 1 элемента. Традиционно список аргументов передается с помощью метода `Function.apply()`, который принимает массив аргументов в качестве второго параметра, но преобразует его в список аргументов при выполнении функции. К сожалению, метод `Function.apply()` нельзя использовать с конструкторами.

Единственный оставшийся вариант — воссоздать логику конструктора Array в конструкторе TypedArray. В приведенном ниже коде показан алгоритм, используемый в конструкторе класса Array, который можно использовать и в конструкторе подкласса Array:

```
public dynamic class Array
{
    public function Array(...args)
    {
        var n:uint = args.length
        if (n == 1 && (args[0] is Number))
        {
            var dlen:Number = args[0];
            var ulen:uint = dlen;
            if (ulen != dlen)
            {
                throw new RangeError("Array index is not a 32-bit unsigned integer ("+dlen+")");
            }
            length = ulen;
        }
        else
        {
            length = n;
            for (var i:int=0; i < n; i++)
            {
                this[i] = args[i]
            }
        }
    }
}
```

Большая часть кода конструктора TypedArray идентична коду конструктора класса Array, за исключением четырех отличий. Во-первых, список параметров включает новый обязательный параметр типа Class, позволяющий задавать тип данных массива. Во-вторых, тип данных, передаваемых конструктору, назначается переменной `dataType`. В-третьих, инструкция `else` — значение свойства `length` — назначается после цикла `for`, поэтому `length` включает только аргументы правильного типа. В-четвертых, внутри цикла `for` используется переопределенный вариант метода `push()`, поэтому к массиву добавляются только аргументы правильного типа данных. В примере ниже показана функция конструктора TypedArray:

```
public dynamic class TypedArray extends Array
{
    private var dataType:Class;
    public function TypedArray(typeParam:Class, ...args)
    {
        dataType = typeParam;
        var n:uint = args.length
        if (n == 1 && (args[0] is Number))
        {
            var dlen:Number = args[0];
            var ulen:uint = dlen
            if (ulen != dlen)
            {
                throw new RangeError("Array index is not a 32-bit unsigned integer ("+dlen+")")
            }
            length = ulen;
        }
        else
        {
            for (var i:int=0; i < n; i++)
            {
                // type check done in push()
                this.push(args[i])
            }
            length = this.length;
        }
    }
}
```

**Переопределенные методы TypedArray**

Класс TypedArray переопределяет четыре метода класса Array, которые позволяют добавлять элементы в массив. В каждом случае переопределенный метод добавляет проверку типа, чтобы гарантировать только добавление элементов правильного типа. Следовательно, каждый метод вызывает вариант самого себя для суперкласса.

Метод `push()` выполняет итерацию в списке аргументов с помощью цикла `for..in` и проверку типа для каждого аргумента. Любой аргумент, не соответствующий правильному типу, удаляется из массива `args` методом `splice()`. После окончания цикла `for..in` в массиве `args` остаются только значения типа `dataType`. Затем вызывается вариант метода `push()` для суперкласса с обновленным массивом `args`, как показано в коде ниже:

```
    AS3 override function push(...args):uint
    {
        for (var i:* in args)
        {
            if (!(args[i] is dataType))
            {
                args.splice(i,1);
            }
        }
        return (super.push.apply(this, args));
    }
```

Метод `concat()` создает временный TypedArray с именем `passArgs` для хранения аргументов, передающих проверку типа. Это позволяет повторно использовать процедуру проверки типа метода `push()`. Цикл `for..in` выполняет итерацию в массиве `args` и вызывает метод `push()` для каждого аргумента. Так как типом `passArgs` является TypedArray, выполняется метод `push()` для TypedArray. Метод `concat()` далее вызывает собственный вариант для суперкласса, как показано ниже:

```
AS3 override function concat(...args):Array
{
    var passArgs:TypedArray = new TypedArray(dataType);
    for (var i:* in args)
    {
        // type check done in push()
        passArgs.push(args[i]);
    }
    return (super.concat.apply(this, passArgs));
}
```

Метод `splice()` принимает произвольный список аргументов, но первые два аргумента всегда обозначают индекс и число удаляемых элементов. Именно поэтому переопределенный метод `splice()` выполняет проверку типа только для элементов массива `args` с индексами 2 и более. Отдельного внимания заслуживает то, что в коде рекурсивно вызывается метод `splice()` внутри цикла `for`, но это не рекурсивный вызов, потому что `args` относится к типу Array, а не TypedArray, следовательно, вызов `args.splice()` является вызовом варианта метода для суперкласса. После окончания цикла `for..in` в массиве `args` остаются значения правильного типа с индексами 2 и выше, а метод `splice()` вызывает свою версию для суперкласса, как показано ниже:

```
AS3 override function splice(...args):*
{
    if (args.length > 2)
    {
        for (var i:int=2; i< args.length; i++)
        {
            if (!(args[i] is dataType))
            {
                args.splice(i,1);
            }
        }
    }
    return (super.splice.apply(this, args));
}
```

Метод `unshift()`, добавляющий элементы в начало массива, также принимает произвольный список аргументов. Переопределенный метод `unshift()` использует алгоритм, очень похожий на метод `push()`, как показано в приведенном ниже коде:

```
AS3 override function unshift(...args):uint
{
    for (var i:* in args)
    {
        if (!(args[i] is dataType))
        {
            args.splice(i,1);
        }
    }
    return (super.unshift.apply(this, args));
}
}
```

# Пример массивов: PlayList

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В примере PlayList показаны техники работы с массивами в контексте приложения, управляющего списками воспроизведения песен. Рассматриваются следующие техники:

- создание индексного массива;
- добавление элементов в индексный массив;
- сортировка массива объектов по различным свойствам с помощью нескольких параметров сортировки;
- преобразование массива в строку значений, разделенных заданным символом.

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения PlayList находятся в каталоге Samples/PlayList. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| PlayList.mxml<br>или<br>PlayList.fla | Основной файл приложения Flash (FLA) или Flex (MXML). |
| com/example/programmingas3/playlist/PlayList.as | Класс, отвечающий за отображение списка композиций. Для хранения списка данный класс использует Array. Он также отвечает за сортировку элементов списка. |
| com/example/programmingas3/playlist/Song.as | Объект значения, представляющий информацию об отдельной песне. Элементы, управляемые классом PlayList, являются экземплярами Song. |
| com/example/programmingas3/playlist/SortProperty.as | Псевдоперечисление, доступные значения которого представляют свойства класса Song, по которым сортируются объекты Song в списке. |

## Обзор класса PlayList

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс PlayList управляет набором объектов Song. У него есть публичные методы с функциональностью, поддерживающей добавление песен в список воспроизведения (метод `addSong()`) и их сортировку в списке (метод `sortList()`). Кроме того, класс включает свойство доступа только для чтения, `songList`, позволяющее подключаться к действительному списку песен в списке воспроизведения. Внутри класса PlayList песни отслеживаются с помощью частной переменной Array:

```
public class PlayList
{
    private var _songs:Array;
    private var _currentSort:SortProperty = null;
    private var _needToSort:Boolean = false;
    ...
}
```

Кроме переменной Array `_songs`, которая используется классом PlayList для отслеживания списка песен, две другие частные переменные отслеживают, нужно ли сортировать список (`_needToSort`), и какое свойство нужно использовать для сортировки в заданное время (`_currentSort`).

Как и в случае всех остальных объектов, объявление экземпляра Array — это лишь половина успеха в деле создания массива. Прежде чем пользоваться свойствами или методами экземпляра Array, необходимо создать экземпляр с помощью конструктора класса PlayList.

```
public function PlayList()
{
    this._songs = new Array();
    // Set the initial sorting.
    this.sortList(SortProperty.TITLE);
}
```

Первая строка конструктора создает экземпляр переменной `_songs`, которая готова к использованию. Кроме того, для установки начального свойства сортировки вызывается метод `sortList()`.

## Добавление песни в список

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Когда пользователь добавляет в приложение новую песню, код в элементе данных вызывает метод класса PlayList `addSong()`.

```
/**
 * Adds a song to the playlist.
 */
public function addSong(song:Song):void
{
    this._songs.push(song);
    this._needToSort = true;
}
```

Внутри метода `addSong()` вызывается метод `push()` массива `_songs`, который добавляет объект Song, переданный методу `addSong()`, в качестве нового элемента массива. Метод `push()` добавляет новый элемент в конец массива независимо от того, выполнялась ли ранее сортировка массива. Это означает, что после вызова метода `push()` список песен вряд ли будет отсортирован правильно, поэтому нужно задать переменной `_needToSort` значение `true`. В теории метод `sortList()` можно вызвать незамедлительно, благодаря чему не нужно будет отслеживать, отсортирован ли список в тот или иной момент времени. Однако на практике список песен нужно сортировать лишь непосредственно перед тем, как к нему обратятся. Отказываясь от сортировки, приложение пропускает ненужные операции, например, в случае, если к списку до обращения было добавлено несколько песен.

## Сортировка списка песен

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Так как экземпляры Song, которыми управляет список воспроизведения, являются сложными объектами, пользователям приложения может потребоваться выполнить сортировку списка песен по разным свойствам, например по названию или по году выпуска. В приложении PlayList задача сортировки списка песен состоит из трех частей: определение свойства, по которому выполняется сортировка; определение параметров сортировки при сортировке по этому свойству; выполнение собственно операции сортировки.

### Свойства сортировки

Объект Song отслеживает некоторые свойства, включая название песни, имя исполнителя, год выпуска, имя файла и жанр песни, который выбирает сам пользователь. Для сортировки подходят только первые три. Для удобства разработчиков в пример включен класс SortProperty, действующий как перечисление со значениями, представляющими свойства для сортировки.

```
public static const TITLE:SortProperty = new SortProperty("title");
public static const ARTIST:SortProperty = new SortProperty("artist");
public static const YEAR:SortProperty = new SortProperty("year");
```

Класс SortProperty содержит три переменные: `TITLE`, `ARTIST` и `YEAR`, в каждой из которых хранится строка с именем связанного свойства класса Song, используемого при сортировке. В остальном коде свойство сортировки определяется с помощью члена перечисления. Например, в конструкторе PlayList список изначально сортируется путем вызова метода `sortList()`, как показано ниже:

```
    // Set the initial sorting.
    this.sortList(SortProperty.TITLE);
```

Свойство сортировки задано в форме `SortProperty.TITLE`, поэтому песни сортируются по названию.

### Сортировка по свойству и установка свойств сортировки

Сама сортировка выполняется классом PlayList в методе `sortList()`, как показано ниже:

```
/**
 * Sorts the list of songs according to the specified property.
 */
public function sortList(sortProperty:SortProperty):void
{
    ...
    var sortOptions:uint;
    switch (sortProperty)
    {
        case SortProperty.TITLE:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.ARTIST:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.YEAR:
            sortOptions = Array.NUMERIC;
            break;
    }

    // Perform the actual sorting of the data.
    this._songs.sortOn(sortProperty.propertyName, sortOptions);

    // Save the current sort property.
    this._currentSort = sortProperty;

    // Record that the list is sorted.
    this._needToSort = false;
}
```

При сортировке по названию или имени исполнителя имеет смысл следовать алфавитному порядку, но при сортировке по году выпуска разумнее использовать числовой порядок. Инструкция `switch` используется для определения подходящего параметра сортировки, хранящегося в переменной `sortOptions`, о чем говорит значение, заданное в параметре `sortProperty`. Здесь члены перечисления снова используются для различения свойств, а не жестко заданных значений.

Когда параметры и свойства сортировки определены, массив `_songs` сортируется путем вызова метода `sortOn()`, который передает эти два значения в качестве параметров. Записывается текущее свойство сортировки, а также фиксируется сам факт сортировки.

## Объединение элементов массива в строку значений, разделенных символом

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Массив в данном примере используется не только для хранения списка песен в классе PlayList, но и для управления списком жанров песен в классе Song. Обратите внимание на следующий фрагмент кода из определения класса Song:

```
private var _genres:String;

public function Song(title:String, artist:String, year:uint, filename:String, genres:Array)
{
    ...
    // Genres are passed in as an array
    // but stored as a semicolon-separated string.
    this._genres = genres.join(";");
}
```

При создании нового экземпляра Song параметр `genres`, используемый для указания жанра (или жанров) песни, определяется как экземпляр Array. Это упрощает группировку нескольких жанров в единую переменную, которую затем можно передавать конструктору. Тем не менее, внутри класса Song жанры хранятся в частной переменной `_genres` в виде экземпляра String со значениями, разделенными точкой с запятой. Параметр Array преобразуется в строку значений, разделенных точкой с запятой, путем вызова метода `join()` со значением литерала строки `";"` в качестве разделителя.

По такому же принципу методы доступа `genres` поддерживают установку или извлечение жанров в виде массива:

```
    public function get genres():Array
    {
        // Genres are stored as a semicolon-separated String,
        // so they need to be transformed into an Array to pass them back out.
        return this._genres.split(";");
    }
    public function set genres(value:Array):void
    {
        // Genres are passed in as an array,
        // but stored as a semicolon-separated string.
        this._genres = value.join(";");
    }
```

Метод доступа `genresset` работает как конструктор: он принимает массив и вызывает метод `join()` для преобразования массива в строку значений, разделенных точкой с запятой. Метод доступа `get` выполняет обратную операцию: вызывается метод `split()` переменной `_genres`, и строка разбивается на массив значений с помощью заданного разделителя (литералом строки, как и прежде, является символ `";"`).

# Глава 4. Обработка ошибок

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

«Обработкой» ошибок называется добавление в приложение логики, которая реагирует на ошибку, или исправляет ее. Ошибки генерируются в ходе компиляции приложения или во время выполнения скомпилированного приложения. Когда приложение обрабатывает ошибки, при их обнаружении каждый раз происходит *некое событие*, которое является ответом на ошибку. В противоположность этому при отсутствии обработки ответ не возникает, и отправивший ошибку процесс прерывается незамеченным. При корректном использовании система обработки ошибок помогает защитить приложение пользователя от непредсказуемого поведения.

В целом обработка ошибок представляет собой многоплановую систему действий, включающую реакцию на многие виды ошибок, которые возникают как при составлении приложения, так и при его выполнении. Здесь обсуждается система обработки ошибок времени выполнения (ошибок, возникающих во время работы приложения), различные типы генерируемых ошибок и преимущества системы обработки ошибок в ActionScript 3.0.

## Основы обработки ошибок

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Ошибка времени выполнения возникает тогда, когда в коде ActionScript происходит некорректный процесс, препятствующий предполагаемому выполнению содержимого. Для обеспечения безотказной работы кода ActionScript необходимо создавать такие приложения, которые обрабатывают ошибки, то есть анализируют и устраняют их или, по крайней мере, сообщают пользователю о том, что случилось. Этот процесс называется *обработкой ошибок*.

В целом обработка ошибок представляет собой многоплановую систему действий, включающую реакцию на многие виды ошибок, которые возникают как при составлении приложения, так и при его выполнении. Ошибки, возникающие во время компиляции программы, как правило, легче идентифицировать — их обязательно нужно устранить, чтобы завершить процесс создания SWF-файла.

Ошибки выполнения обнаружить сложнее, поскольку для их возникновения необходимо, чтобы ошибочный код фактически был запущен. Если сегмент программы имеет несколько ветвей кода, например инструкцию `if..then..else`, тогда для подтверждения отсутствия ошибок в коде необходимо проверить все возможные условия со всеми возможными реальными значениями, которые может применить будущий пользователь программы.

Ошибки выполнения можно разделить на две категории: *ошибки программирования* — ошибки в коде ActionScript, такие как задание неверного типа данных для параметра метода; и *логические ошибки* — ошибки проверки данных и ошибки управления значениями, такие как использование неверной формулы для расчета процентов в банковском приложении. Ошибки обоих этих типов часто можно обнаружить и исправить заблаговременно, если аккуратно проверить приложение.

В идеале следует найти и устранить все ошибки в создаваемом приложении, прежде чем оно поступит в распоряжение конечного пользователя. Тем не менее, не все ошибки можно предвидеть и предотвратить. Например, если приложение ActionScript загружает какую-либо информацию с веб-сайта, этот процесс выходит за пределы контроля программиста. Если какой либо ресурс веб-сайта оказывается недоступен, тогда блок программы, зависящий от внешних данных, не сможет работать корректно. Самый важный аспект обработки ошибок заключается в подготовке к таким непредвиденным случаям и соответствующей реакции на них. Конечным пользователям необходимо продолжить работу с приложением или хотя бы получить сообщение об ошибке, в котором объясняется причина сбоя.

Ошибки выполнения в ActionScript представляются двумя способами.

- Классы ошибок: многие ошибки можно отнести к определенному классу. Когда возникает ошибка, среда выполнения Flash (например, Flash Player или Adobe AIR) создает экземпляр специального класса ошибок, который связан с данной конкретной ошибкой. Код программы может использовать информацию, содержащуюся в объекте ошибки, чтобы подготовить соответствующий ответ на эту ошибку.
- События ошибок: иногда ошибка возникает, когда среда выполнения Flash нормально срабатывает по событию. В этих случаях отправляется событие ошибки. Каждое событие ошибки имеет свой класс. Среда выполнения Flash передает экземпляр этого класса методам, которые приписаны к событию ошибки.

Чтобы определить, может ли конкретный метод инициировать ошибку или событие ошибки, см. раздел описания методов в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

### Важные понятия и термины

В следующем справочном списке перечислены важные термины, связанные с составлением программ для обработки ошибок.

**Асинхронность** Ситуация, при которой команда программы (например, вызов метода) не приводит к непосредственному результату. Результат (или ошибка) в этом случае принимает форму события.

**Перехват** Когда имеет место исключительная ситуация (ошибка выполнения) и когда код «знает» об этом, говорят, что код *перехватил* исключение. При перехвате исключения среда выполнения Flash прекращает уведомление другого кода ActionScript об этом исключении.

**Отладочная версия** Особая версия среды выполнения Flash, например, отладочная версия Flash Player или AIR Debug Launcher (ADL), которая содержит код уведомления пользователей об ошибках времени выполнения. В стандартных версиях Flash Player и Adobe AIR (которые использует большинство пользователей) ошибки, не обрабатываемые с помощью кода ActionScript, игнорируются. В отладочных версиях (включенных в Adobe Flash CS4 Professional и Adobe Flash Builder) при обнаружении необработанной ошибки появляется предупреждающее сообщение.

**Исключительная ситуация** Ошибка, возникающая во время работы приложения, которую среда выполнения Flash не может разрешить самостоятельно.

**Повторная генерация** При перехвате кодом исключения среда выполнения Flash больше не уведомляет о нем другие объекты. Для извещения остальных объектов об исключении важно, чтобы код мог *повторно сгенерировать* исключение и тем самым возобновлять процесс уведомления.

**Синхронность** Ситуация, при которой команда программы, например вызов метода, приводит к непосредственному результату или к немедленной генерации ошибки. При этом ответ может использоваться в одном и том же блоке кода.

**Генерация** Акт извещения среды выполнения Flash (и, как следствие, извещение других объектов и кода ActionScript) о возникновении ошибки называется *генерацией* ошибки.

# Типы ошибок

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При создании и запуске приложений программист может столкнуться с различными типами ошибок и проблемами, связанными с терминологией обработки ошибок. Ниже приводится список основных типов ошибок и соответствующих терминов.

- *Ошибки компиляции* выявляются компилятором ActionScript во время составления кода. Ошибки компиляции возникают в том случае, когда проблемы в синтаксисе кода препятствуют созданию приложения.
- *Ошибки выполнения* возникают после запуска скомпилированного приложения. Ошибки выполнения выявляются во время воспроизведения SWF-файла в среде выполнения Flash (например, в Adobe Flash Player или Adobe AIR). В большинстве случаев ошибки времени выполнения обрабатываются по мере возникновения, чтобы уведомить пользователя и предпринять меры для продолжения работы приложения. Если происходит фатальная ошибка, такая как отсутствие подключения к удаленному веб-сайту или невозможность загрузки требуемых данных, систему обработки ошибок можно использовать для корректного закрытия приложения.
- *Синхронные ошибки* возникают при выполнении программы во время вызова функции, например при попытке использовать определенный метод, аргумент которого оказывается недействительным. Эти ошибки приводят к созданию исключения средой выполнения Flash. Большинство ошибок возникает синхронно, то есть в момент выполнения инструкции. При этом процесс управления немедленно переходит к наиболее приемлемой в данном случае инструкции `catch`.

  Например, следующий фрагмент кода генерирует ошибку выполнения, потому что метод `browse()` не был вызван перед тем, как программа попыталась загрузить файл:

  ```
  var fileRef:FileReference = new FileReference();
  try
  {
      fileRef.upload(new URLRequest("http://www.yourdomain.com/fileupload.cfm"));
  }
  catch (error:IllegalOperationError)
  {
      trace(error);
      // Error #2037: Functions called in incorrect sequence, or earlier
      // call was unsuccessful.
  }
  ```

  В этом случае ошибка выполнения генерируется синхронно, поскольку Flash Player установил, что метод `browse()` не был вызван перед тем, как программа попыталась загрузить файл:

  Для получения подробной информации по этому вопросу см. раздел «Обработка синхронных ошибок в приложении» на странице 62.

- *Асинхронныеошибки* — это ошибки времени выполнения, возникающие вне нормального рабочего процесса программы. Они генерируют события, которые можно зарегистрировать прослушивателями. Асинхронные операции характеризуются тем, что функция инициирует данную операцию, но не ожидает ее завершения. Можно зарегистрировать прослушиватель событий ошибок, который будет следить за операциями приложения или пользователя. Если при выполнении какой-либо операции произойдет сбой, тогда с помощью прослушивателя событий можно будет перехватить ошибку и ответить на нее. Затем прослушиватель событий вызывает функцию обработчика событий, чтобы ответить на событие ошибки наиболее эффективным образом. Например, обработчик событий может открыть диалоговое окно, чтобы помочь пользователю устранить ошибку.

  Рассмотрите синхронную ошибку загрузки файла в примере, который был приведен выше. В случае успешного вызова метода `browse()` перед загрузкой файла Flash Player отправит несколько событий. Например, когда начнется загрузка, будет отправлено событие `open`. После успешного завершения операции загрузки файла отправляется событие `complete`. Поскольку обработка событий выполняется асинхронно (то есть она не происходит в заданное, известное или предсказуемое время), для отслеживания этих конкретных событий необходимо использовать метод `addEventListener()`, как показано в следующем коде:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.OPEN, openHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();

function selectHandler(event:Event):void
{
    trace("...select...");
    var request:URLRequest = new URLRequest("http://www.yourdomain.com/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    event.target.upload(request);
}
function openHandler(event:Event):void
{
    trace("...open...");
}
function completeHandler(event:Event):void
{
    trace("...complete...");
}
```

  Для получения подробной информации об обработке асинхронных ошибок см. раздел «Ответы на события ошибок и ошибки состояния» на странице 68.

- *Необработанные исключения* представляют собой ошибки, генерирующиеся при отсутствии соответствующей логики, которая могла бы ответить на них (например, инструкция `catch`). Если приложение генерирует ошибку, когда на текущем или более высоком уровне невозможно найти соответствующую инструкцию `catch` или обработчик событий для ответа на нее, тогда такая ошибка рассматривается как необработанное исключение.

  Когда обнаруживается неперехваченная ошибка, среда выполнения отправляет событие `uncaughtError`. Это событие также называется глобальным обработчиком ошибок. Это событие отправляется объектом UncaughtErrorEvents, который можно получить в SWF через свойство `LoaderInfo.uncaughtErrorEvents`. Если для события `uncaughtError` не зарегистрирован прослушиватель, среда выполнения игнорирует неперехваченные ошибки и пытается продолжить работу, пока ошибка не закроет SWF-файл.

Помимо отправки события `uncaughtError`, отладочные версии среды выполнения Flash реагируют не неперехваченные ошибки путем прекращения текущего сценария. После этого неперехваченная ошибка выводится на экран с помощью инструкции `trace` или записывается в файле журнала. Если исключение является экземпляром класса Error или одного из его подклассов, в выходных данных будет отображена также информация трассировки стека. Дополнительные сведения об использовании отладочной версии среды выполнения Flash см. в разделе ««Работа с отладочными версиями сред выполнения Flash» на странице 61».

***Примечание.*** *При обработке события uncaughtError, даже если событие ошибки генерируется из обработчика uncaughtError, обработчик событий вызывается несколько раз. В результате этого возникает бесконечный цикл исключений. Рекомендуется избегать подобных сценариев.*

# Обработка ошибок в ActionScript 3.0

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Поскольку многие приложения могут работать без создания логики обработки ошибок, некоторые разработчики не спешат встраивать средства обработки ошибок в свои программы. Тем не менее, без обработки ошибок приложение легко может «зависнуть» или разочаровать пользователя, если не будет работать так, как ожидалось. ActionScript 2.0 имеет класс Error, позволяющий встроить логику в пользовательские функции для генерации исключения со специальным сообщением. Поскольку обработка ошибок является критическим условием для создания удобного для пользователя приложения, ActionScript 3.0 включает в себя расширенную архитектуру перехвата ошибок.

***Примечание.*** *Хотя в справочнике ActionScript® 3.0 для платформы ® Flash® Platform перечислены исключения, выдаваемые многими методами, этот список может включать не все возможные исключения для каждого из них. Метод может генерировать исключения, связанные с ошибками синтаксиса или с другими проблемами, которые не указаны явно в описании метода, даже если в этом описании приводится список исключений, отправляемых методом.*

## Элементы обработки ошибок в ActionScript 3.0

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В ActionScript 3.0 используются многочисленные инструменты обработки ошибок, перечисленные ниже.

- Классы Error. ActionScript 3.0 включает большой набор классов Error, которые расширяют спектр ситуаций, создающих объекты ошибок. Каждый класс Error помогает приложению обрабатывать специфические состояния ошибки. Эти состояния могут относится к ошибкам системы (например, состояние MemoryError), к ошибкам кода (например, состояние ArgumentError), к ошибкам сети и коммуникации (например, состояние URIError) или к другим ошибкам. Для получения дополнительной информации по каждому классу см. раздел «Сравнение классов ошибок» на странице 71.
- Уменьшение количества скрытых сбоев. В ранних версиях Flash Player ошибки генерировались и сообщались только в том случае, когда явно использовалась инструкция `throw`. В Flash Player 9 и в более поздних версиях среды выполнения Flash собственные методы и свойства ActionScript выдают ошибки времени выполнения. Эти ошибки позволяют более эффективно обрабатывать исключения по мере возникновения, а затем отвечать на каждое из них по отдельности.

- В процессе отладки появляются точные сообщения об ошибках. В отладочных версиях среды выполнения Flash проблемные коды и ситуации генерируют обстоятельные сообщения об ошибках, которые дают возможность легко определить причины сбоя в конкретном блоке кода. Эти сообщения позволяют более эффективно исправлять ошибки. Дополнительные сведения см. в разделе ««Работа с отладочными версиями сред выполнения Flash» на странице 61».
- Четкое определение ошибок позволяет передавать пользователям точные сообщения об ошибках. В ранних версиях Flash Player метод `FileReference.upload()` возвращал логическое значение `false`, если вызов метода `upload()` был неуспешным, указывая при этом одну из пяти возможных ошибок. Если ошибка происходит при вызове метода `upload()` в ActionScript 3.0, предусмотрено четыре специфические ошибки, которые позволяют отображать более точные сообщения об ошибках для конечного пользователя.
- Точная обработка ошибок. Для многих стандартных ситуаций генерируются четко определенные ошибки. Например, в ActionScript 2.0 перед заполнением объекта FileReference свойство `name` имеет значение `null` (таким образом, перед использованием или отображением свойства `name` необходимо удостовериться, что это значение не установлено как `null`). В ActionScript 3.0 при попытке доступа к свойству `name` перед тем, как оно было заполнено, Flash Player или AIR генерирует сообщение IllegalOperationError, которое информирует о том, что значение не было задано, так что для обработки ошибки можно использовать блоки `try..catch..finally`. Дополнительные сведения см. в разделе «Использование инструкций try..catch..finally» на странице 62.
- Отсутствие существенного снижения производительности. Использование для обработки ошибок блоков `try..catch..finally` требует значительно меньше дополнительных ресурсов по сравнению с предыдущими версиями ActionScript.
- Класс ErrorEvent позволяет создавать прослушивателей для специфических событий асинхронных ошибок. Для получения подробной информации см. раздел «Ответы на события ошибок и ошибки состояния» на странице 68.

## Стратегия обработки ошибок

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если приложение не сталкивается с проблемными ситуациями, оно может успешно работать даже без встроенной в его код логики обработки ошибок. Тем не менее, при возникновении сбоев в приложении, которое не имеет активной системы обработки ошибок, конечные пользователи никогда не узнают причины этих сбоев.

Существует несколько подходов к вопросу обработки ошибок в приложении. Ниже перечислены три основных способа обработки ошибок.

- Использование инструкций `try..catch..finally`. Они позволяют перехватывать синхронные ошибки при их возникновении. Инструкции можно расположить в иерархическом порядке, чтобы перехватывать исключения на различных уровнях выполнения кода. Дополнительные сведения см. в разделе «Использование инструкций try..catch..finally» на странице 62.
- Создание пользовательских объектов ошибок. Для контроля специфических операций приложения, не предусмотренных встроенными типами ошибок, можно использовать класс Error с целью создания собственных объектов ошибок. После этого можно будет использовать инструкции `try..catch..finally` совместно с созданными объектами ошибок. Для получения дополнительной информации см. раздел «Создание пользовательских классов ошибок» на странице 67.

- Создание прослушивателей событий и обработчиков для ответа на события ошибок. Используя эту стратегию, можно создавать глобальные обработчики ошибок, которые позволяют обрабатывать подобные события без дублирования кода в блоках `try..catch..finally`. С помощью этого метода можно также с большой долей вероятности перехватывать асинхронные ошибки. Для получения подробной информации см. раздел «Ответы на события ошибок и ошибки состояния» на странице 68.

# Работа с отладочными версиями сред выполнения Flash

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Компания Adobe обеспечивает разработчиков специальными версиями сред выполнения Flash, которые имеют расширенные функции отладки. Копию отладочной версии Flash Player можно получить при установке Adobe Flash Professional или Adobe Flash Builder. Кроме того, можно получить утилиту для отладки приложений Adobe AIR, которая называется ADL, при установке любого из этих инструментов или в составе пакета Adobe AIR SDK.

Существуют существенные различия в сообщениях об ошибках между отладочными и обычными версиями Flash Player и Adobe AIR. Отладочные версии сообщают тип ошибки (Error, IOError или EOFError), номер ошибки и отображают текстовое сообщение для пользователя. Обычные версии сообщают только тип ошибки и номер ошибки. Например, рассмотрим следующий код:

```
try
{
    tf.text = myByteArray.readBoolean();
}
catch (error:EOFError)
{
    tf.text = error.toString();
}
```

Если в отладочной версии Flash Player метод `readBoolean()` генерирует ошибку EOFError, в текстовом поле `tf` отображается следующее сообщение: «EOFError: Error #2030: End of file was encountered.» (Обнаружен конец файла).

Тот же самый код в обычной версии Flash Player или Adobe AIR покажет следующее сообщение: «EOFError: Error #2030.»

***Примечание.*** *Проигрыватели отладчика транслируют событие allComplete; избегайте создания пользовательских событий с именем allComplete. В противном случай при отладке может возникать непредсказуемое поведение.*

Чтобы сократить ресурсы и размеры обычных версий, текстовые строки в сообщениях об ошибках не отображаются. Используя номер ошибки, в документации можно найти соответствующее сообщение об ошибке (см. справочник ActionScript® 3.0 для платформы Adobe® Flash® Platform). Альтернативно можно воспроизвести ошибку, используя отладочные версии Flash Player и AIR, и просмотреть полное сообщение об ошибке.

# Обработка синхронных ошибок в приложении

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Наиболее распространенной системой обработки ошибок является синхронная логика. При этом в код вставляются инструкции, которые перехватывают синхронные ошибки во время выполнения программы. Этот тип обработки ошибок позволяет приложению обнаруживать и устранять ошибки выполнения в случае функциональных сбоев. Логика перехвата синхронных ошибок включает инструкции `try..catch..finally`, которые проверяют (try) операцию, перехватывают (catch) любые ответы на ошибку от среды выполнения Flash и в заключение (finally) выполняют какую-либо другую операцию, чтобы устранить сбой программы.

## Использование инструкций try..catch..finally

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При работе с синхронными ошибками выполнения для их перехвата используются инструкции `try..catch..finally`. Когда возникает ошибка выполнения, среда выполнения Flash генерирует исключение. Это означает, что нормальное выполнение программы приостанавливается, и создается специальный объект типа Error. Затем объект Error отправляется в первый доступный блок `catch`.

Инструкция `try` закрывает группу инструкций, которые потенциально могут создавать ошибки. Инструкция `catch` всегда используется с инструкцией `try`. Если обнаруживается ошибка в одной из инструкций блока `try`, начинается выполнение инструкций `catch`, которые прикреплены к данной инструкции `try`.

Инструкция `finally` закрывает инструкции, которые будут выполняться при возникновении ошибки в блоке `try` или нет. Если ошибки нет, инструкции внутри блока `finally` выполняются после завершения выполнения инструкций блока `try`. Если ошибка есть, сначала выполняется соответствующая инструкция `catch`, затем — инструкции блока `finally`.

Следующий код демонстрирует синтаксис для использования инструкций `try..catch..finally`:

```
try
{
    // some code that could throw an error
}
catch (err:Error)
{
    // code to react to the error
}
finally
{
    // Code that runs whether an error was thrown. This code can clean
    // up after the error, or take steps to keep the application running.
}
```

Каждая инструкция `catch` идентифицирует специфический класс того исключения, которое он обрабатывает. Инструкция `catch` может задавать только те подклассы ошибок, которые принадлежат к классу Error. Каждая инструкция `catch` проверяется по порядку. Выполняется только первая инструкция `catch`, которая соответствует типу генерируемой ошибки. Другими словами, если проверяется сначала класс Error высшего уровня, а затем подкласс класса Error, тогда выбран будет только класс Error высшего уровня. Эту систему иллюстрирует следующий код:

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
```

Выше приведенный код отображает следующий вывод:

```
<Error> I am an ArgumentError
```

Чтобы корректно перехватить ошибку ArgumentError, наиболее специфические типы ошибок должны быть перечислены в начале, а наиболее общие — в конце, как показано в следующем примере:

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
```

Некоторые методы и свойства в API-интерфейсе ActionScript генерируют ошибки выполнения, если они сталкиваются с ошибками во время работы. Например, метод `close()` в классе Sound генерирует ошибку IOError, если он (метод) не в состоянии закрыть аудиопоток, как демонстрируется в следующем коде:

```
var mySound:Sound = new Sound();
try
{
    mySound.close();
}
catch (error:IOError)
{
    // Error #2029: This URLStream object does not have an open stream.
}
```

Изучив справочник ActionScript® 3.0 для платформы Adobe® Flash® Platform, вы обратите внимание на то, какие методы генерируют исключения. Это детально описано для каждого метода.

## Инструкция throw

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Среды выполнения Flash генерируют исключения, когда сталкиваются с ошибками в процессе работы вашего приложения. Кроме того, программист может генерировать определенные исключения самостоятельно, используя инструкцию `throw`. При эксплицитной генерации ошибок компания Adobe рекомендует генерировать экземпляры класса Error или его подклассов. Ниже приведен код, в котором инструкция `throw` генерирует экземпляр класса Error `MyErr`, а затем вызывает функцию `myFunction()` для ответа после генерации ошибки:

```
var MyError:Error = new Error("Encountered an error with the numUsers value", 99);
var numUsers:uint = 0;
try
{
    if (numUsers == 0)
    {
        trace("numUsers equals 0");
    }
}
catch (error:uint)
{
    throw MyError; // Catch unsigned integer errors.
}
catch (error:int)
{
    throw MyError; // Catch integer errors.
}
catch (error:Number)
{
    throw MyError; // Catch number errors.
}
catch (error:*)
{
    throw MyError; // Catch any other error.
}
finally
{
    myFunction(); // Perform any necessary cleanup here.
}
```

Обратите внимание, что инструкции `catch` упорядочены таким образом, что типы с наиболее специфичными данными следуют первыми. Если инструкция `catch` с типом данных Number следует первой, тогда ни инструкция catch с типом данных uint, ни инструкция catch с типом данных int уже не будут выполняться.

***Примечание.*** *В языке программирования Java каждая функция, генерирующая исключение, должна объявить этот факт, перечислив доступные для генерации классы в предложении `throws`, прикрепленном к объявлению функции. В ActionScript не требуется объявлять исключения, которые может генерировать функция.*

## Отображение простых сообщений об ошибках

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Одним из важнейших преимуществ новой модели исключений и событий ошибки является возможность сообщения пользователю о том, когда и почему какое-либо действие не было выполнено. Задачей программиста является написать код для отображения сообщений и предусмотреть опции для ответа.

В следующем коде показано простая инструкция `try..catch` для отображения сведений об ошибке в текстовом поле:

```
package
{
    import flash.display.Sprite;
    import flash.text.TextField;

    public class SimpleError extends Sprite
    {
        public var employee:XML =
            <EmpCode>
                <costCenter>1234</costCenter>
                <costCenter>1-234</costCenter>
            </EmpCode>;

        public function SimpleError()
        {
            try
            {
                if (employee.costCenter.length() != 1)
                {
                    throw new Error("Error, employee must have exactly one cost center assigned.");
                }
            }
            catch (error:Error)
            {
                var errorMessage:TextField = new TextField();
                errorMessage.autoSize = TextFieldAutoSize.LEFT;
                errorMessage.textColor = 0xFF0000;
                errorMessage.text = error.message;
                addChild(errorMessage);
            }
        }
    }
}
```

Используя расширенный набор классов ошибок и встроенных ошибок компиляции, ActionScript 3.0 предоставляет больше информации о сбоях по сравнению с предыдущими версиями ActionScript. Эта информация позволяет создавать более стабильные приложения с улучшенной системой обработки ошибок.

## Повторная генерация ошибок

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При создании приложений существуют различные ситуации, в которых требуется повторно генерировать ошибку, если ее не удалось должным образом обработать. Например, в следующем коде показан вложенный блок `try..catch`, который повторно генерирует пользовательскую ошибку ApplicationError, если вложенный блок `catch` не может обработать данную ошибку:

```
try
{
    try
    {
        trace("<< try >>");
        throw new ApplicationError("some error which will be rethrown");
    }
    catch (error:ApplicationError)
    {
        trace("<< catch >> " + error);
        trace("<< throw >>");
        throw error;
    }
    catch (error:Error)
    {
        trace("<< Error >> " + error);
    }
}
catch (error:ApplicationError)
{
    trace("<< catch >> " + error);
}
```

Результатом предыдущего фрагмента будет следующее:

```
<< try >>
<< catch >> ApplicationError: some error which will be rethrown
<< throw >>
<< catch >> ApplicationError: some error which will be rethrown
```

Вложенный блок `try` генерирует пользовательскую ошибку ApplicationError, которая перехватывается следующим блоком `catch`. Вложенный блок `catch` может попытаться обработать ошибку и, если это закончится неудачей, перебросить объект ApplicationError во включающий блок `try..catch`.

# Создание пользовательских классов ошибок

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В ActionScript можно расширить один из стандартных классов ошибок, создав таким образом свой собственный специализированный класс. Существует несколько причин для создания собственных классов ошибок.

- Создание класса для идентификации специфических ошибок или их групп, присущих только для данного приложения.

  Например, предпринимайте различные действия для обработки ошибок, генерируемых собственным кодом программиста, в добавление к тем ошибкам, которые перехватываются средой выполнения Flash. Можно создать подкласс класса Error для отслеживания новых типов данных ошибок в блоках `try..catch`.

- Создание класса для обеспечения уникальных возможностей отображения ошибок, генерируемых пользовательским приложением.

  Например, можно создать новый метод `toString()`, который форматирует сообщения об ошибках определенным образом. Можно также задать метод `lookupErrorString()`, который принимает код ошибки и извлекает из него соответствующее сообщение, базируясь на языковых предпочтениях пользователя.

Специализированный класс ошибки должен расширять основной класс Error ActionScript. Ниже приведен пример специализированного класса AppError, который является расширением класса Error:

```
public class AppError extends Error
{
    public function AppError(message:String, errorID:int)
    {
        super(message, errorID);
    }
}
```

Следующий пример иллюстрирует использование AppError в пользовательском проекте:

```
try
{
    throw new AppError("Encountered Custom AppError", 29);
}
catch (error:AppError)
{
    trace(error.errorID + ": " + error.message)
}
```

***Примечание.*** *Если требуется переопределить метод `Error.toString()` в пользовательском подклассе, задайте для него один параметр `...` (rest). Так определяет метод `Error.toString()` языковая спецификация ECMAScript, на базе которой построен язык ActionScript 3.0, и для обеспечения обратной совместимости ActionScript 3.0 определяет его так же. Таким образом, при замене метода `Error.toString()` необходимо обеспечить точное соответствие параметров. Не имеет смысла задавать параметры для метода `toString()` во время выполнения программы, поскольку в этом режиме они игнорируются.*

# Ответы на события ошибок и ошибки состояния

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Одним из наиболее значительных улучшений обработки ошибок в ActionScript 3.0 является поддержка реагирования на асинхронные ошибки при выполнении программы. (Определение асинхронных ошибок см. в разделе «Типы ошибок» на странице 57).

Для ответа на события ошибок можно создавать прослушивателей событий и обработчиков событий. Многие классы отправляют события ошибок тем же способом, что и другие события. Например, экземпляр класса XMLSocket при нормальных обстоятельствах отправляет события трех типов: `Event.CLOSE`, `Event.CONNECT` и `DataEvent.DATA`. Тем не менее, в случае возникновения проблемы класс XMLSocket может отправлять события `IOErrorEvent.IOError` или `SecurityErrorEvent.SECURITY_ERROR`. Дополнительные сведения о прослушивателях и обработчиках событий см. в разделе «Обработка событий» на странице 134.

События ошибок подразделяются на две категории.

- События ошибок, которые расширяют класс ErrorEvent

  Класс flash.events.ErrorEvent содержит свойства и методы для обработки ошибок выполнения, относящихся к сетевым и коммуникационным операциям. Классы AsyncErrorEvent, IOErrorEvent и SecurityErrorEvent расширяют класс ErrorEvent. При использовании отладочных версий среды выполнения Flash диалоговое окно информирует пользователя о всех событиях ошибок, с которыми сталкивается проигрыватель, без применения функций прослушивателя.

- События ошибок, относящиеся к состоянию

  События ошибок, относящиеся к состоянию, связаны со свойствами `netStatus` и `status` классов сети и коммуникаций. Если среда выполнения Flash сталкивается с проблемой при чтении или записи данных, для свойств `netStatus.info.level` или `status.level` (в зависимости от класса используемого объекта) устанавливается значение `"error"`. Для ответа на эту ошибку необходимо проверить, содержит ли свойство `level` значение `"error"` в функции обработчика событий.

## Работа с событиями ошибок

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс ErrorEvent и его подклассы содержат типы ошибок, используемые для обработки ошибок, отправляемых средой выполнения Flash, когда они пытаются считывать или записывать данные.

В следующем примере используются и инструкция `try..catch`, и обработчики событий ошибки для отображения всех ошибок, обнаруженных в процессе чтения локального файла. Можно добавить более сложный код обработки для предоставления опций пользователю или для автоматической обработки ошибок в местах, обозначенных комментарием «your error-handling code here» (введите код обработки ошибок).

```
package
{
    import flash.display.Sprite;
    import flash.errors.IOError;
    import flash.events.IOErrorEvent;
    import flash.events.TextEvent;
    import flash.media.Sound;
    import flash.media.SoundChannel;
    import flash.net.URLRequest;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;

    public class LinkEventExample extends Sprite
    {
        private var myMP3:Sound;
        public function LinkEventExample()
        {
            myMP3 = new Sound();
            var list:TextField = new TextField();
            list.autoSize = TextFieldAutoSize.LEFT;
            list.multiline = true;
            list.htmlText = "<a href=\"event:track1.mp3\">Track 1</a><br>";
            list.htmlText += "<a href=\"event:track2.mp3\">Track 2</a><br>";
            addEventListener(TextEvent.LINK, linkHandler);
            addChild(list);
        }

        private function playMP3(mp3:String):void
        {
            try
            {
                myMP3.load(new URLRequest(mp3));
                myMP3.play();
            }
            catch (err:Error)
            {
                trace(err.message);
                // your error-handling code here
            }
            myMP3.addEventListener(IOErrorEvent.IO_ERROR, errorHandler);
        }

        private function linkHandler(linkEvent:TextEvent):void
        {
            playMP3(linkEvent.text);
            // your error-handling code here
        }

        private function errorHandler(errorEvent:IOErrorEvent):void
        {
            trace(errorEvent.text);
            // your error-handling code here
        }
    }
}
```

## Работа с событиями изменения состояния

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Среды выполнения Flash динамически изменяют значение свойств `netStatus.info.level` или `status.level` для классов, поддерживающих свойство `level` во время выполнения приложения. Классами, имеющими свойство `netStatus.info.level`, являются: NetConnection, NetStream и SharedObject. Классами, имеющими свойство `status.level`, являются: HTTPStatusEvent, Camera, Microphone и LocalConnection. Можно написать функцию обработчика для ответа на изменения значения `level` и отслеживания ошибок коммуникации.

В следующем примере функция `netStatusHandler()` используется для проверки значения свойства `level`. Если свойство `level` указывает на возникновение ошибки, код трассирует сообщение: «Video stream failed» (прерывание видеопотока).

```
package
{
    import flash.display.Sprite;
    import flash.events.NetStatusEvent;
    import flash.events.SecurityErrorEvent;
    import flash.media.Video;
    import flash.net.NetConnection;
    import flash.net.NetStream;

    public class VideoExample extends Sprite
    {
        private var videoUrl:String = "Video.flv";
        private var connection:NetConnection;
        private var stream:NetStream;

        public function VideoExample()
        {
            connection = new NetConnection();
            connection.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
            connection.addEventListener(SecurityErrorEvent.SECURITY_ERROR, securityErrorHandler);
            connection.connect(null);
        }

        private function netStatusHandler(event:NetStatusEvent):void
        {
            if (event.info.level == "error")
            {
                trace("Video stream failed")
            }
            else
```

```
            {
                connectStream();
            }
        }

        private function securityErrorHandler(event:SecurityErrorEvent):void
        {
            trace("securityErrorHandler: " + event);
        }

        private function connectStream():void
        {
            var stream:NetStream = new NetStream(connection);
            var video:Video = new Video();
            video.attachNetStream(stream);
            stream.play(videoUrl);
            addChild(video);
        }
    }
}
```

# Сравнение классов ошибок

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

ActionScript содержит определенное количество предопределенных классов Error. Однако вы тоже можете использовать такие же классы Error в своем коде. В ActionScript 3.0 есть два основных типа классов ошибок: базовые классы Error и пакет классов flash.error. Пакет flash.error содержит дополнительные классы, которые облегчают процесс разработки и отладки приложений ActionScript 3.0.

## Базовые классы Error

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Базовые классы Error включают в себя: Error, ArgumentError, EvalError, RangeError, ReferenceError, SecurityError, SyntaxError, TypeError, URIError и VerifyError. Все эти классы расположены на верхних уровнях пространства имен.

| Имя класса | Описание | Примечания |
|---|---|---|
| Error | Класс Error используется для генерации исключений. Он является базовым классом для других классов исключений, определенных в ECMAScript: EvalError, RangeError, ReferenceError, SyntaxError, TypeError и URIError. | Класс Error является базовым для всех ошибок выполнения. Его рекомендуется использовать в качестве основы для пользовательских классов ошибок. |
| ArgumentError | Класс ArgumentError представляет ошибку, которая возникает, если значения параметров, переданные при вызове функции, не совпадают с параметрами, определенными для этой функции. | Ниже приведены примеры ошибок аргумента.<br>• Слишком мало или слишком много аргументов передается методу.<br>• Ожидается, что аргумент будет принадлежать к перечислимому типу, но он не оказывается таковым. |
| EvalError | Исключение EvalError генерируется, если какие-либо параметры передаются конструктору функции класса, а также если код пользователя вызывает функцию `eval()`. | В ActionScript 3.0 отсутствует поддержка функции `eval()`, поэтому попытка ее использования приводит к генерации ошибки.<br>В более ранних версиях Flash Player функция `eval()` использовалась для доступа с помощью имени к переменным, свойствам, объектам или фрагментам роликов. |
| RangeError | Исключение RangeError генерируется, если числовое значение выходит за пределы допустимого диапазона. | Например, RangeError генерируется классом Timer, если задержка имеет отрицательное или бесконечное значение. RangeError генерируется также при попытке добавить экранный объект с недопустимой глубиной. |
| ReferenceError | Исключение ReferenceError генерируется при попытке выполнить ссылку на неопределенное свойство для защищенного (нединамического) объекта. Версии компилятора до ActionScript 3.0 не генерировали ошибку при попытке доступа к свойству `undefined`. Однако в такой ситуации ActionScript 3.0 выдает исключение ReferenceError. | Исключения для не определенных переменных указывают на потенциальные ошибки и помогают повысить качество программного продукта. Однако тем программистам, которые не имеют привычки инициализировать свои переменные, придется несколько изменить свою манеру написания кода в связи с этими нововведениями в ActionScript. |
| SecurityError | Исключение SecurityError генерируется, если имеет место нарушение среды безопасности и в доступе отказано. | Ниже приведены примеры ошибок безопасности.<br>• Несанкционированный доступ к свойству или вызов метода, выполненные из-за границы изолированной программной среды.<br>• Предпринята попытка доступа к URL, не разрешенному изолированной программной средой.<br>• Предпринята попытка подключения к сокету, но у данного порта не было необходимого файла политики.<br>• Предпринята попытка доступа к камере или микрофону пользователя, и он запретил доступ к устройству. |
| SyntaxError | Исключение SyntaxError генерируется, если в коде ActionScript имеет место ошибка синтаксического разбора. | Исключение SyntaxError генерируется при нижеследующих обстоятельствах.<br>• ActionScript генерирует исключение SyntaxError, когда класс RegExp пытается проанализировать недопустимое регулярное выражение.<br>• ActionScript генерирует исключение SyntaxError, когда класс XMLDocument пытается проанализировать недопустимый объект XML. |

| Имя класса | Описание | Примечания |
|---|---|---|
| TypeError | Исключение TypeError генерируется, если фактический тип операнда отличается от ожидаемого типа. | Исключение TypeError генерируется при следующих обстоятельствах:<br>• Фактический параметр функции или метода не удается принудительно привязать к типу формального параметра.<br>• Переменной присваивается значение, которое нельзя принудительно привязать к типу переменной.<br>• Правая сторона оператора `is` или `instanceof` не принадлежит действительному типу.<br>• Ключевое слово `super` используется неправомерно.<br>• При поиске свойства найдено более одной привязки, и из-за этого результаты противоречивы.<br>• Метод вызван для несовместимого объекта. Например, исключение TypeError генерируется, если метод класса RegExp «прививается» на родовой объект, а затем вызывается. |
| URIError | Исключение URIError генерируется, если способ использования одной из глобальных функций обработки URI несовместим с ее определением. | Исключение URIError генерируется при следующих обстоятельствах:<br>Недействительный URI указан для функции программного интерфейса (API) проигрывателя Flash Player, которая ожидает URI типа `Socket.connect()`. |
| VerifyError | Исключение VerifyError генерируется, если обнаружен некорректный или поврежденный SWF-файл. | Когда один SWF-файл загружает другой SWF-файл, родительский файл может перехватить исключение VerifyError, генерируемое загруженным файлом. |

## Пакет классов ошибок flash.error

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Пакет flash.error содержит классы Error, которые считаются частью интерфейса среды выполнения Flash. В отличие от вышеописанных классов пакет flash.error связан с теми событиями ошибок, которые характерны для сред выполнения Flash (например, Flash Player или Adobe AIR).

| Имя класса | Описание | Примечания |
|---|---|---|
| EOFError | Исключение EOFError генерируется при попытке чтения за пределом доступных данных. | Например, ошибка EOFError генерируется, когда вызывается один из методов чтения в интерфейсе IDataInput, а данных недостаточно для удовлетворения запроса на чтение. |
| IllegalOperationError | Исключение IllegalOperationError генерируется, если метод не реализован или реализация не рассчитана на текущее применение. | Ниже приведены примеры ошибок для запрещенных операций.<br>• Базовый класс, такой как DisplayObjectContainer, обеспечивает больше функций, чем рабочая область в состоянии поддерживать. Например, при попытке получить или задать маску в рабочей области (используя `stage.mask`) среда выполнения Flash генерирует ошибку IllegalOperationError вместе с сообщением: The Stage class does not implement this property or method (Рабочая область не поддерживает это свойство или метод).<br>• Подкласс наследует метод, который ему не требуется и который он может не поддерживать.<br>• Определенные методы специальных возможностей вызываются, когда компиляция проигрывателя Flash Player не поддерживает специальные возможности.<br>• Специальные функции разработки вызываются в обычной версии проигрывателя Flash Player (с неполным набором функций).<br>• Пользователь пытается задать имя объекта, находящегося на временной шкале. |
| IOError | Исключение IOError генерируется при некоторых ошибках ввода-вывода. | Например, эта ошибка возникает при попытке чтения/записи на сокете, который не был подключен или был отключен. |

| Имя класса | Описание | Примечания |
|---|---|---|
| MemoryError | Исключение MemoryError генерируется при ошибке выполнения запроса на выделение памяти. | По умолчанию виртуальная машина ActionScript Virtual Machine 2 не накладывает ограничений на выделение памяти программой ActionScript. На персональных компьютерах отказы при выделении памяти происходят редко. Ошибка генерируется, когда система не может выделить достаточно памяти для выполнения операции. Следовательно, на персональных компьютерах это исключение возникает редко, если только не требуется чрезмерный объем памяти. Например, запрос на 3 ГБ выполнить невозможно, поскольку 32-разрядная программа Microsoft ® Windows® может получить доступ только к 2 ГБ адресного пространства. |
| ScriptTimeoutError | Исключение ScriptTimeoutError генерируется, если время простоя сценария достигает 15 секунд. Перехват исключения ScriptTimeoutError позволяет решить проблему простоя сценария более корректно. При отсутствии обработчика исключения обработчик необработанных исключений отображает диалоговое окно с сообщением об ошибке. | Для предотвращения преднамеренного перехвата исключения с созданием бесконечного цикла только первое исключение, генерируемое в рамках определенного сценария, может быть перехвачено. Последующее исключение ScriptTimeoutError не может быть перехвачено кодом разработчика и немедленно отправляется к обработчику необработанных исключений. |
| StackOverflowError | Исключение StackOverflowError генерируется, если стек, доступный для сценария, полностью использован. | Исключение StackOverflowError может указывать на возникновение бесконечной рекурсии. |

# Пример обработки ошибок: программа CustomErrors

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Приложение CustomErrors демонстрирует методы работы с пользовательскими ошибками при создании программы. Рассматриваются следующие техники:

- Подтверждение пакета XML
- Создание пользовательской ошибки
- Генерация пользовательской ошибки
- Уведомление пользователей о генерации ошибки

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения CustomErrors находятся в папке Samples/CustomError. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| CustomErrors.mxml<br>или<br>CustomErrors.fla | Основной файл приложения Flash (FLA) или Flex (MXML) |
| com/example/programmingas3/errors/ApplicationError.as | Класс, который служит базой для классов FatalError и WarningError. |
| com/example/programmingas3/errors/FatalError.as | Класс, который определяет ошибку FatalError, генерируемую приложением. Этот класс расширяет пользовательский класс ApplicationError. |
| com/example/programmingas3/errors/Validator.as | Класс, определяющий отдельный метод, который подтверждает поставляемый пользователем служебный пакет XML. |
| com/example/programmingas3/errors/WarningError.as | Класс, который определяет ошибку WarningError, генерируемую приложением. Этот класс расширяет пользовательский класс ApplicationError. |

## Рассмотрение приложения CustomErrors

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При загрузке приложений Flex вызывается метод `initApp()`, а при загрузке приложений Flash Professional выполняется код временной шкалы (при отсутствии функций). Этот код определяет образец пакета XML, который необходимо проверить с помощью класса Validator. Выполняется следующий код:

```
employeeXML =
    <employee id="12345">
        <firstName>John</firstName>
        <lastName>Doe</lastName>
        <costCenter>12345</costCenter>
        <costCenter>67890</costCenter>
    </employee>;
}
```

Пакет XML позже отображается в экземпляре компонента TextArea в рабочей области. Это позволяет модифицировать пакет XML перед попыткой его повторного подтверждения.

При нажатии кнопки Validate (Подтвердить) вызывается метод `validateData()`. Этот метод проверяет служебный пакет XML (список служащих) с помощью метода `validateEmployeeXML()` класса Validator. Следующий код демонстрирует метод `validateData()`:

```
function validateData():void
{
    try
    {
        var tempXML:XML = XML(xmlText.text);
        Validator.validateEmployeeXML(tempXML);
        status.text = "The XML was successfully validated.";
    }
    catch (error:FatalError)
    {
        showFatalError(error);
    }
    catch (error:WarningError)
    {
        showWarningError(error);
    }
    catch (error:Error)
    {
        showGenericError(error);
    }
}
```

Прежде всего создается временный объект XML с помощью содержимого экземпляра компонента TextArea — `xmlText`. Затем вызывается метод `validateEmployeeXML()` в пользовательском классе Validator com.example.programmingas3/errors/Validator.as), который передает временный объект XML в качестве параметра. Если пакет XML оказывается действительным, экземпляр `status` компонента Label отображает подтверждающее сообщение и приложение закрывается. Если метод `validateEmployeeXML()` генерирует пользовательскую ошибку (то есть FatalError, WarningError или базовую ошибку Error), выполняется соответствующая инструкция `catch` , которая вызывает один из следующих методов: `showFatalError()`, `showWarningError()` или `showGenericError()`. Каждый из этих методов отображает в текстовой области с именем `statusText` соответствующее сообщение для уведомления пользователя о возникновении специфической ошибки. Каждый метод обновляет также экземпляр `status` компонента Label специальным сообщением.

Если при попытке подтверждения служебного пакета XML происходит фатальная ошибка, в текстовой области `statusText` отображается сообщение об ошибке, а экземпляр `xmlText` компонента TextArea и экземпляр `validateBtn` компонента Button отключаются, как показано в следующем коде:

```
function showFatalError(error:FatalError):void
{
    var message:String = error.message + "\n\n";
    var title:String = error.getTitle();
    statusText.text = message + " " + title + "\n\nThis application has ended.";
    this.xmlText.enabled = false;
    this.validateBtn.enabled = false;
    hideButtons();
}
```

Если вместо фатальной ошибки происходит предупреждающая ошибка, в текстовой области `statusText` также отображается сообщение об ошибке, но экземпляры компонентов TextArea (`xmlText`) и Button не отключаются. Метод `showWarningError()` отображает сообщение о пользовательской ошибке в текстовой области `statusText`. Также в сообщении пользователю предлагается сделать выбор: продолжить подтверждение XML или отменить выполнение сценария. Следующий фрагмент демонстрирует метод `showWarningError()`:

```
function showWarningError(error:WarningError):void
{
    var message:String = error.message + "\n\n" + "Do you want to exit this application?";
    showButtons();
    var title:String = error.getTitle();
    statusText.text = message;
}
```

Метод `closeHandler()` вызывается при нажатии кнопок «Да» или «Нет». Следующий фрагмент демонстрирует метод `closeHandler()`:

```
function closeHandler(event:CloseEvent):void
{
    switch (event.detail)
    {
        case yesButton:
            showFatalError(new FatalError(9999));
            break;
        case noButton:
            statusText.text = "";
            hideButtons();
            break;
    }
}
```

Если пользователь решает отменить выполнение сценария нажатием кнопки «Да», генерируется фатальная ошибка FatalError, что приводит к закрытию приложения.

## Создание пользовательского валидатора

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Пользовательский класс Validator содержит один метод `validateEmployeeXML()`. Метод `validateEmployeeXML()` использует один аргумент `employee`, который представляет собой проверяемый пакет XML. Метод `validateEmployeeXML()` имеет следующую структуру:

```
public static function validateEmployeeXML(employee:XML):void
{
    // checks for the integrity of items in the XML
    if (employee.costCenter.length() < 1)
    {
        throw new FatalError(9000);
    }
    if (employee.costCenter.length() > 1)
    {
        throw new WarningError(9001);
    }
    if (employee.ssn.length() != 1)
    {
        throw new FatalError(9002);
    }
}
```

Для утверждения служащий должен принадлежать к одному (и только одному) центру учета затрат. Если служащий не принадлежит ни к одному центру учета затрат, метод генерирует неустранимую ошибку FatalError, которая передается методу `validateData()` в главном файле приложения. Если служащий принадлежит к нескольким центрам учета затрат, генерируется предупреждающая ошибка WarningError. Окончательная проверка в валидаторе XML приводит к тому, что пользователь получает только один номер социального страхования (узел `ssn` в пакете XML). Если обнаружить только один узел `ssn` не удается, генерируется фатальная ошибка FatalError.

К методу `validateEmployeeXML()` можно добавить дополнительные проверки. Например, подтверждение того, что узел `ssn` содержит действительный номер или что служащий имеет хотя бы один телефонный номер и адрес электронной почты, а также что оба значения являются действительными. Можно также модифицировать XML таким образом, что каждый служащий и его менеджер будут иметь уникальные идентификационные номера.

## Определение класса ApplicationError

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс ApplicationError служит основой для классов FatalError и WarningError. Класс ApplicationError является расширением класса Error. Он определяет свои специальные методы и свойства, включая идентификаторы ошибок и степень их серьезности, а также объект XML, содержащий коды и сообщения пользовательских ошибок. Этот класс также включает две статические постоянные, которые используются для определения степени серьезности для каждого типа ошибок.

Метод конструктора класса ApplicationError имеет следующую структуру:

```
public function ApplicationError()
{
    messages =
        <errors>
            <error code="9000">
                <![CDATA[Employee must be assigned to a cost center.]]>
            </error>
            <error code="9001">
                <![CDATA[Employee must be assigned to only one cost center.]]>
            </error>
            <error code="9002">
                <![CDATA[Employee must have one and only one SSN.]]>
            </error>
            <error code="9999">
                <![CDATA[The application has been stopped.]]>
            </error>
        </errors>;
}
```

Каждый узел ошибки в объекте XML содержит уникальный числовой код и сообщение об ошибке. Сообщения об ошибках легко просмотреть по их коду с помощью E4X, как показано в следующем методе `getMessageText()`:

```
public function getMessageText(id:int):String
{
    var message:XMLList = messages.error.(@code == id);
    return message[0].text();
}
```

Метод `getMessageText()` принимает один целочисленный аргумент `id` и возвращает строку. Аргумент `id` является кодом ошибки, по которому эту ошибку можно найти. Например, поиск по `id` 9001 выдает ошибку, в сообщении которой говорится, что каждый служащий должен приписываться только к одному центру учета затрат. Если несколько ошибок имеют одинаковый код, ActionScript возвращает сообщение об ошибке только для первого найденного результата (`message[0]` в возвращенном объекте XMLList).

Следующий метод в этом классе, а именно `getTitle()`, не принимает никаких параметров и возвращает строковое значение, содержащее идентификатор для данной специфической ошибки. Это значение позволяет пользователю легко определить, какая именно ошибка произошла в процессе проверки пакета XML. Следующий фрагмент демонстрирует метод `getTitle()`:

```
public function getTitle():String
{
    return "Error #" + id;
}
```

Последним методом в классе ApplicationError является `toString()`. Этот метод заменяет функцию, определенную в классе Error, позволяя настроить передачу сообщения об ошибке. Этот метод возвращает строку, которая определяет номер специфической ошибки и сообщение о том, что произошло.

```
public override function toString():String
{
    return "[APPLICATION ERROR #" + id + "] " + message;
}
```

## Определение класса FatalError

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс FatalError является расширением пользовательского класса ApplicationError. Он определяет три метода: конструктор FatalError, `getTitle()` и `toString()`. Первый метод, конструктор FatalError, принимает один целочисленный аргумент, `errorID`, задает степень серьезности ошибки с помощью значений статической постоянной, определенной в классе ApplicationError, и получает сообщение о специфической ошибке, вызывая метод `getMessageText()` класса ApplicationError. Конструктор FatalError имеет следующую структуру:

```
public function FatalError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.FATAL;
    message = getMessageText(errorID);
}
```

Следующий метод в классе FatalError, `getTitle()`, заменяет метод `getTitle()`, определенный ранее в классе ApplicationError, и дополняет текст заголовком «FATAL», уведомляя таким образом пользователя о фатальной ошибке. Метод `getTitle()` имеет следующую структуру:

```
public override function getTitle():String
{
    return "Error #" + id + " -- FATAL";
}
```

Последний метод в этом классе, `toString()`, заменяет метод `toString()`, определенный в классе ApplicationError. Метод `toString()` имеет следующую структуру:

```
public override function toString():String
{
    return "[FATAL ERROR #" + id + "] " + message;
}
```

## Определение класса WarningError

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс WarningError является расширением класса ApplicationError. Он практически идентичен классу FatalError, за исключением пары небольших изменений в строках и установки серьезности ошибки ApplicationError.WARNING вместо ApplicationError.FATAL, как показано в следующем коде:

```
public function WarningError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.WARNING;
    message = super.getMessageText(errorID);
}
```

# Глава 5. Использование регулярных выражений

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Регулярное выражение описывает образец, используемый для нахождения в строках совпадающего текста и управления им. Регулярные выражения похожи на строки, но отличие в том, что они могут иметь особые коды для описания образцов и повторений. Например, следующее регулярное выражение сопоставляет строки, начинающиеся с символа A, за которым следует одна или несколько последовательных цифр:

```
/A\d+/
```

В следующих разделах описан основной синтаксис построения регулярных выражений. Однако с регулярными выражениями может быть связано много сложностей и нюансов. Дополнительные сведения о регулярных выражениях можно найти в Интернете и печатных изданиях. Следует помнить о том, что разные среды программирования по-разному работают с регулярными выражениями. ActionScript 3.0 работает с регулярными выражениями согласно определению, данному в спецификации к языку ECMAScript версии 3 (ECMA-262).

### Дополнительные разделы справки

RegExp

## Основы регулярных выражений

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Регулярное выражение описывает образец символов. Регулярные выражения обычно используются для проверки соответствия текстового значения определенному образцу (как, например, проверка наличия в телефонном номере, набранном пользователем, правильного числа цифр) или для замены частей текстового значения, соответствующих определенному образцу.

Регулярные выражения могут быть простыми. Например, предположим, что требуется подтвердить, что определенная строка соответствует «ABC», или заменить каждое появление в строке «ABC» другим текстом. В этом случае можно использовать следующее регулярное выражение, определяющее образец, состоящий из букв A, B и C, следующих друг за другом:

```
/ABC/
```

Следует отметить, что литерал регулярного выражения отделяется символом прямой косой черты `/`.

Образцы регулярных выражений также могут быть сложными и иногда имеют зашифрованный вид, как, например, у следующего выражения, что в данном случае необходимо для подбора допустимого адреса электронной почты:

```
/([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-Z]{2,6}/
```

Чаще всего регулярные выражения используются для поиска образцов в строках и замены символов. В таких случаях создается объект регулярного выражения и используется в качестве параметра для одного или нескольких методов класса String. Следующие методы класса String используют регулярные выражения в качестве параметров: `match()`, `replace()`, `search()` и `split()`. Дополнительные сведения об этих методах см. в разделе ««Поиск шаблонов в строках и замена подстрок» на странице 17».

Класс RegExp включает следующие методы: `test()` и `exec()`. Дополнительные сведения см. в разделе ««Методы для использования регулярных выражений со строками» на странице 97».

**Важные понятия и термины**

Ниже приводится список важных терминов, относящихся к этой функции.

**Символ перехода** Символ, указывающий на то, что последующий символ должен рассматриваться как метасимвол, а не как символ литерала. В синтаксисе регулярного выражения символом перехода является обратная косая черта (\), поэтому обратная косая черта, после которой идет другой символ, является особым кодом, а не просто символом.

**Флаг** Символ, указывающий вариант использования образца регулярного выражения, как, например, необходимость в различии между символами верхнего и нижнего регистров.

**Метасимвол** Символ, имеющий особое значение в образце регулярного выражения в отличие от буквального представления данного символа в образце.

**Квантификатор** Символ (или несколько символов), указывающий число повторений части образца. Например, квантификатор может использоваться для обозначения того, что почтовый индекс Соединенных Штатов должен содержать пять или девять чисел.

**Регулярное выражение** Инструкция программы, определяющая образец символов, используемый для подтверждения совпадения других строк данному образцу или для замены частей строки.

# Синтаксис регулярных выражений

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В данном разделе описываются все элементы синтаксиса регулярных выражений ActionScript. Как будет видно далее, с регулярными выражениями может быть связано много сложностей и нюансов. Дополнительные сведения о регулярных выражениях можно найти в Интернете и печатных изданиях. Следует помнить о том, что разные среды программирования по-разному работают с регулярными выражениями. ActionScript 3.0 работает с регулярными выражениями согласно определению, данному в спецификации к языку ECMAScript версии 3 (ECMA-262).

Обычно используются регулярные выражения, соответствующие более сложным образцам, чем простая строка символов. Например, следующее регулярное выражение определяет образец, состоящий из букв A, B и C, следующих друг за другом и имеющих после себя любую цифру:

```
/ABC\d/
```

Код `\d` обозначает «любую цифру». Символ обратной косой черты (`\`) называется символом перехода и в сочетании с символом, следующим за ним (в данном случае это буква d), имеет в регулярном выражении особое значение.

Следующее регулярное выражение определяет образец букв ABC, за которыми следует любое число цифр (обратите внимание на наличие звездочки):

```
/ABC\d*/
```

Символ звездочки (`*`) является *метасимволом*. Метасимвол представляет собой символ, имеющий в регулярном выражении особое значение. Звездочка является особым типом метасимвола, называемым *квантификатор* и используемым с целью определения количества повторений символа или группы символов. Дополнительные сведения см. в разделе ««Квантификаторы» на странице 89».

Помимо образца в состав регулярного выражения могут входить флаги, которые определяют способ сопоставления регулярного выражения. Например, в следующем регулярном выражении используется флаг `i`, который указывает на то, что данное регулярное выражение при сопоставлении строк не учитывает регистр.

```
/ABC\d*/i
```

Дополнительные сведения см. в разделе ««Флаги и свойства» на странице 93».

Регулярные выражения можно использовать со следующими методами класса String: `match()`, `replace()` и `search()`. Дополнительные сведения об этих методах см. в разделе ««Поиск шаблонов в строках и замена подстрок» на странице 17».

## Создание экземпляра регулярного выражения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Экземпляр регулярного выражения можно создать двумя способами. При одном способе для разделения регулярного выражения используется символ прямой косой черты (`/`), при другом — конструктор `new`. Например, эквивалентными являются следующие регулярные выражения:

```
var pattern1:RegExp = /bob/i;
var pattern2:RegExp = new RegExp("bob", "i");
```

Прямые косые черты отделяют регулярное выражение так же, как кавычки отделяют строковый литерал. Часть регулярного выражения, находящаяся внутри прямых косых черт, определяет *образец*. Регулярное выражение также может иметь в своем составе *флаги*, расположенные после конечной отделяющей черты. Эти флаги считаются частью регулярного выражения, но отделены от его образца.

При использовании конструктора `new` для определения регулярного выражения используются две строки. Первая строка определяет образец, а вторая строка — флаги, как показано в следующем примере:

```
var pattern2:RegExp = new RegExp("bob", "i");
```

При включении прямой косой черты *в состав* регулярного выражения, определенного использованием ограничителей прямой косой черты, прямой косой черте должен предшествовать символ перехода (`\`) обратной косой черты. Например, образцу `1/2` соответствует следующее регулярное выражение:

```
var pattern:RegExp = /1\/2/;
```

Чтобы включить кавычки *в состав* регулярного выражения, определенного конструктором `new`, перед кавычками необходимо добавить символ перехода`\` обратной косой черты (так же, как при определении любого строкового литерала). Например, образцу `eat at "joe's"` соответствуют следующие регулярные выражения:

```
var pattern1:RegExp = new RegExp("eat at \"joe's\"", "");
var pattern2:RegExp = new RegExp('eat at "joe\'s"', "");
```

В регулярных выражениях, определяемых использованием ограничителей прямой косой черты, не используйте символ обратной косой черты с кавычками. Аналогичным образом, в регулярных выражениях, определяемых конструктором `new`, не используйте символ перехода с прямыми косыми чертами. Следующие регулярные выражения эквивалентны; они определяют образец `1/2 "joe's"`:

```
var pattern1:RegExp = /1\/2 "joe's"/;
var pattern2:RegExp = new RegExp("1/2 \"joe's\"", "");
var pattern3:RegExp = new RegExp('1/2 "joe\'s"', '');
```

Также в регулярном выражении, которое определяется конструктором `new`, чтобы воспользоваться метапоследовательностью, которая начинается с символа обратной косой черты (`\`) (например, `\d`, что соответствует любой цифре), дважды введите символ обратной косой черты:

```
var pattern:RegExp = new RegExp("\\d+", ""); // matches one or more digits
```

В данном случае необходимо ввести символ обратной косой черты дважды, поскольку первый параметр метода конструктора `RegExp()` является строкой. В строковом литерале также необходимо дважды ввести символ обратной косой черты, чтобы он распознавался как единичный символ.

В разделах, приведенных ниже, описывается синтаксис для определения образцов регулярных выражений.

Дополнительные сведения о флагах см. в разделе ««Флаги и свойства» на странице 93».

## Символы, метасимволы и метапоследовательности

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Самым простым регулярным выражением является такое выражение, последовательность символов которого соответствует последовательности, указанной в следующем примере:

```
var pattern:RegExp = /hello/;
```

Однако следующие символы, известные как метасимволы ,, имеют в регулярных выражениях особое значение:

```
^ $ \ . * + ? ( ) [ ] { } |
```

Например, следующее регулярное выражение соответствует букве A, за которой следует нуль или несколько экземпляров буквы B (метасимвол звездочки обозначает это повторение), за которыми, в свою очередь, идет буква C:

```
/AB*C/
```

Чтобы в образец регулярного выражения добавить метасимвол без его особого значения, необходимо использовать символ перехода \ обратной косой черты. Например, следующее регулярное выражение соответствует букве A, за которой следует буква B, за которой идет звездочка, за которой, в свою очередь, идет буква C:

```
var pattern:RegExp = /AB\*C/;
```

*Метапоследовательность*, как и метасимвол, имеет в регулярном выражении особое значение. Метапоследовательность состоит из более чем одного символа. В следующих разделах подробно описывается использование метасимволов и метапоследовательностей.

### О метасимволах

В следующей таблице собраны метасимволы, которые могут быть использованы в регулярных выражениях.

| **Метасимвол** | **Описание** |
|---|---|
| ^ (знак вставки) | Соответствует началу строки. Вместе с установленным флагом `m` (`multiline`) знак вставки также соответствует началу строки (см. раздел «Флаги и свойства» на странице 93). Обратите внимание, что знак вставки при использовании в начале класса символов обозначает отрицание, а не начало строки. Дополнительные сведения см. в разделе ««Классы символов» на странице 87». |
| $ (знак доллара) | Соответствует концу строки. Вместе с установленным флагом `m` (`multiline`) знак `$` также соответствует положению перед символом (`\n`) новой строки. Дополнительные сведения см. в разделе ««Флаги и свойства» на странице 93». |
| \ (обратная косая черта) | Обозначает переход от особого значения метасимвола особых символов.<br><br>Символ обратной косой черты также применяется, если в литерале регулярного выражения нужно использовать символ прямой косой черты, как показано далее `/1\/2/` (чтобы соответствовать символу 1, за которым следует символ прямой косой черты, за которым идет символ 2). |
| . (точка) | Соответствует любому одиночному символу.<br><br>Точка соответствует символу новой строки (`\n`), только если установлен флаг `s` (`dotall`). Дополнительные сведения см. в разделе ««Флаги и свойства» на странице 93». |
| * (звезда) | Соответствует предыдущему элементу, повторяющемуся нуль или более раз.<br><br>Дополнительные сведения см. в разделе ««Квантификаторы» на странице 89». |
| + (плюс) | Соответствует предыдущему элементу, повторяющемуся один или более раз.<br><br>Дополнительные сведения см. в разделе ««Квантификаторы» на странице 89». |
| ? (знак вопроса) | Соответствует предыдущему элементу, повторяющемуся нуль или один раз.<br><br>Дополнительные сведения см. в разделе ««Квантификаторы» на странице 89». |
| ( и ) | Определяет группы внутри регулярного выражения. Группы используются со следующей целью.<br><br>• Для ограничения области оператора перестановок \| : `/(a\|b\|c)d/`<br><br>• Для определения области квантификатора: `/(walla.){1,2}/`<br><br>• Во встречных ссылках. Например, в следующем регулярном выражении `\1` соответствует всему, что соответствует первой группе образца в круглых скобках:<br><br>• `/(\w*) is repeated: \1/`<br><br>Дополнительные сведения см. в разделе ««Группы» на странице 91». |
| [ и ] | Определяет класс символов, определяющий возможные совпадения для одиночного символа:<br><br>`/[aeiou]/` соответствует любому из указанных символов.<br><br>Дефис (`-`) используется внутри классов символов для обозначения диапазона символов:<br><br>`/[A-Z0-9]/` соответствует буквам верхнего регистра от A до Z или цифрам от 0 до 9.<br><br>Внутри классов символов для перехода от символов ]<br><br>и - используется вставка обратной косой черты:<br><br>`/[+\-]\d+/` соответствует либо `+`, либо `-` перед одной или несколькими цифрами.<br><br>Внутри классов символов другие символы, которые обычно являются метасимволами, рассматриваются как обычные символы (не метасимволы), избегая необходимости использовать обратную косую черту:<br><br>`/[$]/£` соответствует либо `$`, либо `£`.<br><br>Дополнительные сведения см. в разделе ««Классы символов» на странице 87». |
| \| *(вертикальная черта)* | Используется для перестановок, чтобы соответствовать либо части справа, либо части слева:<br><br>`/abc\|xyz/` соответствует либо `abc`, либо `xyz`. |

### О метапоследовательностях

Метапоследовательности представляют собой последовательности символов, имеющих в образце регулярного выражения особое значение. В следующей таблице приведено описание этих метапоследовательностей.

| Метапоследовательность | Описание |
|---|---|
| `{n}`<br>`{n,}`<br>и<br>`{n,n}` | Определяет числовой квантификатор или диапазон квантификатора предыдущего элемента:<br>`/A{27}/` соответствует символу `A`, повторяемому `27` раз.<br>`/A{3,}/` соответствует символу `A`, повторяемому `3` раза или более.<br>`/A{3,5}/` соответствует символу `A`, повторяемому от `3` до `5` раз.<br>Дополнительные сведения см. в разделе ««Квантификаторы» на странице 89». |
| `\b` | Соответствует положению между словесным символом и несловесным символом. Также соответствует началу или концу строки, если первый или последний символ в строке является словесным символом. |
| `\B` | Соответствует положению между двумя словесными символами. Также соответствует положению между двумя несловесными символами. |
| `\d` | Соответствует десятичной цифре. |
| `\D` | Соответствует любому символу, отличному от цифры. |
| `\f` | Соответствует символу перевода страницы. |
| `\n` | Соответствует символу начала строки. |
| `\r` | Соответствует символу возврата. |
| `\s` | Соответствует любому символу разделителя (символу пробела, табуляции, новой строки или возврата). |
| `\S` | Соответствует любому символу, отличному от символа разделителя. |
| `\t` | Соответствует символу табуляции. |
| `\unnnn` | Соответствует символу юникода с кодом символа, заданным шестнадцатеричным числом *nnnn*. Например, `\u263a` является символом смайлика. |
| `\v` | Соответствует символу вертикальной подачи. |
| `\w` | Соответствует словесному символу (`AZ`–, `az`–, `0-9` или `_`). Следует отметить, что `\w` не соответствует символам, не являющимся символами английского языка, таким как `é` , `ñ` или `ç`. |
| `\W` | Соответствует любому символу, отличному от словесного символа. |
| `\\xnn` | Соответствует символу с заданным значением ASCII, определенным шестнадцатеричным числом *nn*. |

## Классы символов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Классы символов используются для установления списка символов и их соответствий определенной позиции в регулярном выражении. Классы символов определяются с помощью квадратных скобок ( `[` и `]` ). Например, следующее регулярное выражение определяет класс символов, соответствующий `bag`, `beg`, `big`, `bog` или `bug`:

```
/b[aeiou]g/
```

### Переход последовательностей в классах символов

Большинство метасимволов и метапоследовательностей, обычно имеющих в регулярном выражении особое значение, *не* сохраняют это значение внутри класса символов. Например, звездочка используется в регулярном выражении для обозначения повторения, но это не так, когда звездочка появляется в классе символов. Следующий класс символов буквально соответствует звездочке наряду с любым из перечисленных символов:

```
/[abc*123]/
```

Однако три символа, приведенные в следующей таблице, действительно функционируют как метасимволы, сохраняя в классах символов особое значение.

| Метасимвол | Значение в классах символов |
|---|---|
| `]` | Определяет конец класса символов. |
| `-` | Определяет диапазон символов (см. следующий раздел «Диапазоны символов в классах символов»). |
| `\` | Определяет метапоследовательности и отменяет особое значение метасимволов. |

Чтобы каждый из этих символов распознавался как символ литерала (без особого значения метасимвола), необходимо поставить перед этим символом символ перехода обратной косой черты. Например, следующее регулярное выражение включает класс символов, соответствующий любому из четырех символов (`$`, `\`, `]` или `-`):

```
/[$\\\]\-]/
```

Помимо метасимволов, сохраняющих свое особое значение, следующие метапоследовательности функционируют как метапоследовательности внутри классов символов:

| Метапоследовательность | Значение в классах символов |
|---|---|
| `\n` | Соответствует символу начала строки. |
| `\r` | Соответствует символу возврата. |
| `\t` | Соответствует символу табуляции. |
| `\u`***nnnn*** | Соответствует символу с заданным значением точки кода юникод (определенным шестнадцатеричным числом *nnnn*). |
| `\\x`***nn*** | Соответствует символу с заданным значением ASCII (определенным шестнадцатеричным числом *nn*). |

Другие метапоследовательности и метасимволы регулярных выражений рассматриваются внутри класса символов как обычные символы.

### Диапазоны символов в классах символов

Дефис используется для определения диапазона символов, как например `A-Z`, `a-z` или `0-9`. Эти символы должны составлять допустимый диапазон в наборе символов. Например, следующий класс символов соответствует любому из символов диапазона `a-z` или любой цифре:

```
/[a-z0-9]/
```

Для определения диапазона значения ASCII можно также использовать код символа ASCII `\\x`*nn*. Например, следующий класс символов соответствует любому символу из набора расширенных символов ASCII (таких как `é` и `ê`):

```
\\x
```

**Классы символов с отрицанием**

При использовании в начале класса символов символа (^) вставки он отрицает этот класс — любой неперечисленный символ рассматривается как совпадение. Следующий класс символов соответствует любому символу, *за исключением* буквы нижнего регистра (az–) или цифры:

```
/[^a-z0-9]/
```

Для обозначения отрицания необходимо поставить символ (^) вставки в *начале* класса символов. Иначе символ вставки просто добавляется к символам в классе символов. Например, следующий класс символов соответствует любому из числа символов, включая символ вставки:

```
/[!.,#+*%$&^]/
```

## Квантификаторы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Квантификаторы используются для определения повторений символов или последовательностей в образце, как показано далее.

| Метасимвол квантификатора | Описание |
|---|---|
| * (звезда) | Соответствует предыдущему элементу, повторяющемуся нуль или более раз. |
| + (плюс) | Соответствует предыдущему элементу, повторяющемуся один или более раз. |
| ? (знак вопроса) | Соответствует предыдущему элементу, повторяющемуся нуль или один раз. |
| {*n*}<br>{*n*,}<br>и<br>{*n*,*n*} | Определяет числовой квантификатор или диапазон квантификатора предыдущего элемента:<br>/A{27}/ соответствует символу A, повторяемому 27 раз.<br>/A{3,}/ соответствует символу A, повторяемому 3 раза или более.<br>/A{3,5}/ соответствует символу A, повторяемому от 3 до 5 раз. |

Квантификатор может применяться в отношении одиночного символа, класса символов или группы:

- /a+/ соответствует символу a, повторяемому один раз или более.
- /\d+/ соответствует одной или нескольким цифрам.
- /[abc]+/ соответствует повторению одного или более символов, каждый из которых является либоa, либо b, либо c.
- /(very, )*/ соответствует слову very, за которым следует запятая и пробел, повторяемый нуль раз или более.

Квантификаторы могут использоваться внутри групп в круглых скобках, имеющих применимые к ним квантификаторы. Например, следующий квантификатор соответствует строкам, таким как word и word-word-word:

```
/\w+(-\w+)*/
```

По умолчанию регулярные выражения представляют то, что известно как *поглощающее совпадение.* Любой подобразец в регулярном выражении (таком как .* пытается соответствовать как можно большему числу символов в строке, прежде чем переходить к следующей части регулярного выражения. Например, рассмотрим следующее регулярное выражение и строку:

```
var pattern:RegExp = /<p>.*<\/p>/;
str:String = "<p>Paragraph 1</p> <p>Paragraph 2</p>";
```

Регулярное выражение соответствует всей строке:

```
<p>Paragraph 1</p> <p>Paragraph 2</p>
```

Хотя предположим, что требуется соответствие только одной группе `<p>...</p>`. Это можно сделать следующим образом:

```
<p>Paragraph 1</p>
```

Добавьте знак вопроса (`?` после любого квантификатора, чтобы изменить его на то, что известно как *«ленивый» квантификатор*. Например, следующее регулярное выражение, использующее «ленивый» квантификатор `*?` , соответствует `<p>`, за которым следует минимально возможное число («ленивых») символов, за которыми идет `</p>`:

```
/<p>.*?<\/p>/
```

Следует держать в памяти следующие пункты, касающиеся квантификаторов.

- Квантификаторы `{0}` и `{0,0}` не исключают из совпадения элемент.
- Не объединяйте несколько квантификаторов, как в `/abc+*/`.
- Символ точки (.) не соединяет строки до тех пор, пока не установлен флаг `s` (`dotall`), даже если за ним идет квантификатор `*`. Например, рассмотрим следующий код:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*<\/p>/;
trace(str.match(re)); // null;

re = /<p>.*<\/p>/s;
trace(str.match(re));
    // output: <p>Test
    //                  Multiline</p>
```

Дополнительные сведения см. в разделе ««Флаги и свойства» на странице 93».

## Перестановки

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Символ вертикальной черты (`|`) используется в регулярных выражениях с той целью, чтобы механизм регулярного выражения учитывал при совпадении альтернативные варианты. Например, следующее регулярное выражение соответствует любому из представленных слов `cat, dog, pig, rat`:

```
var pattern:RegExp = /cat|dog|pig|rat/;
```

Можно использовать круглые скобки для определения групп с целью ограничения области оператора перестановок |. Следующее регулярное выражение соответствует слову `cat`, за которым идет `nap` или `nip`.

```
var pattern:RegExp = /cat(nap|nip)/;
```

Дополнительные сведения см. в разделе ««Группы» на странице 91».

Два следующих регулярных выражения, в одном из которых используется оператор перестановок `|`, а в другом — класс символов (определенный `[` и `]`), эквивалентны.

```
/1|3|5|7|9/
/[13579]/
```

Дополнительные сведения см. в разделе ««Классы символов» на странице 87».

## Группы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Группу в регулярном выражении можно определить с помощью круглых скобок следующим образом:

```
/class-(\d*)/
```

Группа является подразделом образца. Группы можно использовать со следующей целью.

- Для применения квантификатора к нескольким символам.
- Для отделения подобразцов с целью применения при перестановках (с помощью символа `|`).
- Для сбора совпадений подстрок (к примеру, с помощью использования `\1` в регулярных выражениях для подбора ранее сопоставленной группы или с помощью применения `$1` аналогичным образом в методе `replace()` класса String).

В следующих разделах подробно описывается использование таких групп.

### Использование групп с квалификаторами

Если группа не используется, квантификатор применяется к символу или классу символов, предшествующих ему, как это показано далее:

```
var pattern:RegExp = /ab*/ ;
// matches the character a followed by
// zero or more occurrences of the character b

pattern = /a\d+/;
// matches the character a followed by
// one or more digits

pattern = /a[123]{1,3}/;
// matches the character a followed by
// one to three occurrences of either 1, 2, or 3
```

Однако можно использовать группу для применения квантификатора к нескольким символам или классам символов:

```
var pattern:RegExp = /(ab)*/;
// matches zero or more occurrences of the character a
// followed by the character b, such as ababab

pattern = /(a\d)+/;
// matches one or more occurrences of the character a followed by
// a digit, such as a1a5a8a3

pattern = /(spam ){1,3}/;
// matches 1 to 3 occurrences of the word spam followed by a space
```

Дополнительные сведения об этом см. в разделе ««Квантификаторы» на странице 89».

### Использование групп с символом оператора перестановок (|)

Группы можно использовать для определения групп символов, к которым требуется применить символ оператора перестановок `|`), как показано ниже:

```
var pattern:RegExp = /cat|dog/;
// matches cat or dog

pattern = /ca(t|d)og/;
// matches catog or cadog
```

**Использование групп для сбора совпадений подстрок**

При определении в образце стандартной группы в круглых скобках можно позднее сослаться на нее в регулярном выражении. Такой прием известен под названием *обратная ссылка*, а такие виды групп известны как *группы сбора*. Например, в следующем регулярном выражении последовательность `\1` соответствует всему, чему соответствует подстрока группы в круглых скобках:

```
var pattern:RegExp = /(\d+)-by-\1/;
// matches the following: 48-by-48
```

В регулярном выражении можно указать до 99 таких обратных ссылок, напечатав `\1`, `\2`, ... , `\99`.

Аналогичным образом в методе `replace()` класса String можно использовать `$1$99` для вставки в замещающую цепочку собранных совпадений подстрок группы:

```
var pattern:RegExp = /Hi, (\w+)\./;
var str:String = "Hi, Bob.";
trace(str.replace(pattern, "$1, hello."));
    // output: Bob, hello.
```

Также при использовании групп сбора метод `exec()` класса RegExp и метод `match()` класса String возвращают подстроки, соответствующие группам сбора:

```
var pattern:RegExp = /(\w+)@(\w+).(\w+)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@example.com,bob,example,com
```

**Использование групп без сбора и групп просмотра вперед**

Группой без сбора является группа, используемая только для образования групп, она не «собирает» и не совпадает с пронумерованными обратными ссылками. Используйте `(?:` и `)` для определения групп без сбора, как показано далее:

```
var pattern = /(?:com|org|net);
```

Например, обратите внимание на разницу между использованием `(com|org)` в группе сбора по сравнению с использованием в группе без сбора (метод `exec()` дает перечень групп сбора после полного совпадения):

```
var pattern:RegExp = /(\w+)@(\w+).(com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
// bob@example.com,bob,example,com

//noncapturing:
var pattern:RegExp = /(\w+)@(\w+).(?:com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@example.com,bob,example
```

Особым типом группы без сбора является *группа просмотра вперед*, которая бывает двух типов: *позитивная группа просмотра вперед* и *негативная группа просмотра вперед*.

Используйте `(?=` и `)` для определения позитивной группы просмотра вперед, которая задает соответствие подобразца в группе определенному положению. Однако часть строки, соответствующая позитивной группе просмотра вперед, может соответствовать оставшимся образцам в регулярном выражении. Например, поскольку в следующем коде `(?=e)` является позитивной группой просмотра вперед, символ `e`, которому она соответствует, может совпадать с последующей частью регулярного выражения, в данном случае группой сбора `\w*)`:

```
var pattern:RegExp = /sh(?=e)(\w*)/i;
var str:String = "Shelly sells seashells by the seashore";
trace(pattern.exec(str));
// Shelly,elly
```

Используйте `(?!` и `)` для определения негативной группы просмотра вперед, которая задает *отсутствие* совпадения подобразца в группе определенному положению. Например:

```
var pattern:RegExp = /sh(?!e)(\w*)/i;
var str:String = "She sells seashells by the seashore";
trace(pattern.exec(str));
// shore,ore
```

**Использование именованных групп**

Именованная группа является типом группы в регулярном выражении, которой дан именованный идентификатор. Используйте `(?P<name>` и `)` для определения именованной группы. Например, в состав следующего регулярного выражения входит именованная группа с идентификатором под именем `digits`:

```
var pattern = /[a-z]+(?P<digits>\d+)[a-z]+/;
```

При использовании метода `exec()` добавляется совпадающая именованная группа в качестве свойства массива `result`:

```
var myPattern:RegExp = /([a-z]+)(?P<digits>\d+)[a-z]+/;
var str:String = "a123bcd";
var result:Array = myPattern.exec(str);
trace(result.digits); // 123
```

Вот другой пример, в котором используются две именованных группы с идентификаторами `name` и `dom`:

```
var emailPattern:RegExp =
    /(?P<name>(\w|[_.\-])+)@(?P<dom>((\w|-)+))+\.\w{2,4}+/;
var address:String = "bob@example.com";
var result:Array = emailPattern.exec(address);
trace(result.name); // bob
trace(result.dom); // example
```

***Примечание.*** *Именованные группы не являются частью спецификации языка ECMAScript. Они представляют собой добавленную в ActionScript 3.0 возможность.*

## Флаги и свойства

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В следующей таблице представлены пять флагов, которые можно установить для регулярных выражений. Каждый флаг можно оценить как свойство объекта регулярного выражения.

| Флаг | Свойство | Описание |
|---|---|---|
| g | global | Соответствует нескольким совпадениям. |
| i | ignoreCase | Совпадение без учета регистра. Применяется к символам A—Z и a—z, но не к расширенным символам, таким как É и é . |
| m | multiline | Когда этот флаг установлен, символы $ и ^ могут соответствовать началу и концу строки соответственно. |
| s | dotall | Когда этот флаг установлен, символ . (точка) может соответствовать символу новой строки (\n). |
| x | extended | Разрешает расширенные регулярные выражения. В состав регулярного выражения можно ввести пробелы, которые не рассматриваются как часть образца. Благодаря этому код регулярного выражения может быть более четким. |

Следует отметить, что эти свойства доступны только для чтения. Когда задается переменная регулярного выражения, установить флаги (g, i, m, s, x) можно следующим образом:

```
var re:RegExp = /abc/gimsx;
```

Однако именованные свойства нельзя задать напрямую. К примеру, следующий код привел к появлению ошибки:

```
var re:RegExp = /abc/;
re.global = true; // This generates an error.
```

По умолчанию, пока они не будут заданы в объявлении регулярного выражения, флаги не будут установлены и соответствующие свойства будут также установлены на false.

Кроме того, существуют два других свойства регулярного выражения.

- Свойство lastIndex определяет положение указателя в строке с целью использования для следующего вызова метода exec() или test() регулярного выражения.
- Свойство source определяет строку, которая, в свою очередь, определяет часть образца регулярного выражения.

### Флаг g (global)

Если флаг g (global) *не* установлен, регулярное выражение имеет не более одного совпадения. Например, если в регулярном выражении флаг g не установлен, метод String.match() возвращает только одну совпадающую подстроку:

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/;
trace(str.match(pattern)) // output: she
```

Если флаг g установлен, метод Sting.match() возвращает несколько совпадений, как показано далее:

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/g;
// The same pattern, but this time the g flag IS set.
trace(str.match(pattern)); // output: she,shells,shore
```

### Флаг i (ignoreCase)

По умолчанию совпадения регулярных выражений учитывают регистр. Если установлен флаг i (ignoreCase), регистр не учитывается. Например, в регулярных выражениях буква s нижнего регистра не соответствует букве S верхнего регистра, первому символу строки:

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/)); // output: 13 -- Not the first character
```

Однако, если установлен флаг `i`, регулярное выражение не соответствует заглавной букве `S`:

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/i)); // output: 0
```

Флаг `i` не учитывает регистр только в случае с символами `A–Z` и `a–z`, за исключением расширенных символов, таких как `É` и `é` .

### Флаг m (multiline)

Если флаг `m` (`multiline`) не установлен, символ `^` соответствует началу строки, а символ `$` — концу строки. Если флаг `m` установлен, эти символы соответствуют началу и концу строки соответственно. Рассмотрим следующую строку, в состав которой входит символ начала строки:

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/g)); // Match a word at the beginning of the string.
```

Несмотря на то что в регулярном выражении установлен флаг `g` (`global`), метод `match()` соответствует только одной подстроке, поскольку для символа `^` существует только одно совпадение, которым является начало строки. Получены следующие результаты:

```
Test
```

Вот такой же код с установленным флагом `m`:

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/gm)); // Match a word at the beginning of lines.
```

В этот раз результат включает слова в начале обеих строк:

```
Test,Multiline
```

Следует отметить, что только символ `\n` обозначает конец строки. Следующие символы не обозначают конец строки.

- Символ (`\r`) возврата
- Символ (`\u2028`) разделителя строк юникода
- Символ (`\u2029`) разделителя абзацев юникода

### Флаг s (dotall)

Если флаг `s` (`dotall` или «dot all») не установлен, точка ( `.` ) в образце регулярного выражения не соответствует символу начала строки (`\n`). Поэтому для следующего примера совпадение отсутствует:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/;
trace(str.match(re));
```

Однако, если флаг `s` установлен, точка соответствует символу начала строки:

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/s;
trace(str.match(re));
```

В этом случае совпадением является вся подстрока внутри тегов `<p>`, включая символ начала строки:

```
<p>Test
Multiline</p>
```

### Флаг x (extended)

При чтении регулярных выражений могут возникнуть трудности, особенно если в их составе много метасимволов и метапоследовательностей. Например:

```
/<p(>|(\s*[^>]*>)).*?<\/p>/gi
```

При использовании в регулярном выражении флага `x` (`extended`) все пробелы, введенные в образец, игнорируются. Например, следующее регулярное выражение идентично предыдущему примеру:

```
/     <p     (>   |   (\s*   [^>]*   >))     .*?     <\/p>   /gix
```

Если флаг `x` установлен и нет необходимости в совпадении символа пробела, поставьте перед пробелом обратную косую черту. Например, эквивалентными являются следующие два регулярных выражения:

```
/foo bar/
/foo \ bar/x
```

### Свойство lastIndex

Свойство `lastIndex` определяет позицию указателя в строке, с которой должен начинаться следующий поиск. Это свойство влияет на методы `exec()` и `test()`, вызываемые в регулярном выражении, в котором для флага `g` установлено значение `true`. Например, рассмотрим следующий код:

```
var pattern:RegExp = /p\w*/gi;
var str:String = "Pedro Piper picked a peck of pickled peppers.";
trace(pattern.lastIndex);
var result:Object = pattern.exec(str);
while (result != null)
{
    trace(pattern.lastIndex);
    result = pattern.exec(str);
}
```

По умолчанию для свойства `lastIndex` установлено значение 0 (чтобы поиск начинался в начале строки). После каждого совпадения для свойства задается положение указателя, следующее за совпадением. Поэтому результат для предшествующего кода выглядит следующим образом:

```
0
5
11
18
25
36
44
```

Если для флага `global` задано значение `false`, методы `exec()` и `test()` не используют свойство `lastIndex`.

Методы `match()`, `replace()` и `search()` класса String всегда начинают поиск с начала строки, несмотря на использование свойства `lastIndex` регулярного выражения, применяемого при вызове метода. (Однако метод `match()` не задает для свойства `lastIndex` значение 0.)

Свойство `lastIndex` можно установить так, чтобы скорректировать начальную позицию в строке для поиска совпадений регулярных выражений.

**Свойство source**

Свойство `source` определяет строку, которая, в свою очередь, определяет часть образца регулярного выражения. Например:

```
var pattern:RegExp = /foo/gi;
trace(pattern.source); // foo
```

# Методы для использования регулярных выражений со строками

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс RegExp включает два метода: `exec()` и `test()`.

Помимо методов `exec()` и `test()` класса RegExp, класс String включает следующие методы, позволяющие сопоставлять регулярные выражения в строках: `match()`, `replace()`, `search()` и `splice()`.

## Метод test()

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `test()` класса RegExp просто проверяет предоставленные строки на предмет наличия совпадений регулярного выражения, как это показано в примере ниже:

```
var pattern:RegExp = /Class-\w/;
var str = "Class-A";
trace(pattern.test(str)); // output: true
```

## Метод exec()

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `exec()` класса RegExp проверяет предоставленные строки на предмет наличия совпадения регулярного выражения и возвращает массив вместе со следующим.

- Совпадающей подстрокой
- Подстрока соответствует любой группе в круглых скобках регулярного выражения

Массив также включает свойство `index`, определяющее положение указателя начала совпадения подстроки.

Например, рассмотрим следующий код:

```
var pattern:RegExp = /\d{3}\-\d{3}-\d{4}/; //U.S phone number
var str:String = "phone: 415-555-1212";
var result:Array = pattern.exec(str);
trace(result.index, " - ", result);
// 7-415-555-1212
```

Если в регулярном выражении установлен флаг `g` (`global`), несколько раз используйте метод `exec()` для совпадения нескольких подстрок.

```
var pattern:RegExp = /\w*sh\w*/gi;
var str:String = "She sells seashells by the seashore";
var result:Array = pattern.exec(str);

while (result != null)
{
    trace(result.index, "\t", pattern.lastIndex, "\t", result);
    result = pattern.exec(str);
}
//output:
// 0  3  She
// 10  19  seashells
// 27  35  seashore
```

### Методы String, использующие параметры RegExp

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Следующие методы класса String используют регулярные выражения в качестве параметров: `match()`, `replace()`, `search()` и `split()`. Дополнительные сведения об этих методах см. в разделе ««Поиск шаблонов в строках и замена подстрок» на странице 17».

## Пример регулярных выражений: синтаксический анализатор Wiki

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Простой пример преобразования текста Wiki показывает ряд случаев применения регулярных выражений.

- Преобразование строк текста, при котором происходит совпадение исходного образца Wiki с определенными выводными строками HTML.
- Использование регулярного выражения с целью преобразования образцов URL в теги гиперссылок `<a>` HTML.
- Использование регулярного выражения с целью преобразования строк символа доллара США (таких как `"$9.95"`) в строки символа евро (такие как `"8.24 €"`).

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы с образцами применения WikiEditor можно найти в папке Samples/WikiEditor. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| WikiEditor.mxml<br>или<br>WikiEditor.fla | Основной файл приложения Flash (FLA) или Flex (MXML). |
| com/example/programmingas3/regExpExamples/WikiParser.as | Класс, включающий методы, которые используют регулярные выражения для преобразования образцов вводимого текста Wiki в эквивалентный выводной текст HTML. |
| com/example/programmingas3/regExpExamples/URLParser.as | Класс, включающий методы, которые используют регулярные выражения для преобразования строк URL в теги гиперссылок `<a>` HTML. |
| com/example/programmingas3/regExpExamples/CurrencyConverter.as | Класс, включающий методы, которые используют регулярные выражения для преобразования строк символа доллара США в строки евро. |

## Определение класса WikiParser

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс WikiParser включает методы, которые преобразуют вводимый текст Wiki в эквивалентный выводной текст HTML. Этот случай применения преобразования не является достаточно устойчивым к сбоям, но, тем не менее, он иллюстрирует некоторые хорошие примеры использования регулярных выражений для совпадения образцов и преобразования строк.

Функция конструктора вместе с методом `setWikiData()` просто инициализирует строку примера вводимого текста Wiki следующим образом:

```
public function WikiParser()
{
    wikiData = setWikiData();
}
```

Когда пользователь нажимает в демонстрационном приложении кнопку «Проверка», приложение запускает метод `parseWikiString()` объекта WikiParser. Этот метод вызывает ряд других методов, которые, в свою очередь, компонуют получившиеся на выводе строки HTML.

```
public function parseWikiString(wikiString:String):String
{
    var result:String = parseBold(wikiString);
    result = parseItalic(result);
    result = linesToParagraphs(result);
    result = parseBullets(result);
    return result;
}
```

Каждый из этих вызванных методов — `parseBold()`, `parseItalic()`, `linesToParagraphs()` и `parseBullets()` — использует метод `replace()` строки с целью замещения совпадающих образцов, определенных регулярным выражением, чтобы преобразовать вводимый текст Wiki в текст формата HTML.

### Преобразование образцов полужирного и курсивного шрифтов

Метод `parseBold()` производит поиск образца полужирного шрифта Wiki (к примеру, `'''foo'''`) и преобразует его в эквивалентный текст HTML (к примеру, `<b>foo</b>`) следующим образом:

```
private function parseBold(input:String):String
{
    var pattern:RegExp = /'''(.*?)'''/g;
    return input.replace(pattern, "<b>$1</b>");
}
```

Следует отметить, что часть `(.?*)` регулярного выражения соответствует любому числу символов (`*`) между двумя определяющими образцами `'''`. Квантификатор `?` делает совпадение непоглощающим с тем, чтобы для строки, такой как `'''aaa''' bbb '''ccc'''`, первой совпадающей строкой была `'''aaa'''`, а не вся строка (находящаяся внутри образца `'''`).

Круглые скобки в регулярном выражении определяют группу сбора, а метод `replace()` ссылается на эту группу с помощью кода `$1` в замещающей строке. Флаг `g` (`global`) в регулярном выражении обеспечивает замещение методом `replace()` всех совпадений в строке (а не только первого совпадения).

Метод `parseItalic()` функционирует практически так же, как и метод `parseBold()`, за исключением того, что он проверяет наличие двух (не трех) апострофов (`''`), выполняющих роль ограничителей курсивного текста:

```
private function parseItalic(input:String):String
{
    var pattern:RegExp = /''(.*?)''/g;
    return input.replace(pattern, "<i>$1</i>");
}
```

### Преобразование образцов маркера абзаца

Как показывает следующий пример, метод `parseBullet()` производит поиск образца строки маркера абзаца Wiki, (к примеру, `* foo`) и преобразует его в эквивалентный текст HTML (к примеру, `<li>foo</li>`):

```
private function parseBullets(input:String):String
{
    var pattern:RegExp = /^\*(.*)/gm;
    return input.replace(pattern, "<li>$1</li>");
}
```

Символ `^` в начале регулярного выражения соответствует началу строки. Флаг `m` (`multiline`) в регулярном выражении является причиной того, что регулярное выражение сопоставляет символ `^` началу линии, а не просто началу строки.

Образец `\*` соответствует символу звездочки (обратная косая черта используется для обозначения звездочки литерала вместо квантификатора `*`).

Круглые скобки в регулярном выражении определяют группу сбора, а метод `replace()` ссылается на эту группу с помощью кода `$1` в замещающей строке. Флаг `g` (`global`) в регулярном выражении обеспечивает замещение методом `replace()` всех совпадений в строке (а не только первого совпадения).

### Преобразование образцов Wiki абзаца

Метод `linesToParagraphs()` преобразует каждую линию во входной строке Wiki в тег абзаца `<p>` HTML. Линии этого метода убирают пустые линии из входной строки Wiki:

```
var pattern:RegExp = /^$/gm;
var result:String = input.replace(pattern, "");
```

Символы `^` и `$` соответствуют началу и концу строки регулярного выражения. Флаг (`multiline`) `m` в регулярном выражении является причиной того, что регулярное выражение сопоставляет символ ^ началу линии, а не просто началу строки.

Метод `replace()` замещает все совпадающие подстроки (пустые линии) пустой строкой (`""`). Флаг `g` (`global`) в регулярном выражении обеспечивает замещение методом `replace()` всех совпадений в строке (а не только первого совпадения).

## Преобразование URL-адресов в теги <a> HTML

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При нажатии в демонстрационном приложении кнопки «Проверка» и установке флажка `urlToATag` приложение вызывает статический метод `URLParser.urlToATag()` для преобразования строк URL из входной строки Wiki в теги `<a>` HTML.

```
var protocol:String = "((?:http|ftp)://)";
var urlPart:String = "([a-z0-9_-]+\.[a-z0-9_-]+)";
var optionalUrlPart:String = "(\.[a-z0-9_-]*)";
var urlPattern:RegExp = new RegExp(protocol + urlPart + optionalUrlPart, "ig");
var result:String = input.replace(urlPattern, "<a href='$1$2$3'><u>$1$2$3</u></a>");
```

Конструктор `RegExp()` используется для сборки нескольких составных частей в единое регулярное выражение (`urlPattern`). Этими составными частями являются отдельные строки, образующие часть образца регулярного выражения.

Первая часть образца регулярного выражения, определяемая строкой `protocol`, определяет URL-протокол: либо `http://` , либо `ftp://`. Круглые скобки определяют группу без сбора, обозначенную символом `?`. Это означает, что круглые скобки используются лишь для того, чтобы определять группу для образца перестановки `|`; эта группа не будет соответствовать кодам обратных ссылок (`$1`, `$2`, `$3`) в замещающей строке метода `replace()`.

Другие составные части регулярного выражения используют группы сбора (обозначенные в образце круглыми скобками), которые впоследствии используются в кодах обратных ссылок (`$1`, `$2`, `$3`) в замещающей строке метода `replace()`.

Часть образца, определяемая строкой `urlPart`, соответствует, *по крайней мере*, одному из следующих символов: `a-z`, `0-9`, `_` или `-`. Квантификатор `+` указывает на совпадение по крайней мере одного символа. `\.` обозначает символ (`.`) обязательной точки. И оставшаяся часть образца соответствует другой строке с по крайней мере одним из следующих символов: `a-z`, `0-9`, `_` или `-`.

Часть образца, определяемая строкой `optionalUrlPart`, соответствует *нуль или более* раз следующему: символу точки (`.`), за которым идет ряд буквенно-цифровых символов (в том числе `_` и `-`). Квантификатор `*` указывает на совпадение нуля или более символов.

При вызове метода `replace()` используется регулярное выражение, а с помощью обратных ссылок осуществляется сборка замещающих строк HTML.

Метод `urlToATag()` затем вызывает метод `emailToATag()`, который для замещения образцов электронной почты строками гиперссылок `<a>` HTML использует схожие механизмы. Регулярные выражения, используемые в этом демонстрационном файле для подбора HTTP, FTP, URL-адресов электронной почты, достаточно просты для приведения в качестве примера; существуют гораздо более сложные регулярные выражения для более правильного сопоставления таких URL-адресов.

## Преобразование строк символа доллара США в строки символа евро

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При нажатии в демонстрационном приложении кнопки «Проверка» и установке флажка `dollarToEuro` приложение вызывает статический метод `CurrencyConverter.usdToEuro()` для преобразования строк символа доллара США (например, `"$9.95"`) в строки символа евро (например, `"8.24 €"`) следующим образом:

```
var usdPrice:RegExp = /\$([\d,]+.\d+)+/g;
return input.replace(usdPrice, usdStrToEuroStr);
```

Первая линия определяет простой образец для подбора строк символа доллара США. Обратите внимание, что символу `$` предшествует символ перехода `\` обратной косой черты.

Метод `replace()` использует регулярное выражение в качестве обнаружителя совпадений образцов, а также вызывает функцию `usdStrToEuroStr()` для определения замещающей строки (значения в евро).

Когда имя функции используется в качестве второго параметра метода `replace()`, следующие элементы передаются в вызванную функцию в качестве параметров.

- Совпадающая часть строки.
- Любые собранные совпадения групп в круглых скобках. Количество переданных таким образом аргументов варьируется в зависимости от количества собранных совпадений групп в круглых скобках. Количество собранных совпадений групп в круглых скобках можно определить, проверив значение параметра `arguments.length - 3` в коде функции.
- Позиция в индексе в строке, где начинается совпадение.
- Строка полностью.

Метод `usdStrToEuroStr()` преобразует образцы строк символа доллара США в строки символа евро следующим образом:

```
private function usdToEuro(...args):String
{
    var usd:String = args[1];
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.828017;
    var euro:Number = Number(usd) * exchangeRate;
    trace(usd, Number(usd), euro);
    const euroSymbol:String = String.fromCharCode(8364); // €
    return euro.toFixed(2) + " " + euroSymbol;
}
```

Обратите внимание, что `args[1]` представляет собранную группу в круглых скобках, подобранную регулярным выражением `usdPrice`. Это числовая часть строки символа доллара США: сумма долларов без знака `$`. Данный метод использует преобразование курсов валют и возвращает итоговую строку (с замыкающим символом € вместо ведущего символа $).

# Глава 6. Работа с XML

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

ActionScript 3.0 включает группу классов, базирующихся на спецификации ECMAScript для XML (E4X) (ECMA-357 версии 2). Эти классы обладают набором мощных и простых в использовании функций, предназначенных для работы с XML-данными. E4X позволяет быстрее разработать код, содержащий XML-данные, чем это было возможно с помощью ранее существовавших методов программирования. Дополнительным преимуществом является то, что создаваемый код можно будет быстрее прочесть.

### Дополнительные разделы справки

класс XML

Спецификация ECMA-357

## Основы XML

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

XML является стандартным способом представления структурированной информации с тем, чтобы упростить работу за компьютером и упростить способы написания и интерпретации данных. Аббревиатура XML расшифровывается следующим образом: расширяемый язык разметки. Стандарт XML доступен по адресу www.w3.org/XML/.

XML предлагает стандартный и удобный способ классификации данных, чтобы упростить их прочтение, доступ к ним и обращение с ними. XML использует древовидную структуру и структуру тегов, схожие с HTML. Ниже приведен простой пример XML-данных:

```
<song>
    <title>What you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

XML-данные могут иметь и более сложную структуру, где теги вложены в другие теги, а также в атрибуты и другие структурные компоненты. Ниже приведен более сложный пример XML-данных:

```
<album>
    <title>Questions, unanswered</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <tracks>
        <song tracknumber="1" length="4:05">
            <title>What do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:31</lastplayed>
        </song>
        <song tracknumber="2" length="3:45">
            <title>Who do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:35</lastplayed>
        </song>
        <song tracknumber="3" length="5:14">
            <title>When do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:39</lastplayed>
        </song>
        <song tracknumber="4" length="4:19">
            <title>Do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:44</lastplayed>
        </song>
    </tracks>
</album>
```

Обратите внимание, что данный XML-документ содержит внутри себя другие полные XML-структуры (такие как теги `song` и их дочерние элементы). Здесь также представлены другие XML-структуры, такие как атрибуты (`tracknumber` и `length` в тегах `song`) и теги, содержащие другие теги вместо данных (такие как теги `tracks`).

### Начало работы с XML

Если у вас мало опыта работы с XML или он вовсе отсутствует, ниже можно ознакомиться с кратким описанием наиболее общих аспектов XML-данных. XML-данные создаются в свободной форме представления текста с определенным синтаксисом, служащим для организации информации в структурированный формат. Один набор XML-данных обычно известен как *XML-документ*. В XML-формате данные с помощью иерархической структуры организуются в *элементы* (которые могут быть одиночными элементами данных или контейнерами для других элементов). Каждый XML-документ имеет в качестве верхнего уровня или основного элемента одиночный элемент; внутри этого корневого элемента может содержаться единичная доза информации, хотя более вероятно могут быть другие элементы, которые, в свою очередь, содержат другие элементы и т.д. Например, этот XML-документ содержит информацию о музыкальном альбоме:

```
<song tracknumber="1" length="4:05">
    <title>What do you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <mood>Happy</mood>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

Каждый элемент различается по набору *тегов* — имени элемента, заключенного в угловые скобки (знаки «менее» и «более»). Открывающий тег, обозначающий начало элемента, содержит имя элемента:

```
<title>
```

Закрывающий тег, обозначающий конец элемента, содержит перед именем элемента прямую косую черту:

```
</title>
```

Если элемент не имеет содержимого, его можно создать как пустой элемент (иногда называемый самозакрывающимся элементом). В XML следующий элемент:

```
<lastplayed/>
```

идентичен такому элементу:

```
<lastplayed></lastplayed>
```

Помимо содержимого элемента, заключенного в открывающий и закрывающий теги, элемент также может содержать другие значения, известные как *атрибуты*, определенные в открывающем теге элемента. Например, этот XML-документ определяет одиночный атрибут с именем `length` и значением `"4:19"`:

```
<song length="4:19"></song>
```

Каждый XML-элемент имеет содержимое, которым является либо одиночное значение, один или несколько XML-элементов, либо ничего (в случае с пустым элементом).

### Более подробно о XML

Дополнительные сведения о работе с XML содержатся в ряде дополнительных книг и ресурсов, посвященных XML, в том числе и на этих веб-сайтах:

- Самоучитель по XML W3Schools: http://w3schools.com/xml/
- Самоучители по XML с остановками, обсуждения и другая информация: http://xmlpitstop.com/

### Классы ActionScript для работы с XML

ActionScript 3.0 включает несколько классов, используемых для работы со структурированной в XML информацией. Основными классами являются два следующих:

- XML: представляет одиночный XML-элемент, который может быть XML-документом с несколькими дочерними элементами или элементом одиночного значения внутри документа.
- XMLList: представляет набор XML-элементов. Объект XMLList используется при наличии нескольких XML-элементов одного уровня (находящихся на одном и том же уровне и принадлежащих одному и тому же родительскому элементу в иерархии XML-документа). Например, экземпляр XMLList был бы самым простым способом для работы с таким набором XML-элементов (предположительно содержащихся в XML-документе).

  ```
  <artist type="composer">Fred Wilson</artist>
  <artist type="conductor">James Schmidt</artist>
  <artist type="soloist">Susan Harriet Thurndon</artist>
  ```

Для более расширенного использования пространств имен XML в ActionScript также имеются классы Namespace и QName. Дополнительные сведения см. в разделе ««Использование пространств имен XML» на странице 119».

Помимо встроенных классов для работы с XML, в ActionScript 3.0 также имеется несколько операторов, отвечающих за особые функции доступа к XML-данным и обращения с ними. Данное направление работы с XML с привлечением названных классов и операторов известно под названием ECMAScript для XML (E4X), согласно определению спецификации ECMA-357 версии 2.

**Важные понятия и термины**

Ниже приводится список важных терминов, встречающихся в этой главе при программировании процедур обработки XML.

**Элемент** Одиночный элемент XML-документа, определенный как содержимое, заключенное в открывающий и закрывающий теги (включая сами теги). XML-элементы могут содержать текстовые данные или другие элементы либо могут быть пустыми.

**Пустой элемент** XML-элемент, который не содержит дочерних элементов. Пустые элементы часто создаются в качестве самозакрывающихся тегов (к примеру, `<element/>`).

**Документ** Одиночная XML-структура. XML-документ может содержать любое число элементов (или может состоять только из одного пустого элемента); однако XML-документ должен иметь один элемент верхнего уровня, содержащий все другие элементы документа.

**Узел** Еще одно название XML-элемента.

**Атрибут** Именованное значение, связанное с элементом, вписанным в открывающий тег элемента в формате `attributename="value"`, а не в отдельный дочерний элемент, вложенный внутрь элемента.

# Подход E4X к обработке XML

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Спецификация ECMAScript для XML определяет набор классов и функций для работы с XML-данными. Эти классы и функции известны под общим названием *E4X*. ActionScript 3.0 включает следующие классы E4X: XML, XMLList, QName и Namespace.

Методы, свойства и операторы классов E4X разработаны для следующих целей.

- Простота — где это возможно, E4X упрощает написание и понимание кода для работы с XML-данными.
- Согласованность — методы и обоснования E4X изначально согласованы между собой и с другими частями ActionScript.
- Узнаваемость — манипуляции XML-данными осуществляются с помощью хорошо известных операторов, таких как оператор «точка» ( . ).

***Примечание.*** *В ActionScript 2.0. существует другой класс XML. В ActionScript 3.0 этот класс получил новое имя XMLDocument, чтобы избежать конфликта с классом XML ActionScript 3.0, являющимся частью E4X. В ActionScript 3.0 унаследованные классы XMLDocument, XMLNode, XMLParser и XMLTag включены в пакет flash.xml прежде всего с целью поддержки. Новые классы E4X являются основными классами, и нет необходимости в импорте пакета, чтобы можно было ими пользоваться. Сведения о старых классах XML в ActionScript 2.0 см. в описании пакета flash.xml в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.*

Далее приведен пример манипуляции данными с помощью E4X:

```
var myXML:XML =
    <order>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

Приложение часто загружает XML-данные из внешнего источника, такого как веб-служба или RSS-канал. Однако для ясности в приведенных здесь примерах кода данные XML назначаются в качестве литералов.

Как показывает следующий код, E4X включает несколько интуитивно понятных операторов, таких как «точка» (`.`), и операторов идентификатора атрибута (`@`) для получения доступа к свойствам и атрибутам в XML:

```
trace(myXML.item[0].menuName); // Output: burger
trace(myXML.item.(@id==2).menuName); // Output: fries
trace(myXML.item.(menuName=="burger").price); // Output: 3.95
```

Метод `appendChild()` используется с целью назначения нового дочернего узла для XML, как это показано в следующем фрагменте:

```
var newItem:XML =
    <item id="3">
        <menuName>medium cola</menuName>
        <price>1.25</price>
    </item>

myXML.appendChild(newItem);
```

Операторы `@` и `.` используются не только для чтения данных, но и для назначения данных, как это показано в примере ниже:

```
myXML.item[0].menuName="regular burger";
myXML.item[1].menuName="small fries";
myXML.item[2].menuName="medium cola";

myXML.item.(menuName=="regular burger").@quantity = "2";
myXML.item.(menuName=="small fries").@quantity = "2";
myXML.item.(menuName=="medium cola").@quantity = "2";
```

Цикл `for` используется для осуществления повторений через узлы XML, как показано ниже:

```
var total:Number = 0;
for each (var property:XML in myXML.item)
{
    var q:int = Number(property.@quantity);
    var p:Number = Number(property.price);
    var itemTotal:Number = q * p;
    total += itemTotal;
    trace(q + " " + property.menuName + " $" + itemTotal.toFixed(2))
}
trace("Total: $", total.toFixed(2));
```

# Объекты XML

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Объект XML может представлять элемент, атрибут, комментарий, оператор обработки или текстовый элемент.

Выделяют два класса объектов XML: с *простым содержимым* и со *сложным содержимым*. Объект XML, имеющий дочерние узлы, относится к классу с простым содержимым. Объект XML имеет простое содержимое, если он является одним из следующего: атрибутом, комментарием, оператором обработки или текстовым узлом.

Например, следующий объект XML содержит сложное содержимое, в том числе комментарий и инструкцию обработки:

```
XML.ignoreComments = false;
XML.ignoreProcessingInstructions = false;
var x1:XML =
    <order>
        <!--This is a comment. -->
        <?PROC_INSTR sample ?>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

Как видно из следующего примера, теперь для создания новых объектов XML, комментария и инструкции обработки можно использовать методы `comments()` и `processingInstructions()`:

```
var x2:XML = x1.comments()[0];
var x3:XML = x1.processingInstructions()[0];
```

## Свойства XML

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс XML имеет пять статических свойств.

- Свойства `ignoreComments` и `ignoreProcessingInstructions` определяют, что игнорируется при синтаксическом анализе объекта XML — комментарии или инструкции обработки.
- Свойство `ignoreWhitespace` определяет необходимость игнорирования в тегах элемента символов пробела и внедрения выражений, разделенных только символами пробела.
- Свойства `prettyIndent` `prettyPrinting` используются для форматирования текста, возвращенного методами `toString()` и `toXMLString()` класса XML.

Дополнительные сведения об этих свойствах см. в справочнике ActionScript® 3.0 для платформы ® Flash® Platform.

## Методы XML

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Следующие методы позволяют работать с иерархической структурой объектов XML:

- `appendChild()`
- `child()`
- `childIndex()`
- `children()`
- `descendants()`
- `elements()`
- `insertChildAfter()`
- `insertChildBefore()`
- `parent()`
- `prependChild()`

Следующие методы позволяют работать с атрибутами объектов XML:

- `attribute()`
- `attributes()`

Следующие методы позволяют работать со свойствами объектов XML:

- `hasOwnProperty()`
- `propertyIsEnumerable()`
- `replace()`
- `setChildren()`

Следующие методы предназначены для работы с уточненными именами и пространствами имен:

- `addNamespace()`
- `inScopeNamespaces()`
- `localName()`
- `name()`
- `namespace()`
- `namespaceDeclarations()`
- `removeNamespace()`
- `setLocalName()`
- `setName()`
- `setNamespace()`

Следующие методы предназначены для работы с определенными типами XML-содержимого и для его определения:

- `comments()`
- `hasComplexContent()`

- `hasSimpleContent()`
- `nodeKind()`
- `processingInstructions()`
- `text()`

Следующие методы предназначены для преобразования в строки и для форматирования объектов XML:

- `defaultSettings()`
- `setSettings()`
- `settings()`
- `normalize()`
- `toString()`
- `toXMLString()`

Далее представлено несколько дополнительных методов:

- `contains()`
- `copy()`
- `valueOf()`
- `length()`

Дополнительные сведения об этих методах см. в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

# Объекты XMLList

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Экземпляр XMLList представляет произвольную коллекцию объектов XML. Он может содержать полные XML-документы, XML-фрагменты или результаты XML-запроса.

Следующие методы позволяют работать с иерархической структурой объектов XMLList:

- `child()`
- `children()`
- `descendants()`
- `elements()`
- `parent()`

Следующие методы позволяют работать с атрибутами объектов XMLList:

- `attribute()`
- `attributes()`

Следующие методы позволяют работать со свойствами объектов XMLList:

- `hasOwnProperty()`
- `propertyIsEnumerable()`

Следующие методы предназначены для работы с определенными типами XML-содержимого и для его определения:

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `processingInstructions()`
- `text()`

Следующие методы предназначены для преобразования в строки и для форматирования объектов XMLList:

- `normalize()`
- `toString()`
- `toXMLString()`

Далее представлено несколько дополнительных методов:

- `contains()`
- `copy()`
- `length()`
- `valueOf()`

Дополнительные сведения об этих методах см. в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

Для объекта XMLList, содержащего только один XML-элемент, можно использовать все свойства и методы класса XML, поскольку XMLList с одним XML-элементом рассматривается как объект, аналогичный объекту XML. Например, в следующем коде можно использовать метод `appendChild()` класса XML, поскольку `doc.div` является объектом XMLList, содержащим один элемент.

```
var doc:XML =
        <body>
            <div>
                <p>Hello</p>
            </div>
        </body>;
doc.div.appendChild(<p>World</p>);
```

Список свойств и методов XML см. в разделе «Объекты XML» на странице 108.

# Инициализация переменных XML

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Присвоить литерал XML объекту XML можно следующим образом:

```
var myXML:XML =
    <order>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

Как показано в следующем фрагменте, для создания экземпляра объекта XML из строки, содержащей XML-данные, можно также использовать конструктор `new`:

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
                            + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

Если XML-данные плохо сформированы в строке (например, если отсутствует закрывающий тег), появится ошибка периода исполнения.

Передавать данные в объект XML можно также по ссылке (из других переменных), как это показано в следующем примере:

```
var tagname:String = "item";
var attributename:String = "id";
var attributevalue:String = "5";
var content:String = "Chicken";
var x:XML = <{tagname} {attributename}={attributevalue}>{content}</{tagname}>;
trace(x.toXMLString())
    // Output: <item id="5">Chicken</item>
```

Чтобы загрузить XML-данные из URL-адреса, используйте класс URLLoader, как показано в следующем примере:

```
import flash.events.Event;
import flash.net.URLLoader;
import flash.net.URLRequest;

var externalXML:XML;
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("xmlFile.xml");
loader.load(request);
loader.addEventListener(Event.COMPLETE, onComplete);

function onComplete(event:Event):void
{
    var loader:URLLoader = event.target as URLLoader;
    if (loader != null)
    {
        externalXML = new XML(loader.data);
        trace(externalXML.toXMLString());
    }
    else
    {
        trace("loader is not a URLLoader!");
    }
}
```

Для считывания XML-данных из соединения гнезд используйте класс XMLSocket. Дополнительные сведения см. в описании класса XMLSocket в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

# Компоновка и преобразование объектов XML

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Используйте метод `prependChild()` или `appendChild()` для добавления свойства в начало или конец списка свойств объектов XML, как показано в следующем примере:

```
var x1:XML = <p>Line 1</p>
var x2:XML = <p>Line 2</p>
var x:XML = <body></body>
x = x.appendChild(x1);
x = x.appendChild(x2);
x = x.prependChild(<p>Line 0</p>);
    // x == <body><p>Line 0</p><p>Line 1</p><p>Line 2</p></body>
```

Используйте метод `insertChildBefore()` или `insertChildAfter()` для добавления свойства перед или после определенного свойства, как показано далее:

```
var x:XML =
    <body>
        <p>Paragraph 1</p>
        <p>Paragraph 2</p>
    </body>
var newNode:XML = <p>Paragraph 1.5</p>
x = x.insertChildAfter(x.p[0], newNode)
x = x.insertChildBefore(x.p[2], <p>Paragraph 1.75</p>)
```

Как видно из следующего примера, для передачи данных по ссылке (из других переменных) при создании объектов XML можно также использовать операторы фигурных скобок ({ и }):

```
var ids:Array = [121, 122, 123];
var names:Array = [["Murphy","Pat"], ["Thibaut","Jean"], ["Smith","Vijay"]]
var x:XML = new XML("<employeeList></employeeList>");

for (var i:int = 0; i < 3; i++)
{
    var newnode:XML = new XML();
    newnode =
        <employee id={ids[i]}>
            <last>{names[i][0]}</last>
            <first>{names[i][1]}</first>
        </employee>;

    x = x.appendChild(newnode)
}
```

С помощью оператора = можно присвоить объекту XML свойства и атрибуты, как показано в примере ниже:

```
var x:XML =
    <employee>
        <lastname>Smith</lastname>
    </employee>
x.firstname = "Jean";
x.@id = "239";
```

Этим для объекта XML `x` задается следующее:

```
<employee id="239">
    <lastname>Smith</lastname>
    <firstname>Jean</firstname>
</employee>
```

Для объединения объектов XMLList можно использовать операторы + и += :

```
var x1:XML = <a>test1</a>
var x2:XML = <b>test2</b>
var xList:XMLList = x1 + x2;
xList += <c>test3</c>
```

Этим для объекта XMLList `xList` задается следующее:

```
<a>test1</a>
<b>test2</b>
<c>test3</c>
```

# Пересечение XML-структур

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Одной из мощных возможностей XML является способность предоставлять через линейную строку текстовых символов сложные, вложенные данные. При загрузке данных в объект XML ActionScript проводит синтаксический анализ данных и загружает в память их иерархическую структуру (или выводит ошибку периода исполнения, если XML-данные плохо сформированы).

Операторы и методы объектов XML и XMLList упрощают задачу пересечения структуры XML-данных.

Используйте оператор точки (.) и оператор доступа к более низкому уровню (..) для получения доступа к свойствам дочернего элемента объекта XML. Рассмотрим следующий объект XML:

```
var myXML:XML =
    <order>
        <book ISBN="0942407296">
            <title>Baking Extravagant Pastries with Kumquats</title>
            <author>
                <lastName>Contino</lastName>
                <firstName>Chuck</firstName>
            </author>
            <pageCount>238</pageCount>
        </book>
        <book ISBN="0865436401">
            <title>Emu Care and Breeding</title>
            <editor>
                <lastName>Case</lastName>
                <firstName>Justin</firstName>
            </editor>
            <pageCount>115</pageCount>
        </book>
    </order>
```

Объект `myXML.book` является объектом XMLList, содержащим свойства дочернего элемента объекта `myXML` с именем `book`. Существует два объекта XML, соответствующих двум свойствам `book` объекта `myXML`.

Объект `myXML..lastName` является объектом XMLList, содержащим любое свойство более низкого уровня с именем `lastName`. Существует два объекта XML, соответствующих двум свойствам `lastName` объекта `myXML`.

Объект `myXML.book.editor.lastName` является объектом XMLList, содержащим любой дочерний элемент с именем `lastName` дочернего элемента с именем `editor` дочернего элемента с именем `book` объекта `myXML`; в данном случае объект XMLList содержит только один объект XML (свойство `lastName` со значением `"Case"`).

## Получение доступа к родительским и дочерним узлам

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `parent()` возвращает родительский элемент объекта XML.

Для получения доступа к определенным дочерним объектам можно использовать исходные значения указателей списка дочерних элементов. Например, рассмотрим объект XML `myXML`, имеющий два свойства дочернего элемента с именем `book`. Каждое свойство дочернего элемента с именем `book` имеет связанный с ним номер указателя:

```
myXML.book[0]
myXML.book[1]
```

Чтобы получить доступ к определенному внучатому элементу, можно указать номера указателей для дочернего и внучатого имен:

```
myXML.book[0].title[0]
```

Однако при наличии только одного дочернего элемента объекта `x.book[0]` с именем `title` можно пропустить ссылку на указатель следующим образом:

```
myXML.book[0].title
```

Аналогичным образом при наличии только одного дочернего элемента book объекта `x` и при условии, что этот дочерний объект имеет только один объект title, можно пропустить ссылки на оба указателя следующим образом:

```
myXML.book.title
```

Можно использовать метод `child()` для перемещения к дочерним элементам с именами на основе переменной или выражения, как это показано в следующем примере:

```
var myXML:XML =
        <order>
            <book>
                <title>Dictionary</title>
            </book>
        </order>;

var childName:String = "book";

trace(myXML.child(childName).title) // output: Dictionary
```

## Получение доступа к атрибутам

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Используйте символ `@` (оператор идентификатора атрибута) для получения доступа к атрибутам объекта XML или XMLList, как показано в следующем коде:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@id); // 6401
```

Можно использовать групповой символ `*` с символом `@` для получения доступа ко всем атрибутам объекта XML или XMLList, как показано в следующем коде:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@*.toXMLString());
// 6401
// 233
```

Можно использовать метод `attribute()` или `attributes()` для получения доступа к определенному атрибуту или ко всем атрибутам объекта XML или XMLList, как показано в следующем коде:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.attribute("id")); // 6401
trace(employee.attribute("*").toXMLString());
// 6401
// 233
trace(employee.attributes().toXMLString());
// 6401
// 233
```

Обратите внимание, что можно использовать следующий синтаксис для получения доступа к атрибутам, как это показано в следующем примере:

```
employee.attribute("id")
employee["@id"]
employee.@["id"]
```

Все элементы данного примера эквивалентны `employee.@id`. Однако синтаксис `employee.@id` является предпочитаемым подходом.

## Фильтрация по значению атрибута или элемента

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Можно использовать операторы круглых скобок — `(` и `)` — для фильтрации элементов по определенному имени элемента или значению атрибута. Рассмотрим следующий объект XML:

```
var x:XML =
    <employeeList>
        <employee id="347">
            <lastName>Zmed</lastName>
            <firstName>Sue</firstName>
            <position>Data analyst</position>
        </employee>
        <employee id="348">
            <lastName>McGee</lastName>
            <firstName>Chuck</firstName>
            <position>Jr. data analyst</position>
        </employee>
    </employeeList>
```

Все следующие выражения являются допустимыми:

- `x.employee.(lastName == "McGee")` — это второй узел `employee`.
- `x.employee.(lastName == "McGee").firstName` — это свойство `firstName` второго узла `employee`.
- `x.employee.(lastName == "McGee").@id` — это значение атрибута `id` второго узла `employee`.
- `x.employee.(@id == 347)` — это первый узел `employee`.
- `x.employee.(@id == 347).lastName` — это свойство `lastName` первого узла `employee`.
- `x.employee.(@id > 300)` — это объект XMLList с обоими свойствами `employee`.
- `x.employee.(position.toString().search("analyst") > -1)` — это объект XMLList с обоими свойствами `position`.

Если предпринимается попытка фильтрации несуществующих атрибутов или элементов, создается исключение. Например, последняя строка следующего кода генерирует ошибку, поскольку во втором элементе `p` отсутствует атрибут `id`:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(@id == '123'));
```

Аналогичным образом последняя строка следующего кода генерирует ошибку, поскольку во втором элементе `p` отсутствует свойство `b`:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(b == 'Bob'));
```

Чтобы избежать этих ошибок, можно с помощью методов `attribute()` и `elements()` обозначить свойства, имеющие совпадающие атрибуты или элементы, как это показано на примере следующего кода:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(attribute('id') == '123'));
trace(doc.p.(elements('b') == 'Bob'));
```

Можно также использовать метод `hasOwnProperty()`, как это показано в следующем коде:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(hasOwnProperty('@id') && @id == '123'));
trace(doc.p.(hasOwnProperty('b') && b == 'Bob'));
```

## Использование инструкций for..in и for each..in

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

ActionScript 3.0 включает инструкции `for..in` и `for each..in`, предназначенные для повторения в объектах XMLList. Например, рассмотрим следующий объект XML `myXML` и объект XMLList `myXML.item`. Объект XMLList `myXML.item` состоит из двух узлов `item` объекта XML.

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2' quantity='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>;
```

Инструкция цикла `for..in` позволяет осуществлять повторение в наборе имен свойств объекта XMLList:

```
var total:Number = 0;
for (var pname:String in myXML.item)
{
    total += myXML.item.@quantity[pname] * myXML.item.price[pname];
}
```

Инструкция цикла `for each..in` позволяет осуществлять повторение в свойствах объекта XMLList:

```
var total2:Number = 0;
for each (var prop:XML in myXML.item)
{
    total2 += prop.@quantity * prop.price;
}
```

# Использование пространств имен XML

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В объекте (или документе) XML пространства имен определяют тип данных, содержащихся в объекте. Например, при отправке и доставке XML-данных в веб-службу, использующую протокол передачи сообщений SOAP, в открывающем теге объекта XML указывается пространство имен:

```
var message:XML =
    <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <soap:Body xmlns:w="http://www.test.com/weather/">
            <w:getWeatherResponse>
                <w:tempurature >78</w:tempurature>
            </w:getWeatherResponse>
        </soap:Body>
    </soap:Envelope>;
```

Пространство имени имеет приставку `soap` и URI `http://schemas.xmlsoap.org/soap/envelope/`, определяющий пространство имен.

ActionScript 3.0 включает класс Namespace, предназначенный для работы с пространствами имен XML. Для объекта XML из предыдущего примера класс Namespace можно использовать следующим образом:

```
var soapNS:Namespace = message.namespace("soap");
trace(soapNS); // Output: http://schemas.xmlsoap.org/soap/envelope/

var wNS:Namespace = new Namespace("w", "http://www.test.com/weather/");
message.addNamespace(wNS);
var encodingStyle:XMLList = message.@soapNS::encodingStyle;
var body:XMLList = message.soapNS::Body;

message.soapNS::Body.wNS::GetWeatherResponse.wNS::tempurature = "78";
```

Класс XML включает следующие методы, предназначенные для работы с пространствами имен: `addNamespace()`, `inScopeNamespaces()`, `localName()`, `name()`, `namespace()`, `namespaceDeclarations()`, `removeNamespace()`, `setLocalName()`, `setName()` и `setNamespace()`.

Директива `default xml namespace` позволяет присваивать объектам XML пространства имен по умолчанию. Например, в следующем примере как `x1`, так и `x2` имеют одинаковое пространство имен по умолчанию:

```
var ns1:Namespace = new Namespace("http://www.example.com/namespaces/");
default xml namespace = ns1;
var x1:XML = <test1 />;
var x2:XML = <test2 />;
```

# Преобразование типа XML

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Объекты XML и XMLList можно преобразовать в строковые значения. Аналогичным образом строки можно преобразовать в объекты XML и XMLList. Помимо этого следует помнить о том, что все значения, имена и текстовые значения атрибутов XML являются строками. Все эти типы преобразований обсуждаются в следующих подразделах раздела «Преобразование типа XML».

## Преобразование объектов XML и XMLList в строки

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Классы XML и XMLList включают методы `toString()` и `toXMLString()`. Метод `toXMLString()` возвращает строку, включающую все теги, атрибуты, декларации пространств имен и содержимое объекта XML. Для объектов XML со сложным содержимым (дочерними элементами) метод `toString()` неидентичен методу `toXMLString()`. Для объектов XML с простым содержимым (только с одним текстовым элементом) метод `toString()` возвращает только текстовое содержимое элемента, как это показано в следующем примере:

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName.toXMLString());
    // <menuName>burger</menuName>
trace(myXML.item[0].menuName.toString());
    // burger
```

При использовании метода `trace()` без указания методов `toString()` или `toXMLString()` преобразование данных по умолчанию происходит с помощью метода `toString()`, как показано в следующем коде:

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName);
    // burger
```

При использовании метода `trace()` с целью отладки кода часто требуется использовать метод `toXMLString()`, чтобы метод `trace()` выводил более полные данные.

## Преобразование строк в объекты XML

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для создания объекта XML из строки можно использовать конструктор `new XML()` следующим образом:

```
var x:XML = new XML("<a>test</a>");
```

При попытке преобразования строки в XML из строки, представляющей недопустимый или плохо сформированный XML, будет выведена ошибка времени выполнения, как показано ниже:

```
var x:XML = new XML("<a>test"); // throws an error
```

## Преобразование значений, имен и текстовых значений атрибутов из строк

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Все значения, имена и текстовые значения атрибутов являются типами строковых данных, и может возникнуть необходимость преобразовать их в другие типы данных. Например, в следующем коде функция `Number()` используется для преобразования текстовых значений в числа:

```
var myXML:XML =
                        <order>
                            <item>
                                <price>3.95</price>
                            </item>
                            <item>
                                <price>1.00</price>
                            </item>
                        </order>;

var total:XML = <total>0</total>;
myXML.appendChild(total);

for each (var item:XML in myXML.item)
{
    myXML.total.children()[0] = Number(myXML.total.children()[0])
                                                + Number(item.price.children()[0]);
}
trace(myXML.total); // 4.95;
```

Если функция `Number()` не будет использоваться в этом коде, этот код будет интерпретировать оператор + как оператор объединения строк, а метод `trace()` в последней строке будет выводить следующее:

```
01.003.95
```

# Чтение внешних XML-документов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для загрузки XML-данных из URL-адреса можно использовать класс URLLoader. Чтобы использовать следующий код в ваших приложениях, замените значение `XML_URL` допустимым URL-адресом:

```
import flash.events.Event;
import flash.net.URLLoader;

var myXML:XML = new XML();
var XML_URL:String = "http://www.example.com/Sample3.xml";
var myXMLURL:URLRequest = new URLRequest(XML_URL);
var myLoader:URLLoader = new URLLoader(myXMLURL);
myLoader.addEventListener(Event.COMPLETE, xmlLoaded);

function xmlLoaded(event:Event):void
{
    myXML = XML(myLoader.data);
    trace("Data loaded.");
}
```

Можно также использовать класс XMLSocket для установки асинхронного подключения сокета XML к серверу. Дополнительные сведения см. в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

# Пример XML в ActionScript: загрузка данных RSS из Интернета

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Демонстрационное приложение RSSViewer обладает рядом возможностей, позволяющих работать с XML в ActionScript, включая следующее.

- Использование методов XML с целью перевода XML-данных в форму RSS-канала.
- Использование методов XML с целью компоновки XML-данных в форму HTML для последующего использования в текстовом поле.

Формат RSS широко используется для сбора новостей посредством XML. Простой файл с RSS-данными может выглядеть следующим образом:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel>
    <title>Alaska - Weather</title>
    <link>http://www.nws.noaa.gov/alerts/ak.html</link>
    <description>Alaska - Watches, Warnings and Advisories</description>

    <item>
        <title>
            Short Term Forecast - Taiya Inlet, Klondike Highway (Alaska)
        </title>
        <link>
            http://www.nws.noaa.gov/alerts/ak.html#A18.AJKNK.1900
        </link>
        <description>
            Short Term Forecast Issued At: 2005-04-11T19:00:00
            Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
            Homepage: http://pajk.arh.noaa.gov
        </description>
</item>
    <item>
        <title>
            Short Term Forecast - Haines Borough (Alaska)
        </title>
            <link>
            http://www.nws.noaa.gov/alerts/ak.html#AKZ019.AJKNOWAJK.190000
        </link>
        <description>
            Short Term Forecast Issued At: 2005-04-11T19:00:00
            Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
            Homepage: http://pajk.arh.noaa.gov
        </description>
    </item>
</channel>
</rss>
```

Приложение SimpleRSS считывает RSS-данные из Интернета, делает анализ этих данных, выделяя заголовки (названия), ссылки и описания, а затем возвращает их. Класс SimpleRSSUI обеспечивает интерфейс пользователя и вызывает класс SimpleRSS, отвечающий за все этапы обработки XML.

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы с образцами применения приложения RSSViewer можно найти в папке Samples/RSSViewer. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| RSSViewer.mxml<br>или<br>RSSViewer.fla | Основной файл приложения Flash (FLA) или Flex (MXML). |
| com/example/programmingas3/rssViewer/RSSParser.as | Класс, содержащий методы, использующие E4X для пересечения RSS (XML)-данных и создания соответствующего представления в HTML. |
| RSSData/ak.rss | Демонстрационный RSS-файл. Данное приложение настроено на считывание RSS-данных из Интернета по RSS-каналу Flex, управляемому Adobe. Однако очень просто можно изменить настройки приложения на считывание RSS-данных из этого документа, в котором применяется схема, слегка отличающаяся от схемы RSS-канала Flex. |

## Считывание и выполнение синтаксического анализа XML-данных

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс RSSParser включает метод `xmlLoaded()`, который преобразует входные RSS-данные, хранимые в переменной `rssXML`, в строку `rssOutput`, содержащую выходные данные в формате HTML.

Если исходные RSS-данные включают пространство имени по умолчанию, незадолго до начала работы метода с помощью кода задается пространство имени XML по умолчанию.

```
if (rssXML.namespace("") != undefined)
{
    default xml namespace = rssXML.namespace("");
}
```

Следующие строки впоследствии проходят через содержимое исходных XML-данных, исследуя каждое свойство более низкого уровня с именем `item`:

```
for each (var item:XML in rssXML..item)
{
    var itemTitle:String = item.title.toString();
    var itemDescription:String = item.description.toString();
    var itemLink:String = item.link.toString();
    outXML += buildItemHTML(itemTitle,
                            itemDescription,
                            itemLink);
}
```

Первые три строки просто задают переменные для представления свойств заголовков, описаний и ссылок свойства `item` XML-данных. Следующая строка впоследствии вызывает метод `buildItemHTML()` для получения HTML-данных в форме объекта XMLList с помощью трех новых строковых переменных, выступающих в роли параметров.

## Компоновка данных XMLList

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

HTML-данные (объект XMLList) имеют следующую форму:

```
<b>itemTitle</b>
<p>
    itemDescription
    <br />
    <a href="link">
        <font color="#008000">More...</font>
    </a>
</p>
```

Первые строки метода удаляют пространство имени xml по умолчанию:

```
default xml namespace = new Namespace();
```

Директива `default xml namespace` функционирует в пределах определенной области. Это означает, что областью действия данной декларации является метод `buildItemHTML()`.

Строки, идущие после компоновки XMLList, на основе строковых аргументов передаются в функцию:

```
var body:XMLList = new XMLList();
body += new XML("<b>" + itemTitle + "</b>");
var p:XML = new XML("<p>" + itemDescription + "</p>");

var link:XML = <a></a>;
link.@href = itemLink; // <link href="itemLinkString"></link>
link.font.@color = "#008000";
        // <font color="#008000"></font></a>
        // 0x008000 = green
link.font = "More...";

p.appendChild(<br/>);
p.appendChild(link);
body += p;
```

Этот объект XMLList представляет строковые данные, подходящие для текстового поля HTML ActionScript.

Метод `xmlLoaded()` использует возвращаемое значение метода `buildItemHTML()` и преобразует его в строку:

```
XML.prettyPrinting = false;
rssOutput = outXML.toXMLString();
```

## Извлечение названия RSS-канала и отправка пользовательского события

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `xmlLoaded()` задает строковую переменную `rssTitle` на основе исходных XML-данных RSS-канала.

```
rssTitle = rssXML.channel.title.toString();
```

И, наконец, метод `xmlLoaded()` генерирует событие, которое уведомляет приложение о том, что синтаксический анализ данных выполнен и данные доступны:

```
            dataWritten = new Event("dataWritten", true);
```

# Глава 7. Использование встроенных функций JSON

ActionScript 3.0 предоставляет встроенный API-интерфейс для кодирования и декодирования объектов ActionScript с помощью формата JSON (JavaScript Object Notation). Класс JSON и входящие в него вспомогательные функции разработаны в соответствии с 5-й редакцией спецификации ECMA-262, за исключением нескольких расхождений.

Участник сообщества Тодд Андерсон (Todd Anderson) подготовил сравнение собственного API-интерфейса JSON и стороннего класса as3corelib JSON. См. документ Working with Native JSON in Flash Player 11 (Работа со встроенным API-интерфейсом JSON во Flash Player 11).

**Дополнительные разделы справки**

JSON

## Обзор API-интерфейса JSON

API-интерфейс ActionScript JSON включает класс JSON и функции `toJSON()` нескольких встроенных классов. Для приложений, требующих пользовательскую кодировку JSON для какого-либо класса, платформа ActionScript обеспечивает способы переопределения кодировки по умолчанию.

Класс JSON выполняет внутреннюю обработку импорта и экспорта любого класса ActionScript, в котором отсутствует `toJSON()`. В подобных случаях JSON проверяет общедоступные свойства каждого обнаруживаемого объекта. Если объект содержит другие объекты, JSON переходит к вложенным объектам и выполняет такой же обход. Если объект предоставляет метод `toJSON()`, JSON использует этот пользовательский метод вместо внутреннего алгоритма.

Интерфейс JSON включает метод кодирования, `stringify()`, и декодирования, `parse()`. Каждый из этих методов предоставляет параметр, который позволяет вставлять собственную логику в рабочий процесс кодировки или декодировки JSON. Для метода `stringify()` это параметр `replacer`; для `parse()` — `reviver`. Эти параметры принимают определение функции с двумя аргументами, используя следующую подпись:

```
function(k, v):*
```

### Методы toJSON()

Для методов `toJSON()` используется подпись

```
public function toJSON(k:String):*
```

`JSON.stringify()` вызывает метод `toJSON()`, если он существует, для каждого общедоступного свойства, обнаруженного при прохождении объекта. Свойство содержит пару «ключ/значение». Когда `stringify()` вызывает метод `toJSON()`, он передает ключ, `k`, рассматриваемого в данный момент свойства. При типичной реализации метод `toJSON()` анализирует имя каждого свойства и возвращает желаемую кодировку его значения.

Метод `toJSON()` может возвращать значение любого типа (обозначенное как *), а не только строку. Это позволяет методу `toJSON()` возвращать объекты, если это допустимо. Например, если свойство пользовательского класса содержит объект из другой библиотеки стороннего поставщика, то когда метод `toJSON()` обнаруживает такое свойство, может возвращаться объект. В этом случае JSON переходит к объекту стороннего поставщика. Процесс кодировки проходит следующим образом.

- Если `toJSON()` возвращает объект, не являющийся строкой, `stringify()` переходит к этому объекту.
- Если `toJSON()` возвращает строку, `stringify()` упаковывает значение в другую строку, возвращает упакованную строку, а затем переходит к следующему значению.

Во многих случаях возвращение объекта предпочтительнее возвращения строки JSON, созданной приложением. При возвращении объекта используется встроенный алгоритм кодировки JSON. Это также позволяет JSON переходить к вложенных объектам.

Метод `toJSON()` не определен в классе Object, как и в большинстве других встроенных классов. Если метод отсутствует, JSON выполняет стандартный обход общедоступных свойств объекта. При желании можно также можно использовать метод `toJSON()` для отображения закрытых свойств объекта.

При работе с несколькими встроенными классами возникают трудности, которые библиотеки ActionScript не могут эффективно решить для всех случаев использования. Для таких классов в ActionScript предусмотрена простая реализация, которую пользователи могут настраивать в соответствии со своими потребностями. Ниже перечислены классы, предоставляющие тривиальные методы `toJSON()`:

- ByteArray
- Date
- Dictionary
- XML

Для класса ByteArray можно создать подкласс, чтобы переопределить его метод `toJSON()`. Также можно переопределить его прототип. Для классов Date и XML, объявленных как окончательные, требуется использовать прототип класса с целью переопределения `toJSON()`. Класс Dictionary объявляется как динамический, что предоставляет дополнительную свободу при переопределении `toJSON()`.

# Определение пользовательского поведения JSON

Чтобы реализовать собственную кодировку и декодировку JSON для встроенных классов, можно выбрать один из нескольких вариантов:

- Определение или переопределение `toJSON()` в пользовательском подклассе неокончательно встроенного класса
- Определение или переопределение `toJSON()` в прототипе класса
- Определение свойства `toJSON` в динамическом классе
- Использование параметров `JSON.stringify() replacer` и `JSON.parser() reviver`

### Дополнительные разделы справки

ECMA-262, 5-я редакция

## Определение метода toJSON() для прототипа встроенного класса

Встроенная реализация JSON в ActionScript отражает механизм ECMAScript JSON (см. 5-ю редакцию ECMA-262). Так как ECMAScript не поддерживает классы, ActionScript определяет поведение JSON с точки зрения организации на основе прототипов. Прототипы — это предшественники классов ActionScript 3.0, которые позволяют имитировать наследование, а также осуществлять добавление и переопределение элементов.

ActionScript позволяет определять или переопределять `toJSON()` для прототипа любого класса. Это преимущество применимо даже к классам, помеченным как окончательные. Когда метод `toJSON()` определяется для прототипа класса, это определение становится текущим для всех экземпляров этого класса, которые находятся в области действия приложения. Например, вот как можно определить метод `toJSON()` в прототипе класса MovieClip:

```
MovieClip.prototype.toJSON = function(k):* {
    trace("prototype.toJSON() called.");
    return "toJSON";
}
```

Когда после этого приложение вызывает метод `stringify()` для любого экземпляра MovieClip, `stringify()` возвращает вывод метода `toJSON()`:

```
var mc:MovieClip = new MovieClip();
var js:String = JSON.stringify(mc); //"prototype toJSON() called."
trace("js: " + js); //"js: toJSON"
```

Метод `toJSON()` также можно переопределить в четырех определяющих его встроенных классах. Например, следующий код переопределяет `Date.toJSON()`:

```
Date.prototype.toJSON = function (k):* {
    return "any date format you like via toJSON: "+
        "this.time:"+this.time + " this.hours:"+this.hours;
}
var dt:Date = new Date();
trace(JSON.stringify(dt));
// "any date format you like via toJSON: this.time:1317244361947 this.hours:14"
```

## Определение и переопределение toJSON() на уровне класса

Приложениям не всегда требуется использовать прототипы для переопределения `toJSON()`. Также можно определить `toJSON()` как элемент подкласса, если родительский класс не помечен как окончательный. Например, можно расширить класс ByteArray и определить общедоступную функцию `toJSON()`.

```
package {

    import flash.utils.ByteArray;
    public class MyByteArray extends ByteArray
    {
        public function MyByteArray() {
        }

        public function toJSON(s:String):*
        {
            return "MyByteArray";
        }

    }
}


var ba:ByteArray = new ByteArray();
trace(JSON.stringify(ba)); //"ByteArray"
var mba:MyByteArray = new MyByteArray(); //"MyByteArray"
trace(JSON.stringify(mba)); //"MyByteArray"
```

Если класс динамический, можно добавить свойство `toJSON` в объект этого класса и назначить ему следующую функцию:

```
var d:Dictionary = new Dictionary();
trace(JSON.stringify((d))); // "Dictionary"
d.toJSON = function(){return {c : "toJSON override."};} // overrides existing function
trace(JSON.stringify((d))); // {"c":"toJSON override."}
```

Метод `toJSON()` можно переопределить или определить для любого класса ActionScript. Однако большинство встроенных классов ActionScript не определяют метод `toJSON()`. Класс Object не определяет `toJSON` в своем прототипе по умолчанию и не объявляет его в качестве элемента класса. Только небольшое число встроенных классов определяют этот метод как функцию прототипа. Таким образом, в большинстве случаев `toJSON()` нельзя переопределить традиционными способами.

Встроенные классы, которые не определяют `toJSON()`, подвергаются сериализации в JSON посредством внутренней реализации JSON. По возможности не заменяйте эту встроенную функцию. Если определить элемент `toJSON()`, класс JSON использует пользовательскую логику вместо собственной.

## Использование параметра replacer метода JSON.stringify()

Рекомендуется переопределять `toJSON()` в прототипе и для изменения поведения класса JSON при экспорте в рамках всего приложения. Однако в некоторых случаях логика экспорта может применяться только к особым случаям в переходных условиях. Чтобы применить изменения узкого плана, можно использовать параметр `replacer` метода `JSON.stringify()`.

Метод `stringify()` применяет функцию, переданную через параметр `replacer` в кодируемый объект. Подпись для данной функции напоминает подпись `toJSON()`:

```
function (k,v):*
```

В отличие от `toJSON()`, функция `replacer` требует значение, `v`, а также ключ, `k`. Это отличие необходимо, так как метод `stringify()` определяется для статического объекта JSON, а не для кодируемого объекта. Когда метод `JSON.stringify()` вызывает `replacer(k,v)`, он проходит исходный введенный объект. Неявный параметр `this`, передаваемый функции `replacer`, указывает на объект, который содержит ключ и значение. Так как метод `JSON.stringify()` не затрагивает исходный введенный объект, он остается без изменений в контейнере, который проходится в данный момент. Таким образом можно использовать код `this[k]` с целью запроса ключа для исходного объекта. Параметр `v` содержит значение, которое преобразуется методом `toJSON()`.

Как `toJSON()`, функция `replacer` может возвращать значение любого типа. Если `replacer` возвращает строку, модуль JSON маскирует его содержимое с помощью кавычек, а затем заключает замаскированное содержимое также в кавычки. Такое заключение в кавычки гарантирует, что метод `stringify()` получит действительный объект строки JSON, который останется строкой при последующем вызове `JSON.parse()`.

В следующем коде используется параметр `replacer` и неявный параметр `this` для возврата значений `time` и `hours` объекта Date.

```
JSON.stringify(d, function (k,v):* {
    return "any date format you like via replacer: "+
        "holder[k].time:"+this[k].time + " holder[k].hours:"+this[k].hours;
});
```

## Использование параметра reviver метода JSON.parse()

Параметр `reviver` метода `JSON.parse()` выполняет функцию, противоположную функции `replacer`: он преобразует строку JSON в пригодный для использования объект ActionScript. Аргумент `reviver` представляет собой функцию, которая принимает два параметра и возвращает значение любого типа:

```
function (k,v):*
```

В этой функции `k` является ключом, а `v` reviver значением `k`. Как и `stringify()`, `parse()` обходит пары «ключ-значение» JSON и применяет к каждой паре функцию `reviver`, если она существует. Потенциальная проблема заключается в том, что класс JSON не выводит имя класса ActionScript объекта. Это создает трудности при определении типа объекта, который требуется регенерировать. Эта проблема встает особенно остро для вложенных объектов. При разработке функций `toJSON()`, `replacer` и `reviver` можно найти способы идентификации экспортируемых объектов ActionScript, при использовании которых не затрагиваются исходные объекты.

## Пример разбора

Следующий пример демонстрирует стратегию регенерации объектов, полученных в результате разбора строк JSON. В этом примере определяется два класса: JSONGenericDictExample и JSONDictionaryExtnExample. Класс JSONGenericDictExample — это пользовательский класс словаря. Каждая запись содержит имя и день рождения человека, а также уникальный идентификатор. При каждом вызове конструктора JSONGenericDictExample вновь созданный объект добавляется во внутренний статический массив с уникальным идентификатором в виде статически увеличиваемого целого числа. Класс JSONGenericDictExample также определяет метод `revive()`, который извлекает только целую часть из длинного элемента `id`. Метод `revive()` использует это целое число для поиска и возврата правильного регенерируемого объекта.

Класс JSONDictionaryExtnExample расширяет класс ActionScript Dictionary. Его записи не имеют заданной структуры и могут содержать любые данные. Вместо использования свойств, определенных для класса, данные назначаются после создания объекта JSONDictionaryExtnExample. В записях JSONDictionaryExtnExample объекты JSONGenericDictExample используются как ключи. Когда объект JSONDictionaryExtnExample регенерируется, функция `JSONGenericDictExample.revive()` использует идентификатор, связанный с JSONDictionaryExtnExample для получения правильного объекта ключа.

Самое важное то, что вместе с объектом JSONDictionaryExtnExample метод `JSONDictionaryExtnExample.toJSON()` возвращает строку маркера. Эта строка идентифицирует вывод JSON как экземпляр класса JSONDictionaryExtnExample. Этот маркер позволяет точно определить тип объекта, обрабатываемого при вызове метода `JSON.parse()`.

```
package {
    // Generic dictionary example:
    public class JSONGenericDictExample {
        static var revivableObjects = [];
        static var nextId = 10000;
        public var id;
        public var dname:String;
        public var birthday;

        public function JSONGenericDictExample(name, birthday) {
            revivableObjects[nextId] = this;
            this.id       = "id_class_JSONGenericDictExample_" + nextId;
            this.dname     = name;
            this.birthday = birthday;
            nextId++;
        }
        public function toString():String { return this.dname; }
        public static function revive(id:String):JSONGenericDictExample {
            var r:RegExp = /^id_class_JSONGenericDictExample_([0-9]*)$/;
            var res = r.exec(id);
            return JSONGenericDictExample.revivableObjects[res[1]];
        }
    }
}

package {
    import flash.utils.Dictionary;
    import flash.utils.ByteArray;

    // For this extension of dictionary, we serialize the contents of the
    // dictionary by using toJSON
    public final class JSONDictionaryExtnExample extends Dictionary {
        public function toJSON(k):* {
            var contents = {};
            for (var a in this) {
                contents[a.id] = this[a];
            }

            // We also wrap the contents in an object so that we can
            // identify it by looking for the marking property "class E"
            // while in the midst of JSON.parse.
```

```
            return {"class JSONDictionaryExtnExample": contents};
        }

        // This is just here for debugging and for illustration
        public function toString():String {
            var retval = "[JSONDictionaryExtnExample <";
            var printed_any = false;
            for (var k in this) {
                retval += k.toString() + "=" +
                "[e="+this[k].earnings +
                ",v="+this[k].violations + "], "
                printed_any = true;
            }
            if (printed_any)
                retval = retval.substring(0, retval.length-2);
            retval += ">]"
            return retval;
        }
    }
}
```

Следующий сценарий среды выполнения вызывает `JSON.parse()` для объекта JSONDictionaryExtnExample, функция `reviver` вызывает `JSONGenericDictExample.revive()` для каждого объекта в JSONDictionaryExtnExample. Этот вызов извлекает идентификатор, представляющий ключ объекта. Функция `JSONGenericDictExample.revive()` использует этот идентификатор для извлечения и возврата сохраненного объекта JSONDictionaryExtnExample из статического массива с ограниченным доступом.

```
import flash.display.MovieClip;
import flash.text.TextField;

var a_bob1:JSONGenericDictExample = new JSONGenericDictExample("Bob", new
Date(Date.parse("01/02/1934")));
var a_bob2:JSONGenericDictExample = new JSONGenericDictExample("Bob", new
Date(Date.parse("05/06/1978")));
var a_jen:JSONGenericDictExample = new JSONGenericDictExample("Jen", new
Date(Date.parse("09/09/1999")));

var e = new JSONDictionaryExtnExample();
e[a_bob1] = {earnings: 40, violations: 2};
e[a_bob2] = {earnings: 10, violations: 1};
e[a_jen]  = {earnings: 25, violations: 3};

trace("JSON.stringify(e): " + JSON.stringify(e)); // {"class JSONDictionaryExtnExample":
                        //{"id_class_JSONGenericDictExample_10001":
                        //{"earnings":10,"violations":1},
                        //"id_class_JSONGenericDictExample_10002":
                        //{"earnings":25,"violations":3},
                        //"id_class_JSONGenericDictExample_10000":
                        // {"earnings":40,"violations":2}}}

var e_result = JSON.stringify(e);

var e1 = new JSONDictionaryExtnExample();
var e2 = new JSONDictionaryExtnExample();

// It's somewhat easy to convert the string from JSON.stringify(e) back
// into a dictionary (turn it into an object via JSON.parse, then loop
```

```
// over that object's properties to construct a fresh dictionary).
//
// The harder exercise is to handle situations where the dictionaries
// themselves are nested in the object passed to JSON.stringify and
// thus does not occur at the topmost level of the resulting string.
//
// (For example: consider roundtripping something like
//   var tricky_array = [e1, [[4, e2, 6]], {table:e3}]
// where e1, e2, e3 are all dictionaries.  Furthermore, consider
// dictionaries that contain references to dictionaries.)
//
// This parsing (or at least some instances of it) can be done via
// JSON.parse, but it's not necessarily trivial.  Careful consideration
// of how toJSON, replacer, and reviver can work together is
// necessary.

var e_roundtrip =
    JSON.parse(e_result,
            // This is a reviver that is focused on rebuilding JSONDictionaryExtnExample objects.
                function (k, v) {
                     if ("class JSONDictionaryExtnExample" in v) { // special marker tag;
                         //see JSONDictionaryExtnExample.toJSON().
                        var e = new JSONDictionaryExtnExample();
                        var contents = v["class JSONDictionaryExtnExample"];
                        for (var i in contents) {
                           // Reviving JSONGenericDictExample objects from string
                           // identifiers is also special;
                           // see JSONGenericDictExample constructor and
                           // JSONGenericDictExample's revive() method.
                            e[JSONGenericDictExample.revive(i)] = contents[i];
                        }
                        return e;
                    } else {
                        return v;
                    }
                });

trace("// == Here is an extended Dictionary that has been round-tripped  ==");
trace("// == Note that we have revived Jen/Jan during the roundtrip.    ==");
trace("e:           " + e); //[JSONDictionaryExtnExample <Bob=[e=40,v=2], Bob=[e=10,v=1],
                          //Jen=[e=25,v=3]>]
trace("e_roundtrip: " + e_roundtrip); //[JSONDictionaryExtnExample <Bob=[e=40,v=2],
                                      //Bob=[e=10,v=1], Jen=[e=25,v=3]>]
trace("Is e_roundtrip a JSONDictionaryExtnExample? " + (e_roundtrip is
JSONDictionaryExtnExample)); //true
trace("Name change: Jen is now Jan");
a_jen.dname = "Jan"

trace("e:           " + e); //[JSONDictionaryExtnExample <Bob=[e=40,v=2], Bob=[e=10,v=1],
                          //Jan=[e=25,v=3]>]
trace("e_roundtrip: " + e_roundtrip); //[JSONDictionaryExtnExample <Bob=[e=40,v=2],
                                      //Bob=[e=10,v=1], Jan=[e=25,v=3]>]
```

# Глава 8. Обработка событий

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Система обработки событий облегчает для программистов процедуру ответа на действия пользователя и события системы. Модель событий ActionScript 3.0 не только удобна, но и соответствует принятым стандартам, а также хорошо интегрируется со списком отображения. Будучи созданной на основе спецификации событий из объектной модели документа (DOM) третьего уровня (стандартной для отрасли архитектуры обработки событий), новая модель событий представляет собой мощный и вместе с тем интуитивно понятный инструмент обработки событий для программистов ActionScript.

Система обработки событий ActionScript 3.0 тесно взаимодействует со списком отображения. Чтобы лучше разобраться в основах применения списка отображения, прочтите раздел «Программирование отображаемого содержимого» на странице 162.

### Дополнительные разделы справки

Пакет flash.events

Объектная модель документа (DOM), уровень 3, спецификация событий

## Основы обработки событий

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

События можно представить в качестве любых явлений в SWF-файле, которые могут заинтересовать вас как программиста. Например, большинство SWF-файлов поддерживает тот или иной способ взаимодействия с пользователем (от простейших вещей вроде реакции на нажатие кнопки мыши до более сложных операций, таких как принятие и обработка данных, введенных в форму). Любое подобное взаимодействие пользователя с SWF-файлом рассматривается в качестве события. События могут происходить без прямого взаимодействия с пользователем (например, завершение загрузки данных с сервера или активация подключенной камеры).

В ActionScript 3.0 каждое событие представлено объектом события, который является экземпляром класса Event или одного из его подклассов. Объект события не только хранит информацию о том или ином событии, но и содержит методы, облегчающие управление объектом события. Например, когда программа Flash Player или AIR регистрирует нажатие кнопки мыши, она создает объект события (экземпляр класса MouseEvent), который представляет определенное событие щелчка мыши.

После создания объекта события Flash Player или AIR *отправляет* его. Это значит, что объект события передается объекту, который является целью события. Объект, выступающий в качестве назначения для отправленного объекта события, называется *целью события*. Например, когда становится активна прикрепленная камера, Flash Player отправляет объект события напрямую в цель события, которая в данном случае является объектом, представляющим камеру. Однако если цель события находится в списке отображения, объект события передается вниз по иерархии списка отображения до тех пор, пока не достигнет цели события. В некоторых случаях объект события затем «всплывает» обратно в иерархии списка отображения по тому же пути. Этот обход иерархии списка отображения называется *потоком событий*.

«Прослушивать» объекты событий в своем коде можно с помощью прослушивателей событий. *Прослушиватели событий* — это функции или методы, которые записываются для реакции на определенные события. Чтобы обеспечить ответ программы на события, необходимо добавить прослушиватели событий к цели события либо к любому объекту списка отображения, который принадлежит потоку событий объекта события.

Каждый раз при написании кода прослушивателя событий он следует этой основной структуре (элементы, выделенные полужирным шрифтом — это заместители, которые заполняются, исходя из требований вашего конкретного случая):

```
function eventResponse(eventObject:EventType):void
{
    // Actions performed in response to the event go here.
}

eventTarget.addEventListener(EventType.EVENT_NAME, eventResponse);
```

Такой код выполняет две задачи. Во-первых, он определяет функцию. Таким образом указываются действия, которые будут выполняться в ответ на это событие. Во-вторых, он вызывает метод `addEventListener()` исходного объекта, по существу «подписывая» функцию на указанное событие. Таким образом, когда это событие происходит, выполняются действия функции. При фактическом появлении события цель события проверяет свой список, который состоит из всех функций и методов, зарегистрированных в качестве прослушивателей событий. Затем он по очереди вызывает их, передавая в качестве параметра объект.

Чтобы создать собственный прослушиватель событий, необходимо изменить в этом коде 4 вещи. Сначала необходимо заменить имя функции именем, которое вы хотите использовать (это необходимо сделать дважды там, где в коде присутствует надпись **eventResponse**). Во-вторых, необходимо указать соответствующее имя класса объекта события, отправленного событием, прослушивание которого необходимо выполнить (**EventType** в данном коде). Кроме того, нужно указать подходящую константу для определенного события (**EVENT_NAME** в примере кода). В-третьих, следует вызвать метод `addEventListener()`, применив его к объекту, который отправит событие (**eventTarget** в этом коде). По желанию можно изменить имя переменной, использованной в качестве параметра функции (**eventObject** в данном коде).

### Важные понятия и термины

Ниже приводится список важных терминов, встречающихся при написании программ обработки событий:

**Восходящая маршрутизация** Восходящая маршрутизация имеет место при определенных событиях, причем родительский экранный объект может отвечать на события, отправляемые его дочерним элементом.

**Фаза восходящей маршрутизации** Часть потока событий, в которой событие передается вверх к родительским экранным объектам. Фаза восходящей маршрутизации наступает после фаз захвата и цели.

**Фаза захвата** Часть потока событий, в которой событие передается вниз от наиболее общего к наиболее определенному объекту цели. Фаза захвата протекает до фаз цели и восходящей маршрутизации.

**Поведение по умолчанию** Некоторые события включают поведение, которое обычно сопровождает появление события и называется поведением по умолчанию. Например, когда пользователь набирает текст в текстовом поле, возникает явление события ввода текста. Поведение по умолчанию для этого события состоит в том, чтобы фактически показать символ, который был введен в текстовое поле. Но это поведение можно заменить другим (если по какой-либо причине не требуется показывать вводимый символ).

**Отправка** Действие, уведомляющее прослушиватели событий о появлении события.

**Событие** То, что происходит с объектом и о чем он может сообщить другим объектам.

**Поток событий** Когда с объектом в списке отображения (показанном на экране) происходят события, все объекты, которые содержат этот объект, уведомляются о событии и уведомляют в свою очередь собственные прослушиватели событий. Процесс начинается в рабочей области, продолжается при работе со списком отображения, переходит к фактическому объекту, в котором произошло событие, а затем снова возвращается к рабочей области. Этот процесс называется потоком событий.

**Объект Event** Объект, содержащий информацию об определенном вхождении события, которая отправляется всем прослушивателям при отправке события.

**Цель события** Объект, который фактически отправляет событие. Например, если пользователь нажимает кнопку внутри объекта Sprite, который вложен в свою очередь в рабочую область, все эти объекты отправляют события, но целью события является то, где фактически произошло событие (в данном случае это нажатая кнопка).

**Прослушиватель** Объект или функция, зарегистрировавшая себя с объектом и указывающая, что его необходимо уведомить при появлении определенного события.

**Фаза цели** Точка потока событий, в которой событие достигло максимально определенной цели. Фаза цели находится между фазами захвата и восходящей маршрутизации.

# Чем обработка событий в ActionScript 3.0 отличается от предыдущих версий

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Наиболее заметное отличие обработки событий в ActionScript 3.0 от обработки событий в предыдущих версиях этого языка состоит в том, что в ActionScript 3.0 существует только одна система обработки событий, тогда как в предыдущих версиях ActionScript имеется несколько разных систем. Этот раздел начинается с обзора обработки событий в предыдущих версиях ActionScript, вслед за которым идет обсуждение изменений обработки событий в ActionScript 3.0.

## Обработка событий в предыдущих версиях ActionScript

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В версиях ActionScript, вышедших перед ActionScript 3.0, имеется ряд различных способов обработки событий:

- обработчики событий `on()`, которые можно разместить непосредственно в экземплярах Button и MovieClip;
- обработчики `onClipEvent()`, которые можно разместить непосредственно в экземплярах MovieClip;
- свойства функции обратного вызова (например, `XML.onload` и `Camera.onActivity`);
- прослушиватели событий, зарегистрированные с помощью метода `addListener()`;
- класс UIEventDispatcher, в котором частично реализована модель событий DOM.

Каждый из этих механизмов обладает собственным набором преимуществ и ограничений. Обработчики `on()` и `onClipEvent()` просты в применении, но усложняют последующую работу с проектами, поскольку код, который размещен непосредственно на кнопках и фрагментах роликов, бывает сложно найти. Функции обратного вызова легко реализуются, но для каждого события можно использовать только одну функцию обратного вызова. Прослушиватели событий реализуются сложнее: они требуют не только создания объекта прослушивателя и функции, но также и регистрации прослушивателя вместе с объектом, который генерирует событие. Однако это увеличение объема служебной информации позволяет создать несколько объектов прослушивателя и зарегистрировать их все для одного и того же события.

Разработка компонентов для ActionScript 2.0 привела к появлению еще одной модели событий. Эта новая модель, включенная в класс UIEventDispatcher, основана на подмножестве спецификации событий DOM. Разработчики, знакомые с обработкой компонентов событий, относительно безболезненно смогут перейти к новой модели событий в ActionScript 3.0.

К сожалению, в различных моделях событий используется синтаксис, который бывает то идентичным, то разным. Например, в ActionScript 2.0 некоторые свойства (например, `TextField.onChanged`) могут применяться как функция обратного вызова либо как прослушиватель событий. Однако синтаксис для регистрации объектов прослушивателя отличается в зависимости от того, используется ли один из шести классов, поддерживающих прослушиватели, или класс UIEventDispatcher. Для классов Key, Mouse, MovieClipLoader, Selection, Stage и TextField применяется метод `addListener()`, а для обработки событий компонентов — метод `addEventListener()`.

Еще одна сложность, связанная с различными моделями обработки событий, заключалась в том, что область действия функции обработчика событий в значительной степени варьируется в зависимости от использованного механизма. Иными словами, значение ключевого слова `this` не было идентичным в системах обработки событий.

## Обработка событий в ActionScript 3.0

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

ActionScript 3.0 располагает единой моделью обработки событий, которая заменяет многие различные механизмы обработки событий, которые существовали в предыдущих версиях языка. Новая модель событий основана на спецификации событий из объектной модели документа (DOM) третьего уровня. Хотя формат SWF-файла не ориентирован специально на стандарт объектной модели документа, существует достаточное сходство между списком отображения и структурой DOM, обеспечивающее возможность реализации модели событий DOM. Объект в списке отображения аналогичен узлу в иерархической структуре DOM, а термины *объект списка отображения* и *узел* используются в качестве аналогов в рамках данного обсуждения.

Модель событий DOM, реализованная в программах Flash Player и AIR, включает понятие «вид поведения по умолчанию». *Поведение по умолчанию* — это действие, которое выполняется программами Flash Player или AIR в качестве нормальной последовательности определенных событий.

### Варианты поведения по умолчанию

Обычно код, который отвечает на события, пишется разработчиком. Однако в ряде случаев поведение настолько часто ассоциируется с событием, что Flash Player или AIR автоматически выполняет его, если разработчик не добавляет код отмены. Поскольку Flash Player или AIR автоматически воспроизводит это поведение, подобные виды поведения называются видами поведения по умолчанию.

Например, когда пользователь вводит текст в объект TextField, настолько часто предполагается, что он будет показан в объекте TextField, что это поведение встраивается в Flash Player и AIR. Если вы не хотите, чтобы это поведение выполнялось, его можно отменить с помощью новой системы обработки событий. Когда пользователь вводит текст в объект TextField, Flash Player или AIR создают экземпляр класса TextEvent, отражающий введенные данные. Чтобы запретить этим программам показ текста в объекте TextField, необходимо открыть соответствующий экземпляр TextEvent и вызвать его метод `preventDefault()`.

Существуют поведения по умолчанию, которые нельзя отменить. Например, Flash Player и AIR генерируют объект MouseEvent, когда пользователь дважды щелкает слово в объекте TextField. Поведение по умолчанию, которое невозможно предотвратить, состоит из подсвечивания слова под курсором.

Многие типы объектов событий не имеют собственного поведения по умолчанию. Например, Flash Player отправляет объект события подключения при установке сетевого подключения, но с ним не связано поведение по умолчанию. В документации по API-интерфейсу класса Event и его подклассов перечисляются все события и связанное с каждым из них поведение по умолчанию, а также указывается, можно ли его отменить.

Важно понять, что виды поведения по умолчанию ассоциируются только с объектами событий, отправленными Flash Player или AIR, и не существуют для объектов событий, которые отправлены программным образом с помощью ActionScript. Например, можно воспользоваться методами класса EventDispatcher для отправки объекта события с типом `textInput`, но с этим объектом события не будет связано поведение по умолчанию. Иными словами, Flash Player и AIR не будут показывать символ в объекте TextField в результате события `textInput`, которое отправлено программным методом.

**Новые функции прослушивателей событий в ActionScript 3.0**

Для разработчиков, имеющих опыт использования метода `addListener()` в ActionScript 2.0, может оказаться полезным указать на различия между моделью прослушивателя событий ActionScript 2.0 и моделью событий ActionScript 3.0. В следующем списке описан ряд основных различий между двумя моделями событий:

- Чтобы добавить прослушиватели событий в ActionScript 2.0, необходимо воспользоваться методом `addListener()` в одних случаях и методом `addEventListener()` в других. А в ActionScript 3.0 всегда применяется метод `addEventListener()`.
- В ActionScript 2.0 нет потока событий. Это означает, что метод `addListener()` можно вызвать только по отношению к объекту, который выполняет широковещание события. В ActionScript 3.0 метод `addEventListener()` можно вызвать по отношению к любому объекту, включенному в поток событий.
- В ActionScript 2.0 в качестве прослушивателей событий могут выступать функции, методы или объекты, тогда как в ActionScript 3.0 прослушивателями событий могут быть только функции или методы.

# Поток событий

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

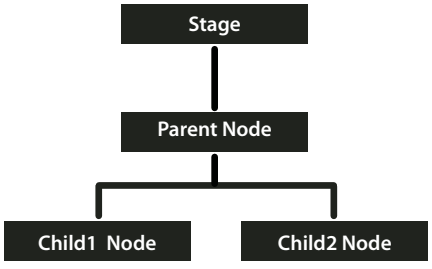
Flash Player или AIR отправляет объекты событий при каждом появлении события. Если цель события не внесена в список отображения, Flash Player или AIR отправляет объект события напрямую к цели события. Например, Flash Player отправляет объект события хода выполнения напрямую в объект URLStream. Однако если цель события внесена в список отображения, Flash Player отправляет объект события в список отображения, и этот объект проходит по списку отображения к цели события.

*Поток событий* описывает перемещение объекта события в списке отображения. Список отображения организован как иерархия, которую можно описать в виде дерева. В верхней части иерархии списка отображения находится рабочая область, которая является особым контейнером экранного объекта, выступающим в роли корня списка отображения. Рабочая область представлена классом flash.display.Stage. Она вызывается только с помощью экранного объекта. Каждый экранный объект имеет свойство `stage`, которое относится к рабочей области приложения.

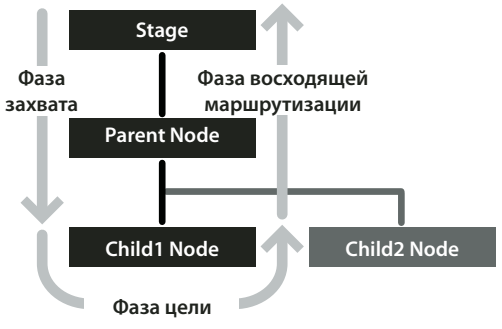
Когда Flash Player или AIR отправляет объект события для события, относящегося к списку отображения, этот объект события появляется в рабочей области и возвращается к *узлу цели*. Спецификация событий DOM определяет узел цели как узел, представляющий цель события. Иными словами, целью события является объект списка отображения, в котором произошло событие. Например, если пользователь щелкнет объект списка отображения с именем `child1`, Flash Player или AIR отправит объект события, указав в качестве узла цели `child1`.

Поток событий на понятийном уровне делится на три части. Первая фаза называется фазой захвата; эта фаза включает в себя все узлы от рабочей области до родительского узла узла цели. Вторая часть называется фазой цели, которая состоит исключительно из целевого узла. Третья часть называется фазой восходящей маршрутизации. Фаза восходящей маршрутизации включает в себя узлы, встреченные на обратном пути от родительского узла узла цели к рабочей области.

Смысл фаз проясняется, если представить список отображения в виде вертикальной иерархии, где рабочая область находится сверху, как показано на следующей диаграмме:




Если пользователь нажимает `Child1 Node`, Flash Player или AIR отправляет объект события в поток событий. Как указано на следующем изображении, объект начинает движение с рабочей области `Stage`, перемещается вниз к родительскому узлу `Parent Node`, затем к дочернему узлу `Child1` и, наконец, перемещается вверх обратно в рабочую область `Stage`, снова проходя через родительский узел `Parent Node` на пути к рабочей области `Stage`.



В данном примере фаза захвата охватывает рабочую область `Stage` и родительский узел `Parent Node` во время первоначального нисходящего движения. Фаза цели состоит из времени, затраченного на дочерний узел `Child1`. Фаза восходящей маршрутизации состоит из родительского узла `Parent Node` и рабочей области `Stage`, поскольку они встречаются на обратном пути вверх к корневому узлу.

Этот поток событий относится к более мощной системе обработки событий, чем та, которая была раньше доступна программистам ActionScript. В предыдущих версиях ActionScript поток событий не существует. Это означает, что прослушиватели события можно добавить только к объекту, который генерирует его. В ActionScript 3.0 прослушиватели событий можно добавить не только к узлу цели, но и к любому узлу в потоке событий.

Возможность добавлять прослушиватели событий в поток событий полезна, когда компонент пользовательского интерфейса состоит более чем из одного объекта. Например, объект кнопки часто содержит текстовый объект, который служит меткой кнопки. Не имея возможности добавить прослушиватель к потоку событий, вам бы пришлось добавить прослушиватель как к объекту кнопки, так и к текстовому объекту, чтобы обеспечить получение уведомления о событиях, связанных с щелчком в любом месте кнопки. Однако наличие потока событий позволяет разместить один прослушиватель события на объекте кнопки, обрабатывающем события щелчков, которые происходят с текстовым объектом, либо в областях объекта кнопки, не закрытых текстовым объектом.

Однако не всякий объект события участвует во всех трех фазах потока событий. Некоторые типы событий (например, типы событий `enterFrame` и `init`) отправляются напрямую в узел цели и не участвуют ни в фазе захвата, ни в фазе восходящей маршрутизации. Другие события могут иметь в качестве цели объекты, которые не входят в список отображения (например, события, отправленные экземпляру класса Socket). Эти объекты событий будут также напрямую поступать в объект цели, не участвуя в фазах захвата и восходящей маршрутизации.

Чтобы выяснить, как ведет себя тот или иной тип события, можно либо проверить документацию API, либо изучить свойства объекта события. Изучение свойств объекта события описано в следующем разделе.

# Объекты Event

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Объекты событий служат двум основным целям в новой системе обработки событий. Во-первых, объекты событий представляют действительные события, сохраняя информацию о тех или иных событиях в наборе свойств. Во-вторых, объекты событий содержат ряд методов, позволяющих управлять ими и изменять поведение системы обработки событий.

Чтобы упростить доступ к этим свойствам и методам, Flash Player API определяет класс Event, который выступает в качестве базового класса для всех объектов событий. Класс Event определяет фундаментальный набор свойств и методов, общих для всех объектов событий.

Этот раздел начинается с описания свойств класса Event, вслед за которым следует описание методов класса Event, и завершается сведениями о том, почему существуют подклассы класса Event.

## Описание свойств класса Event

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс Event определяет ряд свойств и констант только для чтения, которые содержат важные сведения об объекте события. Наиболее важными являются следующие:

- Типы объекта Event представлены константами и сохраняются в свойстве `Event.type`.
- Если поведение события по умолчанию можно предотвратить, это выражается логическим значением, которое сохраняется в свойстве `Event.cancelable`.
- Информация о потоке событий содержится в сохраняющихся свойствах.

### Типы объектов Event

С каждым объектом события связан определенный тип события. Типы событий хранятся в свойстве `Event.type` в виде строк. Знать тип события нужно для того, чтобы написанный код отличал объекты разного типа друг от друга. Например, в следующем коде указано, что функция прослушивателя `clickHandler()` должна реагировать на любые объекты событий щелчка мышью, которые передаются `myDisplayObject`:

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

С самим классом Event связано около 24 типов событий, которые представлены константами класса Event. Некоторые из них показаны в следующем фрагменте определения класса Event:

```
package flash.events
{
    public class Event
    {
        // class constants
        public static const ACTIVATE:String = "activate";
        public static const ADDED:String= "added";
        // remaining constants omitted for brevity
    }
}
```

Эти константы обеспечивают простой способ ссылки на определенные типы событий. Необходимо использовать эти константы вместо строк, которые они представляют. Если вы ошибетесь в написании имени константы, приведенном в коде, компилятор перехватит ошибку, но если вместо него будут выбраны строки, во время компиляции опечатка может не появиться. Это может привести к неожиданному поведению, которое будет сложно отладить. Например, при добавлении прослушивателя событий воспользуйтесь следующим кодом:

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

а не

```
myDisplayObject.addEventListener("click", clickHandler);
```

### Информация о поведении по умолчанию

Код способен проверить, могло ли поведение по умолчанию для того или иного объекта события быть отменено через свойство `cancelable`. Свойство `cancelable` содержит логическое значение, которое указывает, можно или нет предотвратить поведение по умолчанию. Поведение по умолчанию, связанное с несколькими событиями сразу, можно отменить с помощью метода `preventDefault()`. Дополнительные сведения см. в пункте «Отмена поведения события по умолчанию» раздела «Описание методов класса Event» на странице 143.

### Информация о потоке событий

Оставшиеся свойства класса Event содержат важную информацию об объекте события и его отношении к потоку событий, как указано в следующем списке:

- Свойство `bubbles` содержит сведения о стадиях потока событий, в которых принимает участие объект события.
- Свойство `eventPhase` указывает на текущую фазу в потоке событий.
- Свойство `target` сохраняет ссылку на цель события.
- Свойство `currentTarget` ссылается на объект списка отображения, который в данный момент обрабатывает объект события.

### Свойство bubbles

Говорят, что событие перемещается вверх, если его объект события участвует в фазе восходящей маршрутизации потока событий. Это означает, что объект события передается из узла цели обратно через родительские элементы до тех пор, пока не достигнет рабочей области. Свойство `Event.bubbles` содержит сведения о логическом значении, указывающем на участие объекта событий в фазе восходящей маршрутизации. Поскольку все движущиеся вверх события участвуют также в фазах захвата и цели, любое движущееся вверх событие принимает участие во всех трех фазах потока событий. Если выбрано значение `true`, объект события участвует во всех трех фазах. Если присвоено значение `false`, объект события не участвует в фазе восходящей маршрутизации.

### Свойство eventPhase

Фазу события для любого объекта события можно определить путем изучения его свойства `eventPhase`. Свойство `eventPhase` содержит целое число без знака, представляющее одну из трех фаз потока событий. Flash Player API определяет отдельный класс EventPhase, который содержит три константы, соответствующие трем целым значениям без знака, как указано в следующем фрагменте кода:

```
package flash.events
{
    public final class EventPhase
    {
        public static const CAPTURING_PHASE:uint = 1;
        public static const AT_TARGET:uint = 2;
        public static const BUBBLING_PHASE:uint= 3;
    }
}
```

Эти константы соответствуют трем действительным значениям свойства `eventPhase`. Эти константы можно использовать, чтобы сделать ваш код более читабельным. Например, если необходимо убедиться в том, что функция с именем `myFunc()` вызывается только тогда, когда цель события находится в его целевой рабочей области, можно воспользоваться следующим кодом для проверки данного условия:

```
if (event.eventPhase == EventPhase.AT_TARGET)
{
    myFunc();
}
```

### Целевое свойство

Свойство `target` имеет ссылку на объект, являющийся целью события. В ряде случаев она дается напрямую (например, при активации микрофона целью объекта события становится объект Microphone). Однако если цель находится в списке отображения, необходимо учесть иерархию списка отображения. Например, если пользователь вводит данные о щелчке мыши в тот момент, когда объекты списка отображения перекрываются, Flash Player и AIR всегда выбирают в качестве цели события объект, который находится дальше всего от рабочей области.

В случае сложных SWF-файлов, особенно когда кнопки в них, как это бывает, украшены меньшими по размеру дочерними объектами, свойство `target` нельзя использовать часто, поскольку оно будет время от времени указывать на дочерний объект кнопки вместо самой кнопки. В этих ситуациях чаще всего к кнопке добавляются прослушиватели событий и используется свойство `currentTarget`, поскольку оно указывает на кнопку, тогда как свойство `target` может указывать на дочерний объект кнопки.

### Свойство currentTarget

Свойство `currentTarget` содержит ссылку на объект, который в данный момент обрабатывает объект события. Хотя может показаться странным не знать, какой узел в данный момент обрабатывает изучаемый объект события, учтите, что вы можете добавить функцию прослушивателя к любому экранному объекту в потоке событий для данного объекта события, и эту функцию можно разместить в любом месте. Более того, одну и ту же функцию прослушивателя можно добавить к разным экранным объектам. По мере того, как размер и сложность проекта увеличиваются, свойство `currentTarget` становится все более и более полезным.

## Описание методов класса Event

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Методы класса Event делятся на три категории:

- Служебные методы: создают копии объекта события или преобразуют его в строку.
- Методы потока событий, которые удаляют объекты событий из потока событий.
- Методы поведения по умолчанию: отменяют поведение по умолчанию или проверяют, отменено ли оно.

### Служебные методы класса Event

В классе Event существует два служебных метода. Метод `clone()` позволяет создавать копии объекта события. Метод `toString()` позволяет создавать строковое представление свойств объекта события и их значений. Оба этих метода применяются внутренним образом в системе модели событий, но разработчики могут найти им и более общее применение.

Опытным пользователям, создающим подклассы класса Event, необходимо переопределить и реализовать версии обоих служебных методов, чтобы гарантировать правильную работу подкласса события.

### Остановка потока событий

Чтобы предотвратить дальнейшее перемещение объекта события в потоке событий, можно вызвать метод `Event.stopPropagation()` или метод `Event.stopImmediatePropagation()`. Два этих метода практически идентичны и отличаются только тем, разрешено ли выполнение других прослушивателей событий для текущего узла:

- Метод `Event.stopPropagation()` предотвращает перемещение объекта события к следующему узлу. Это происходит только после того, как разрешается выполнение любых прослушивателей событий в текущем узле.

- Метод `Event.stopImmediatePropagation()` предотвращает также перемещение объекта события к следующему узлу, но не разрешает выполнение любых других прослушивателей событий в текущем узле.

Вызов любого из этих методов не влияет на выполнение поведения по умолчанию, связанного с событием. Чтобы предотвратить поведение по умолчанию, воспользуйтесь методами класса Event, связанными с поведением по умолчанию.

**Отмена поведения по умолчанию**

С отменой поведения по умолчанию связаны два метода: `preventDefault()` и `isDefaultPrevented()`. Метод `preventDefault()` отменяет поведение по умолчанию, связанное с событием. Чтобы убедиться в том, что метод `preventDefault()` уже вызван для объекта события, вызовите метод `isDefaultPrevented()`, возвращающий значение `true`, если этот метод уже был вызван, и значение `false` в противном случае.

Метод `preventDefault()` будет работать только в том случае, если поведение по умолчанию отменено. Убедиться в наличии подобной ситуации можно путем ссылки на документацию API для типа события или путем использования ActionScript для изучения свойства `cancelable` объекта события.

Отмена поведения по умолчанию никак не влияет на ход объекта события в потоке. Методы потока событий класса Event удаляют объект события из потока.

## Подклассы класса Event

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Многим событиям достаточно общего набора свойств, определенного классом Event. Однако другие события обладают уникальными характеристиками, которые невозможно зарегистрировать в свойствах, доступных в классе Event. Для этих событий в языке ActionScript 3.0 определено несколько подклассов, расширяющих класс Event.

Каждый подкласс содержит дополнительные свойства и типы событий, уникальные для этой категории событий. Например, события, которые имеют отношение к мыши, имеют несколько уникальных характеристик, которые невозможно записать в свойства, определенные в классе Event. Класс MouseEvent является расширением класса Event. В него добавлены десять свойств, которые содержат такие сведения, как местоположение события мыши и нажатие определенных клавиш во время события мыши.

Подкласс Event содержит также константы, которые представляют типы событий, связанные с подклассом. Например, класс MouseEvent определяет константы для ряда типов событий мыши, включая `click`, `doubleClick`, `mouseDown` и `mouseUp`.

Как описано в пункте «Объекты Event» на странице 140 раздела «Служебные методы класса Event», при создании подкласса Event необходимо переопределить методы `clone()` и `toString()`, чтобы обеспечить функциональные возможности для этого подкласса.

# Прослушиватели событий

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Прослушиватели событий, которых называют также обработчиками событий — это функции, выполняемые Flash Player и AIR в ответ на конкретные события. Добавление прослушивателя событий происходит в два этапа. Во-первых, создается метод функции или класса для Flash Player или AIR, который выполняется в ответ на появление события. Иногда он называется функцией прослушивателя или функцией обработчика событий. Во-вторых, необходимо использовать метод `addEventListener()`, чтобы зарегистрировать свою функцию прослушивателя в цели события или любом объекте из списка отображения, который находится в соответствующем потоке событий.

## Создание функции прослушивателя

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Создание функций прослушивателя — это область, где модель событий ActionScript 3.0 отличается от модели событий DOM. В модели событий DOM существует четкое различие между прослушивателем событий и функцией прослушивателя. Прослушиватель событий является экземпляром класса, реализующего интерфейс EventListener, тогда как функция прослушивателя является методом этого класса с именем `handleEvent()`. В модели событий DOM регистрируется экземпляр класса, содержащий функцию прослушивателя, а не фактическая функция прослушивателя.

В модели событий ActionScript 3.0 прослушиватель события не отличается от функции прослушивателя. ActionScript 3.0 лишен интерфейса EventListener, а функции прослушивателя могут быть определены вне класса или его части. Более того, функции прослушивателя не обязательно называются `handleEvent()`: в качестве имени может выступать любой действительный идентификатор. В ActionScript 3.0 регистрируется имя фактической функции прослушивателя.

**Функция прослушивателя определена вне класса**

Следующий код создает простой SWF-файл, который показывает на экране красный квадрат. Функция прослушивателя с именем `clickHandler()`, которая не принадлежит какому-либо классу, регистрирует события щелчка мыши по красному квадрату.

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

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
}

function clickHandler(event:MouseEvent):void
{
    trace("clickHandler detected an event of type: " + event.type);
    trace("the this keyword refers to: " + this);
}
```

Когда пользователь взаимодействует с итоговым SWF-файлом, щелкая по квадрату, Flash Player или AIR генерирует следующий вывод трассировки:

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

Учтите, что объект события передается в качестве аргумента методу `clickHandler()`. Это позволяет функции прослушивателя изучить объект события. В данном примере свойство `type` используется, чтобы проверить, что событие является событием щелчка.

В примере проверяется также значение ключевого слова `this`. В данном случае `this` представляет глобальный объект. Это имеет смысл, поскольку эта функция определена вне любого заказного класса или объекта.

### Функция прослушивателя, определенная в качестве метода класса

Следующий пример идентичен предыдущему примеру, определяющему класс ClickExample, за исключением того, что функция `clickHandler()` определена как метод класса ChildSprite:

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

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
    private function clickHandler(event:MouseEvent):void
    {
        trace("clickHandler detected an event of type: " + event.type);
        trace("the this keyword refers to: " + this);
    }
}
```

Когда пользователь взаимодействует с итоговым SWF-файлом путем нажатия на красный квадрат, Flash Player или AIR выводят следующие данные трассировки:

```
clickHandler detected an event of type: click
the this keyword refers to: [object ChildSprite]
```

Учтите, что ключевое слово `this` относится к экземпляру ChildSprite с именем `child`. В этом состоит отличие от поведения в ActionScript 2.0. Если вы использовали компоненты в ActionScript 2.0, то могли запомнить, что когда метод класса передавался `UIEventDispatcher.addEventListener()`, область его действия была привязана к компоненту, который выполнял широковещательную рассылку события вместо класса, в котором был определен метод прослушивателя. Иными словами, если вы пользовались этим методом в ActionScript 2.0, ключевое слово `this` будет относиться к компоненту, осуществляющему широковещательную рассылку события, а не к экземпляру ChildSprite.

Это оказалось значительным препятствием для некоторых программистов, поскольку означало, что они не могут вызвать другие методы и свойства класса, содержащего метод прослушивателя. В качестве обходного пути программисты ActionScript 2.0 могут использовать класс `mx.util.Delegate`, чтобы изменить область действия метода прослушивателя. Однако это больше не требуется, так как ActionScript 3.0 создает метод привязки при вызове метода `addEventListener()`. В результате ключевое слово `this` относится к экземпляру ChildSprite с именем `child`, и программист имеет доступ к другим методам и свойствам класса ChildSprite.

**Прослушиватель событий, который не нужно использовать**

Существует третий метод, при выборе которого создается родовой объект со свойством, указывающим на динамически присвоенную функцию прослушивателя, но пользоваться им не рекомендуется. Он обсуждается здесь по причине широкого применения в ActionScript 2.0. Тем не менее, его не следует использовать в ActionScript 3.0. Этот метод не рекомендован, поскольку ключевое слово `this` будет относиться к глобальному объекту вместо вашего объекта прослушивателя.

Следующий пример идентичен предыдущему примеру класса ClickExample за исключением того, что функция прослушивателя определена как часть родового объекта с именем `myListenerObj`:

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

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, myListenerObj.clickHandler);
    }
}

var myListenerObj:Object = new Object();
myListenerObj.clickHandler = function (event:MouseEvent):void
{
        trace("clickHandler detected an event of type: " + event.type);
        trace("the this keyword refers to: " + this);
}
```

Результаты трассировки выглядят следующим образом:

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

Ожидается, что `this` будет ссылаться на `myListenerObj` и что данные о трассировке станут выводиться в формате `[object Object]`, но вместо этого ключевое слово ссылается на глобальный объект. При передаче имени динамического свойства в качестве аргумента методу `addEventListener()` Flash Player или AIR не могут создать метод привязки. Это происходит потому, что в качестве параметра `listener` передается не что иное, как адрес памяти вашей функции прослушивателя. Flash Player и AIR не могут связать этот адрес памяти с экземпляром `myListenerObj`.

## Управление прослушивателями событий

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Функциями прослушивателя можно управлять с помощью методов интерфейса IEventDispatcher. Интерфейс IEventDispatcher — это версия интерфейса EventTarget модели событий DOM для ActionScript 3.0. Хотя и кажется, что основная цель имени IEventDispatcher состоит в отправке (или рассылке) объектов событий, методы этого класса фактически используются гораздо чаще для регистрации прослушивателей событий, проверки наличия прослушивателей событий и их удаления. Интерфейс IEventDispatcher определяет пять методов, как указано в следующем коде:

```
package flash.events
{
    public interface IEventDispatcher
    {
        function addEventListener(eventName:String,
                        listener:Object,
                        useCapture:Boolean=false,
                        priority:Integer=0,
                        useWeakReference:Boolean=false):Boolean;

        function removeEventListener(eventName:String,
                    listener:Object,
                    useCapture:Boolean=false):Boolean;

        function dispatchEvent(eventObject:Event):Boolean;

        function hasEventListener(eventName:String):Boolean;
        function willTrigger(eventName:String):Boolean;
    }
}
```

В Flash Player API интерфейс IEventDispatcher реализуется в классе EventDispatcher, который служит в качестве базового класса для всех классов, которые могут быть целями событий или частью потока событий. Так, например, класс DisplayObject наследует классу EventDispatcher. Это означает, что любой объект в списке отображения имеет доступ к методам интерфейса IEventDispatcher.

### Добавление прослушивателей событий

Метод `addEventListener()` — это основополагающий компонент интерфейса IEventDispatcher. Он используется для регистрации ваших функций прослушивателя. Необходимо обязательно указывать параметры `type` и `listener`. Параметр `type` применяется для определения типа события. Параметр `listener` указывает на функцию прослушивателя, которая выполняется при наступлении события. Параметр `listener` может быть ссылкой либо на функцию, либо на метод класса.

При задании параметра `listener` не используйте скобки. Например, функция `clickHandler()` указывается без скобок в следующем вызове метода `addEventListener()`:

```
addEventListener(MouseEvent.CLICK, clickHandler)
```

Параметр `useCapture` метода `addEventListener()` позволяет управлять фазой потока событий, в течение которой ваш прослушиватель будет активным. Если `useCapture` присвоено значение `true`, ваш прослушиватель будет активным во время фазы захвата потока событий. Если `useCapture` присвоено значение `false`, ваш прослушиватель будет активным во время фаз цели и восходящей маршрутизации потока событий. Чтобы прослушивать событие в ходе всех фаз потока событий, необходимо дважды вызвать метод `addEventListener()`. Первый раз сделайте это с помощью параметра `useCapture` со значением `true`, а второй раз — с помощью параметра `useCapture` со значением `false`.

Параметр `priority` метода `addEventListener()` не является официальной частью модели событий DOM уровня 3. Он включен в ActionScript 3.0 и обеспечивает большую гибкость при организации ваших прослушивателей событий. При вызове метода `addEventListener()` можно установить приоритет для этого прослушивателя событий, передав целое значение в качестве параметра `priority`. Значение по умолчанию равно 0, но можно также выбрать положительное или отрицательное целое значение. Чем больше число, тем скорее будет выполнен этот прослушиватель событий. Прослушиватели событий с одним и тем же приоритетом выполняются в порядке добавления. Чем раньше добавляется прослушиватель, тем скорее он будет выполнен.

Параметр `useWeakReference` позволяет указать, является ли ссылка на функцию прослушивателя слабой или нормальной. Выбор значения `true` для этого параметра позволяет избежать ситуаций, при которых функции прослушивателя остаются в памяти, хотя они уже больше не нужны. Flash Player и AIR используют метод *«сбора мусора»*, чтобы убрать из памяти объекты, которые больше не используются. Объект считается неиспользуемым, если на него больше не содержится ссылок. Сборщик мусора не учитывает слабые ссылки. Это означает, что функция прослушивателя, на которую указывает только слабая ссылка, имеет право быть подвергнутой «сбору мусора».

### Удаление прослушивателей событий

Метод `removeEventListener()` можно использовать для удаления неиспользуемых прослушивателей событий. Ненужные прослушиватели рекомендуется удалять. Обязательными являются параметры `eventName` и `listener` — как и для метода `addEventListener()`. Помните, что вы можете прослушивать события на всех фазах события, дважды вызвав метод `addEventListener()`. Один раз это необходимо сделать с помощью параметра `useCapture` со значением `true`, а другой раз с тем же параметром со значением `false`. Чтобы удалить оба прослушивателя событий, необходимо дважды вызвать метод `removeEventListener()`. Причем в первый раз это делается с помощью параметра `useCapture` со значением `true`, а затем с тем же параметром, которому присвоено значение `false`.

### Отправка событий

Опытные программисты могут использовать метод `dispatchEvent()` для отправки заказного объекта события в поток событий. Единственный параметр, принятый этим методом, является ссылкой на объект события, который должен быть экземпляром класса Event или подкласса этого класса. После отправки свойству `target` объекта события присваивается в качестве значения объект, по отношению к которому вызван метод `dispatchEvent()`.

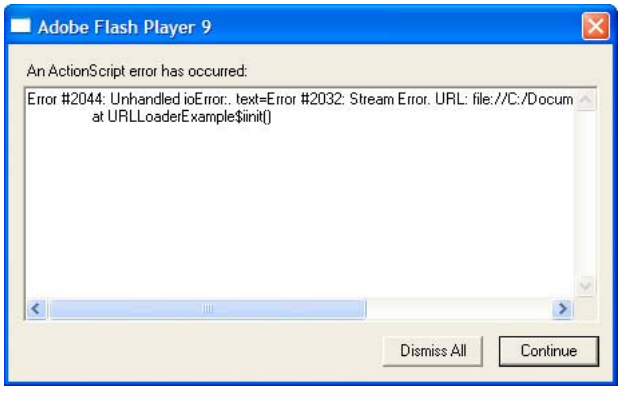
### Проверка существующих прослушивателей событий

Два последних метода интерфейса IEventDispatcher содержат полезную информацию о наличии прослушивателей событий. Метод `hasEventListener()` возвращает значение `true`, если прослушиватель события найден для определенного типа события в том или ином объекте списка отображения. Метод `willTrigger()` возвращает также `true`, если найден прослушиватель для определенного объекта списка отображения. Но метод `willTrigger()` проверяет наличие прослушивателей не только в этом экранном объекте, но и во всех его родительских элемекнтах на всех фазах потока событий.

## События Error без прослушивателей

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для обработки событий в ActionScript 3.0 вместо событий применяются исключения, но обработка событий не действует для асинхронных операций (например, загрузки файлов). Если во время подобной асинхронной операции возникает ошибка, Flash Player и AIR отправляют объект события ошибки. Если для события ошибки не создан прослушиватель, отладочные версии Flash Player и AIR выведут на экран диалоговое окно с информацией об ошибке. Например, отладочная версия проигрывателя Flash Player выводит следующее диалоговое окно с описанием ошибки, когда приложение пытается загрузить файл из недействительного URL-адреса:




Большинство событий ошибок основаны на классе ErrorEvent и имеют свойство `text`, которое используется для хранения сообщения об ошибке, которое выдает Flash Player или AIR. Классы StatusEvent и NetStatusEvent являются двумя исключениями. Оба этих класса обладают свойством `level` (`StatusEvent.level` и `NetStatusEvent.info.level`). Когда свойству `level` присвоено значение `"error"`, эти типы событий рассматриваются в качестве событий ошибок.

Событие ошибки не приведет к остановке воспроизведения SWF-файла. Оно приведет лишь к появлению диалогового окна в отладочных версиях подключаемых модулей веб-обозревателя и автономных проигрывателей, сообщения на панели вывода в проигрывателе инструмента разработки и записи в файле журнала для Adobe Flash Builder. В рабочих версиях Flash Player или AIR оно никак не проявляется.

# Пример обработки событий: будильник

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Пример будильника состоит из часов, которые дают пользователю возможность указать время отключения будильника, а также сообщение, которое необходимо показывать в это время. Пример будильника построен на приложении SimpleClock из раздела «Работа с датами и временем» на странице 1. Будильник иллюстрирует несколько аспектов работы с событиями в ActionScript 3.0, включая:

- прослушивание и реакцию на событие;
- уведомляющие прослушиватели события;
- создание заказного типа события.

Получить файлы приложения Flash Professional для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Получить файлы приложения Flex для этого примера можно на странице http://www.adobe.com/go/as3examples_ru. Файлы приложения «Будильник» находятся в папке Samples/AlarmClock. Приложение включает следующие файлы:

| File | Описание |
|---|---|
| AlarmClockApp.mxml<br>или<br>AlarmClockApp.fla | Основной файл приложения Flash (FLA) или Flex (MXML). |
| com/example/programmingas3/clock/AlarmClock.as | Класс, который является расширением класса SimpleClock, добавляя к нему функцию будильника. |
| com/example/programmingas3/clock/AlarmEvent.as | Класс заказного события (подкласс flash.events.Event), который служит объектом события для события `alarm` класса AlarmClock. |
| com/example/programmingas3/clock/AnalogClockFace.as | Рисует на экране круглый циферблат, а также часовую, минутную и секундную стрелки на основе значения времени (см. описание в примере SimpleClock). |
| com/example/programmingas3/clock/SimpleClock.as | Компонент интерфейса часов с простыми функциональными возможностями хронометрирования (см. описание в примере SimpleClock). |

## Обзор программы «Будильник»

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Основные функциональные возможности часов в данном примере, включая слежение за временем и отображение циферблата, повторно обращаются к коду приложения SimpleClock, который описан в разделе ««Пример даты и времени: простые аналоговые часы» на странице 6». Класс AlarmClock является расширением класса SimpleClock из данного примера. Он добавляет в него функции, необходимые для будильника, включая установку времени сигнала и предоставление уведомления об «отключении» сигнала.

События созданы для того, чтобы предоставлять сведения о происходящем. Класс AlarmClock предоставляет событие Alarm, который другие объекты могут прослушивать, чтобы выполнить необходимые действия. Кроме того, класс AlarmClock использует экземпляр класса Timer, чтобы определить время включения сигнала. Как и класс AlarmClock, класс Timer предоставляет событие для уведомления других объектов (в данном случае экземпляра AlarmClock) по прошествии определенного промежутка времени. Как и в случае большинства приложений на языке ActionScript, события составляют важную часть функциональных возможностей образца приложения «Будильник».

## Включение сигнала

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Как было указано ранее, единственная функция, фактически обеспечиваемая классом AlarmClock, относится к установке времени сигнала и его включению. Встроенный класс Timer (flash.utils.Timer) обеспечивает способ определения разработчиком кода, который выполняется по прошествии определенного промежутка времени. Класс AlarmClock использует экземпляр Timer, чтобы определить время выключения сигнала.

```
import flash.events.TimerEvent;
import flash.utils.Timer;

/**
 * The Timer that will be used for the alarm.
 */
public var alarmTimer:Timer;
...
/**
 * Instantiates a new AlarmClock of a given size.
 */
public override function initClock(faceSize:Number = 200):void
{
    super.initClock(faceSize);
    alarmTimer = new Timer(0, 1);
    alarmTimer.addEventListener(TimerEvent.TIMER, onAlarm);
}
```

Экземпляр Timer, определенный в классе AlarmClock, называется `alarmTimer`. Метод `initClock()`, который совершает необходимые операции установки для экземпляра AlarmClock, выполняет две операции с переменной `alarmTimer`. Во-первых, создается экземпляр переменной с параметрами, заставляющими экземпляр Timer выждать 0 миллисекунд и включить событие timer один раз. После создания экземпляра `alarmTimer` код вызывает метод `addEventListener()` этой переменной, чтобы указать, что он должен прослушивать событие `timer`. Экземпляр Timer действует путем отправки события `timer` по прошествии определенного периода времени. Классу AlarmClock потребуется информация о времени отправки события `timer`, чтобы включить собственный сигнал. Путем вызова метода `addEventListener()` код AlarmClock регистрируется как прослушиватель в `alarmTimer`. Два параметра указывают, что классу AlarmClock необходимо прослушивать событие `timer` (указанное константой `TimerEvent.TIMER`) и что когда это событие произойдет, метод `onAlarm()` класса AlarmClock необходимо вызвать в ответ на это событие.

Чтобы фактически установить сигнал, метод `setAlarm()` класса AlarmClock вызывается следующим образом:

```
/**
 * Sets the time at which the alarm should go off.
 * @param hour The hour portion of the alarm time.
 * @param minutes The minutes portion of the alarm time.
 * @param message The message to display when the alarm goes off.
 * @return The time at which the alarm will go off.
 */
public function setAlarm(hour:Number = 0, minutes:Number = 0, message:String = "Alarm!"):Date
{
    this.alarmMessage = message;
    var now:Date = new Date();
    // Create this time on today's date.
    alarmTime = new Date(now.fullYear, now.month, now.date, hour, minutes);

    // Determine if the specified time has already passed today.
    if (alarmTime <= now)
    {
        alarmTime.setTime(alarmTime.time + MILLISECONDS_PER_DAY);
    }

    // Stop the alarm timer if it's currently set.
    alarmTimer.reset();
    // Calculate how many milliseconds should pass before the alarm should
    // go off (the difference between the alarm time and now) and set that
    // value as the delay for the alarm timer.
    alarmTimer.delay = Math.max(1000, alarmTime.time - now.time);
    alarmTimer.start();

    return alarmTime;
}
```

Этот метод выполняет ряд операций, включая сохранение сообщения сигнала и создание объекта Date (`alarmTime`), отражающего фактический момент выключения сигнала. Наибольшее отношение к обсуждаемой теме имеют последние несколько строк метода, где задан и активирован таймер переменной `alarmTimer`. Во-первых, вызывается его метод `reset()`, который останавливает таймер и сбрасывает его, если он уже запущен. Затем текущее значение времени (выражаемое переменной `now`) вычитается из значения переменной `alarmTime`, чтобы определить, сколько миллисекунд должно пройти перед тем, как будут отключен сигнал. Класс Timer не включает событие `timer` в абсолютное время, поэтому свойству `delay` переменной `alarmTimer` присваивается относительная временная разница. В итоге вызывается метод `start()`, чтобы запустить таймер.

После того, как указанный промежуток времени прошел, `alarmTimer` отправляет событие `timer`. Поскольку класс AlarmClock регистрирует свой метод `onAlarm()` в качестве прослушивателя события, когда происходит событие `timer`, вызывается метод `onAlarm()`.

```
/**
 * Called when the timer event is dispatched.
 */
public function onAlarm(event:TimerEvent):void
{
    trace("Alarm!");
    var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
    this.dispatchEvent(alarm);
}
```

Метод, зарегистрированный в качестве прослушивателя событий, должен быть определен с помощью соответствующей подписи (то есть набора параметров и типа возврата метода). Чтобы быть прослушивателем события `timer` класса Timer, метод должен определить один параметр с типом данных TimerEvent (flash.events.TimerEvent), относящийся к подклассу класса Event. Экземпляр Timer вызывает свои прослушиватели событий, передавая экземпляр TimerEvent в качестве объекта события.

## Уведомление других пользователей о сигнале

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Как и класс Timer, класс AlarmClock предоставляет событие, которое позволяет другому коду получать уведомления об отключении сигнала. Чтобы класс мог использовать систему обработки событий, встроенную в ActionScript, в этом классе должен быть реализован интерфейс flash.events.IEventDispatcher. Как правило, это выполняется путем расширения класса flash.events.EventDispatcher, являющегося стандартной реализацией IEventDispatcher (или путем расширения одного из подклассов EventDispatcher). Как было сказано выше, класс AlarmClock расширяет класс SimpleClock, который в свою очередь расширяет (через цепь наследования) класс EventDispatcher. Это означает, что класс AlarmClock уже имеет встроенные функциональные возможности для предоставления собственных событий.

Другой код может зарегистрироваться, чтобы получать уведомления о событии `alarm` класса AlarmClock путем вызова метода `addEventListener()`, который AlarmClock унаследовал от EventDispatcher. Когда экземпляр AlarmClock готов уведомить другой код о появлении события `alarm`, он делает это путем вызова метода `dispatchEvent()`, который также наследуется от EventDispatcher.

```
var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
this.dispatchEvent(alarm);
```

Эти строки кода взяты из метода `onAlarm()` класса AlarmClock (который ранее был показан целиком). Вызывается метод `dispatchEvent()` экземпляра AlarmClock, который в свою очередь уведомляет всех зарегистрированных прослушивателей о том, что инициировано событие `alarm` экземпляра AlarmClock. Параметр, переданный методу `dispatchEvent()`, является объектом события, который будет передан методам прослушивателя. В данном случае это экземпляр класса AlarmEvent: подкласса Event, созданного специального для этого примера.

## Предоставление заказного события сигнала

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Все прослушиватели событий получают параметр объекта события со сведениями об инициированном событии. Во многих случаях объект события является экземпляром класса Event. Однако в некоторых случаях полезно предоставить прослушивателям событий дополнительную информацию. Распространенным способом для этого является определение нового класса (подкласса класса Event) и использование экземпляра этого класса в качестве объекта события. В данном примере экземпляр AlarmEvent использован в качестве объекта события при отправке события `alarm` класса AlarmClock. Показанный здесь класс AlarmEvent предоставляет дополнительную информацию о событии `alarm`, в частности следующее сигнальное сообщение:

```
import flash.events.Event;

/**
 * This custom Event class adds a message property to a basic Event.
 */
public class AlarmEvent extends Event
{
    /**
     * The name of the new AlarmEvent type.
     */
    public static const ALARM:String = "alarm";

    /**
     * A text message that can be passed to an event handler
     * with this event object.
     */
    public var message:String;

    /**
     *Constructor.
     *@param message The text to display when the alarm goes off.
     */
    public function AlarmEvent(message:String = "ALARM!")
    {
        super(ALARM);
        this.message = message;
    }
    ...
}
```

Наилучший способ создания заказного класса объекта события состоит в том, чтобы определить класс, который является расширением класса Event, как показано в предыдущем примере. В качестве дополнения унаследованных функциональных возможностей класс AlarmEvent определяет свойство `message`, которое содержит текст сигнального сообщения, связанного с событием; значение `message` передается в качестве параметра в конструкторе AlarmEvent. Класс AlarmEvent определяет также константу `ALARM`, которую можно применять для того, чтобы сослаться на конкретное событие (`alarm`) при вызове метода `addEventListener()` класса AlarmClock.

Помимо добавления заказных функций каждый подкласс Event должен переопределить унаследованный метод `clone()` как часть инфраструктуры обработки событий в ActionScript. Подклассы событий могут также при желании переопределять унаследованный метод `toString()`, чтобы включить свойства заказного события в возвращенное значение при вызове метода `toString()`.

```
    /**
     * Creates and returns a copy of the current instance.
     * @return A copy of the current instance.
     */
    public override function clone():Event
    {
        return new AlarmEvent(message);
    }

    /**
     * Returns a String containing all the properties of the current
     * instance.
     * @return A string representation of the current instance.
     */
    public override function toString():String
    {
        return formatToString("AlarmEvent", "type", "bubbles", "cancelable", "eventPhase",
"message");
    }
```

Переопределенному методу `clone()` необходимо вернуть новый экземпляр заказного подкласса Event, где все заказные свойства соответствуют текущему экземпляру. В переопределенном методе `toString()` служебный метод `formatToString()` (унаследованный из класса Event) используется для предоставления строки с именем заказного типа, а также имен и значений всех его свойств.

# Глава 9. Работа с доменами приложений

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс ApplicationDomain предназначен для хранение таблицы определений ActionScript 3.0. Все программные коды в SWF-файле определены для существования в домене приложения. Домены приложений служат для разделения классов, находящихся в одном домене безопасности. Это обеспечивает возможность существования нескольких определений одного класса, а также позволяет дочерним элементам повторно использовать определения родительских элементов.

Домены приложений можно использовать при загрузке внешних SWF-файлов, написанных на языке ActionScript 3.0, с помощью API-интерфейса класса Loader. (Обратите внимание, что домены приложений нельзя использовать для загрузки изображений или SWF-файлов, написанных на языке ActionScript 1.0 или ActionScript 2.0.) Все определения ActionScript 3.0 в загруженном классе хранятся в домене приложения. Загружая SWF-файл, можно включить его в тот же домен приложения, что и объект Loader, задав для параметра `applicationDomain` объекта LoaderContext значение `ApplicationDomain.currentDomain`. Разместив загруженный SWF-файл в том же домене приложений, можно вызывать его классы напрямую. Это полезно, если в загружаемом SWF-файле есть встроенное мультимедийное содержимое, которое можно получить через соответствующие имена классов, или если требуется получить доступ к методам загруженного SWF-файла.

В следующем примере показано, что имеется доступ к отдельному файлу Greeter.swf, который определяет публичный метод с именем welcome():

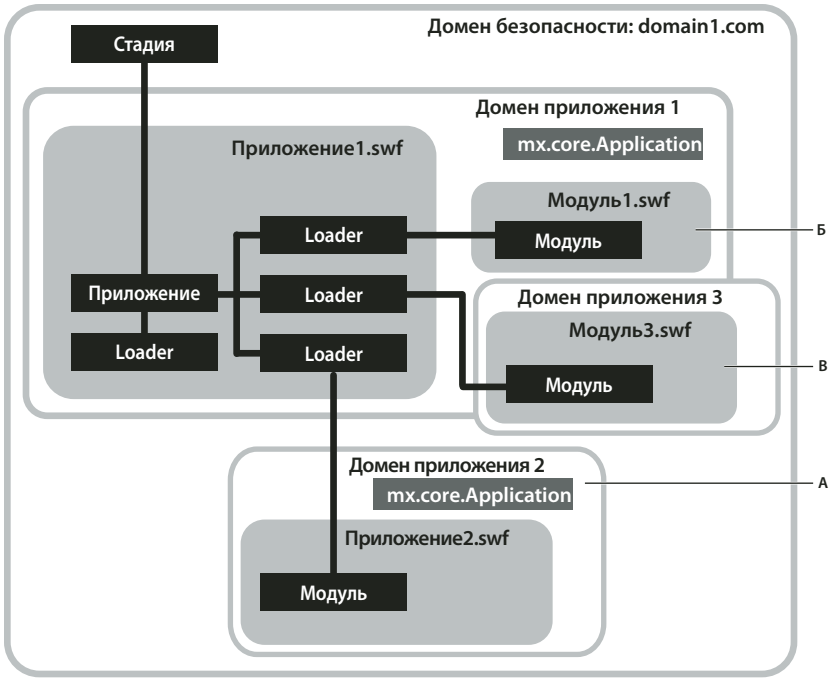
```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLRequest;
    import flash.system.ApplicationDomain;
    import flash.system.LoaderContext;

    public class ApplicationDomainExample extends Sprite
    {
        private var ldr:Loader;
        public function ApplicationDomainExample()
        {
            ldr = new Loader();
            var req:URLRequest = new URLRequest("Greeter.swf");
            var ldrContext:LoaderContext = new LoaderContext(false,
ApplicationDomain.currentDomain);
            ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, completeHandler);
            ldr.load(req, ldrContext);
        }
        private function completeHandler(event:Event):void
        {
            var myGreeter:Class = ApplicationDomain.currentDomain.getDefinition("Greeter") as
Class;
            var myGreeter:Greeter = Greeter(event.target.content);
            var message:String = myGreeter.welcome("Tommy");
            trace(message); // Hello, Tommy
        }
    }
}
```

Дополнительные сведения см. в примере класса ApplicationDomain в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

При работе с доменами приложений следует помнить следующее.

- Все программные коды в SWF-файле определены для существования в домене приложения. *Текущим доменом* приложения является домен, где выполняется основное приложение. *Системный домен* содержит все домены приложений, включая текущий домен, и это означает, что он содержит все классы проигрывателя Flash Player.
- Все домены приложений, кроме системного, имеют связанный родительский домен. Родительским доменом для домена основного приложения является системный домен. Загруженные классы определяются только в том случае, если они уже не определены в своих родительских классах. Нельзя перезаписать определение загруженного класса новым определением.

На следующей схеме показано приложение, которое загружает содержимое из разных SWF-файлов в одном домене, domain1.com. В зависимости от загружаемого содержимого, можно использовать разные домены приложений. Далее описывается логика назначения соответствующего домена приложения для каждого SWF-файла в приложении.




***А.*** *Использование A* ***Б.*** *Использование B* ***В.*** *Использование C*

Основным файлом приложения является application1.swf. Он содержит объекты Loader, которые загружают содержимое из других SWF-файлов. По данному сценарию текущий домен приложения — Application domain 1 (Домен приложения 1). Использования A, B и C демонстрируют разные методы назначения соответствующего домена приложения для каждого SWF-файла в приложении.

**Использование A** Дочерний SWF-файл отделяется путем создания дочернего элемента системного домена. На схеме домен Application domain 2 создается как дочерний элемент системного домена. Файл application2.swf загружается в домен Application domain 2, таким образом, его определения классов отделяются от классов, определенных в файле application1.swf.

Этот прием применяется, чтобы старая версия приложения в динамическом режиме загрузила более новую версию, не создавая конфликта. Конфликт не возникает, так как одноименные классы сохранены раздельно, в разных доменах приложений.

Следующий код создает домен приложения, который является дочерним элементом системного домена, и начинает загружать SWF-файл с использованием этого домена.

```
var appDomainA:ApplicationDomain = new ApplicationDomain();

var contextA:LoaderContext = new LoaderContext(false, appDomainA);
var loaderA:Loader = new Loader();
loaderA.load(new URLRequest("application2.swf"), contextA);
```

**Использование B** К текущим определениям классов добавляются новые. В качестве домена приложения для module1.swf назначен текущий домен (Application domain 1). Это позволяет добавлять новые определения классов в текущий набор определений данного приложения. Этот метод можно использовать для динамической общей библиотеки основного приложения. Загружаемый SWF-файл обрабатывается как удаленная общая библиотека (RSL). Используйте этот прием для загрузки библиотек RSL программой предварительной загрузки до запуска приложения.

Следующий код загружает SWF-файл, назначая текущий домен в качестве домена приложения.

```
var appDomainB:ApplicationDomain = ApplicationDomain.currentDomain;

var contextB:LoaderContext = new LoaderContext(false, appDomainB);
var loaderB:Loader = new Loader();
loaderB.load(new URLRequest("module1.swf"), contextB);
```

**Использование C** Родительские определения классов используются путем создания нового дочернего домена для текущего домена. Доменом приложения для module3.swf является дочерний элемент текущего домена, и этот дочерний элемент использует родительские версии всех классов. Этот метод можно использовать для создания модуля многоэкранного мультимедийного интернет-приложения (RIA), загруженного в качестве дочернего элемента основного приложения и использующего типы данных основного приложения. Если есть возможность обеспечить, чтобы все классы всегда обновлялись для совместимости с более ранними версиями и чтобы загружающее приложение было новее загружаемых файлов, то дочерние домены будут использовать родительские версии. Создание нового домена приложения также позволяет выгружать все определения классов для сборки мусора, если предварительно удалить все ссылки на дочерний SWF-файл.

Этот метод позволяет загруженным модулям совместно использовать единичные объекты и статические элементы класса загружающего файла.

Следующий код создает дочерний домен для текущего домена и начинает загрузку SWF-файла с использованием нового домена приложений.

```
var appDomainC:ApplicationDomain = new ApplicationDomain(ApplicationDomain.currentDomain);

var contextC:LoaderContext = new LoaderContext(false, appDomainC);
var loaderC:Loader = new Loader();
loaderC.load(new URLRequest("module3.swf"), contextC);
```

# Глава 10. Программирование отображаемого содержимого

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

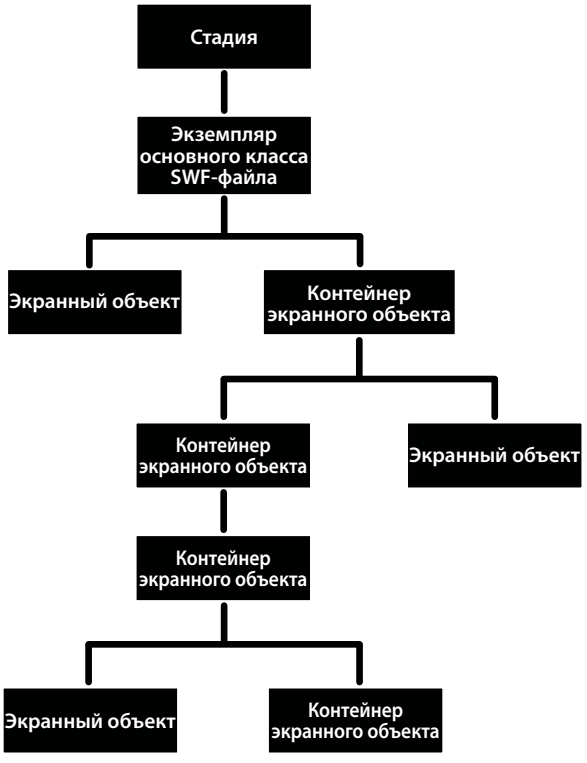
Визуальные элементы программируются в Adobe® ActionScript® 3.0 посредством использования экранных объектов на стадии отображения. Например, можно складывать, перемещать, удалять и располагать по порядку экранные объекты, использовать фильтры и маски, создавать рисунки векторной и растровой графики, выполнять трехмерные преобразования, используя API-интерфейс для программирования отображения на языке ActionScript. Основные классы, используемые для программирования отображения, входят в состав пакета flash.display.

***Примечание.*** *Adobe® AIR™ для визуализации и отображения содержимого HTML предоставляет объект HTMLLoader. HTMLLoader визуализирует визуальные элементы HTML DOM как единый экранный объект. Невозможно получить доступ к отдельным элементам DOM непосредственно через иерархию списка отображения ActionScript. Однако можно получить доступ к ним при помощи отдельного API-интерфейса DOM, предоставляемого объектом HTMLLoader.*

# Основы программирования отображаемого содержимого

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Во всех приложениях, созданных с помощью ActionScript 3.0, существует иерархия отображаемых объектов, которая называется *списком отображения*. См. пример ниже. Список отображения содержит все видимые элементы в приложении.




Как показано на рисунке, экранные объекты делятся на одну или несколько следующих групп:

- Рабочая область

  Рабочая область — это основной контейнер экранных объектов. В каждом приложении есть один объект рабочей области, который содержит все экранные объекты. Рабочая область — это контейнер верхнего уровня иерархии списка отображения:

  С каждым SWF-файлом связан класс ActionScript, который называется *основным классом SWF-файла*. При открытии SWF-файла в Flash Player или Adobe AIR вызывается функция конструктора для этого класса. Создаваемый экземпляр (который всегда принадлежит к типу экранного объекта) добавляется в качестве объекта рабочей области. Основной класс SWF-файла всегда расширяет класс Sprite (дополнительные сведения см. в разделе ««Преимущества подхода с использованием списка отображения» на странице 168»).

  Рабочую область можно вызвать с помощью свойства `stage` любого экземпляра DisplayObject. Дополнительные сведения см. в разделе ««Установка свойств рабочей области» на странице 177».

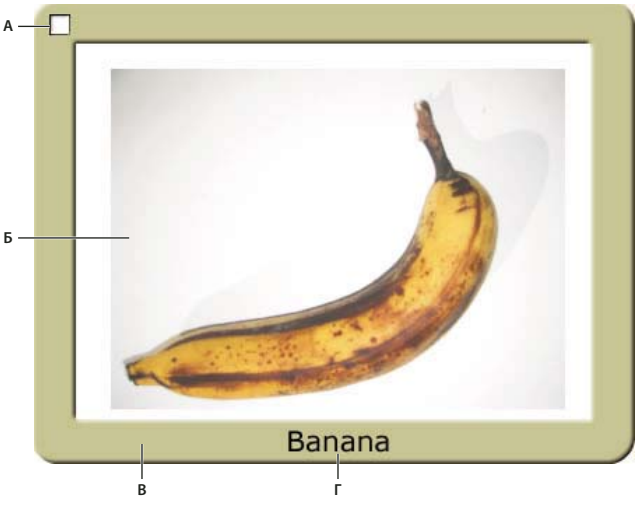
- Экранные объекты

В ActionScript 3.0 все появляющиеся на экране в приложении элементы принадлежат к типу *экранных объектов*. В пакет flash.display входит класс DisplayObject, который является базовым классом, расширяемым рядом других классов. Эти классы представляют разные типы экранных объектов, включая векторные фигуры, фрагменты роликов и текстовые поля. Обзор этих классов см. в разделе ««Преимущества подхода с использованием списка отображения» на странице 168».

- Контейнеры экранных объектов

  Контейнеры экранных объектов — это особые типы экранных объектов, которые могут не только воспроизводиться на экране, но и иметь дочерние экранные объекты.

  Класс DisplayObjectContainer является подклассом для класса DisplayObject. Объект DisplayObjectContainer может содержать экранные объекты в списке *дочернихобъектов*. Например, на следующей иллюстрации показан тип объекта DisplayObjectContainer, который называется Sprite и содержит различные экранные объекты:




***А.** Объект SimpleButton. Этот тип экранного объекта может принимать состояния «up» (кнопка не нажата), «down» (кнопка нажата) и «over» (курсор на кнопке). **Б.** Объект Bitmap. В этом случае объект Bitmap был загружен из внешнего файла JPEG с помощью объекта Loader. **В.** Объект Shape. «Рамка изображения» содержит прямоугольник со скругленными углами, который отрисован в ActionScript. К этому объекту Shape применен фильтр «Тень». **Г.** Объект TextField.*

  В контексте обсуждения экранных объектов объекты DisplayObjectContainer называют также *контейнерами экранных объектов* или просто *контейнерами*. Как отмечалось ранее, рабочая область является контейнером экранного объекта.

  Хотя все видимые экранные объекты наследуют класс DisplayObject, тип каждого из них определяется конкретным подклассом класса DisplayObject. Например, для классов Shape и Video существует функция конструктора, а для класса DisplayObject — нет.

### Важные понятия и термины

Ниже приводится список важных терминов, встречающихся при программировании графики с помощью ActionScript:

**Альфа** Значение цвета, отражающее меру прозрачности (точнее говоря, непрозрачности) цвета. Например, цвет, значение альфа-канала которого составляет 60 %, обладает насыщенностью 60 % и прозрачен на 40 %.

**Растровая графика** Графическое изображение, которое определяется в компьютере как сетка (строки и столбцы) из цветных пикселов. Обычно растровая графика включает цифровые фотографии и подобные изображения.

**Режим наложения** Описание взаимодействия содержимого двух перекрывающихся изображений. Как правило, непрозрачное изображение, размещенное поверх другого изображения, просто загораживает картинку внизу так, что ее совсем не видно. Однако при использовании различных режимов смешивания цвета изображений совмещаются разными способами. В результате в роли итогового содержания выступает то или иное сочетание двух этих изображений.

**Список отображения** Иерархия экранных объектов, которая визуализируется Flash Player и AIR в качестве видимого содержимого экрана. Рабочая область является корневой для списка отображения, и все экранные объекты, которые прикрепляются к рабочей области или ее дочерним объектам, формируют список отображения. Это происходит даже тогда, когда объект фактически не визуализируется (например, если он находится за границами рабочей области).

**Экранный объект** Объект, представляющий некоторый тип визуального содержимого в Flash Player или AIR. В список отображения можно включать только экранные объекты. Все классы экранных объектов являются подклассами класса DisplayObject.

**Контейнер экранного объекта** Особый тип экранного объекта, который может содержать дочерние экранные объекты помимо (обычно) собственного экранного представления.

**Основной класс SWF-файла** Класс, определяющий поведение самого внешнего экранного объекта в SWF-файле, который в сущности является классом самого SWF-файла. Например, в SWF-файле, созданном в среде разработки Flash, основным классом является класс документа. Он имеет «основную временную шкалу», в которой содержатся все остальные временные шкалы. Основной класс SWF-файла — это класс, экземпляром которого является основная временная шкала.

**Маскирование** Метод скрытия определенных частей изображения (или, напротив, их показа). Части изображения маски становятся прозрачными, так что становится видно содержание под ней. Этот термин относится к липкой ленте, которая применяется малярами для предотвращения попадания краски в те или иные области.

**Рабочая область** Визуальный контейнер, который служит основой или фоном всего визуального содержимого SWF-файла.

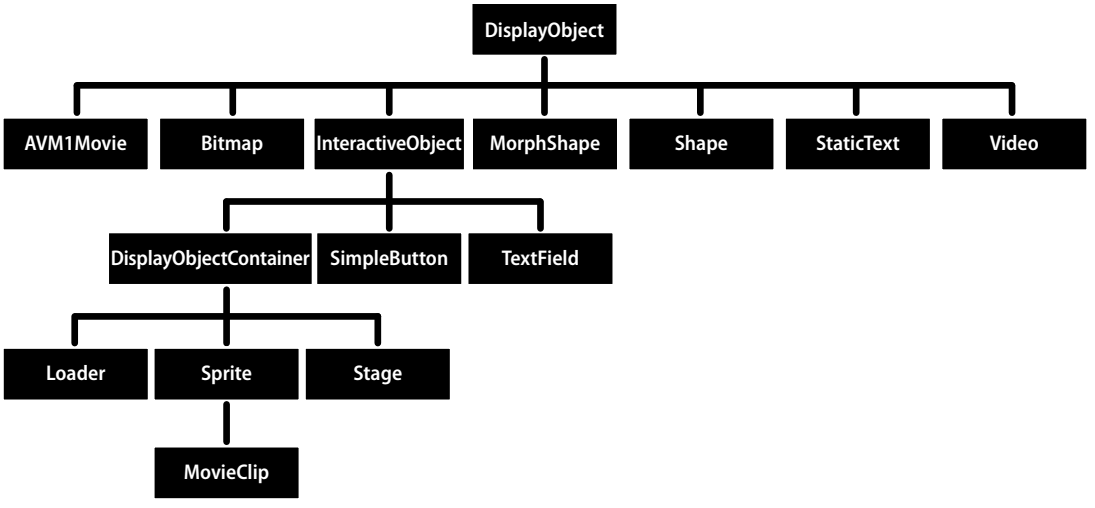
**Преобразование** Корректировка визуальных характеристик графики, например: поворот, масштабирование, сдвиг, деформация или изменение цвета объекта.

**Векторная графика** Графика, которая определяется в компьютере как линии и фигуры, прорисованные с определенными характеристиками (толщина, длина, размер, угол и позиция).

# Основные классы показа

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Пакет flash.display в ActionScript 3.0 включает классы для визуальных объектов, которые могут появиться в Flash Player или AIR. На иллюстрации ниже показаны взаимоотношения между подклассами, относящимися к этим основным классам экранных объектов.




На иллюстрации показано наследование классов, относящихся к экранным объектам. Учтите, что ряд этих классов, особенно StaticText, TextField и Video, отсутствует в пакете flash.display. Однако они все равно наследуют классу DisplayObject.

Все классы, которые являются расширениями класса DisplayObject, наследуют его методы и свойства. Дополнительные сведения см. в разделе ««Свойства и методы класса DisplayObject» на странице 171».

Можно создать экземпляры следующих классов, содержащихся в пакете flash.display.

- Bitmap — класс Bitmap используется для определения растровых объектов, которые загружаются из внешних файлов или визуализируются с помощью ActionScript. Растровые изображения можно загружать из внешних файлов с помощью класса Loader. Загружать можно GIF-, JPG- и PNG-файлы. Кроме того, можно создать объект BitmapData с заказными данными, а затем создать объект Bitmap, в котором они используются. Методы класса BitmapData можно применять для изменения растровых изображений, когда они загружены или созданы с помощью ActionScript. Дополнительные сведения см. в разделах ««Загрузка экранных объектов» на странице 215» и ««Работа с растровыми изображениями» на странице 261».
- Loader — класс Loader применяется для загрузки внешних ресурсов (SWF-файлов или графики). Дополнительные сведения см. в разделе ««Динамическая загрузка содержимого для показа» на странице 214».
- Shape — класс Shape применяется для создания векторной графики (например, прямоугольников, линий, кругов и т.д.) Дополнительные сведения см. в разделе ««Использование API рисования» на странице 239».
- SimpleButton — объект SimpleButton является представлением ActionScript для символа кнопки, созданного в инструменте разработки Flash. У экземпляра SimpleButton имеются 4 состояния кнопки: «не нажата», «нажата», «наведен курсор» и «проверка попаданий» (относится к области, реагирующей на события мыши и клавиатуры).

- Sprite — объект Sprite может содержать собственную графику, а также дочерние экранные объекты. Класс Sprite является расширением класса DisplayObjectContainer. Дополнительные сведения см. в разделе ««Работа с контейнерами экранных объектов» на странице 172» и ««Использование API рисования» на странице 239».
- MovieClip — объект MovieClip представляет собой символ фрагмента ролика, предназначенный для ActionScript и созданный в инструменте разработки Flash. На практике объект MovieClip аналогичен объекту Sprite, за исключением того, что в нем также присутствует временная шкала. Дополнительные сведения см. в разделе ««Работа с фрагментами роликов» на странице 348».

Следующие классы, не входящие в пакет flash.display, являются подклассами класса DisplayObject.

- Класс TextField, входящий в состав пакета flash.text, является экранным объектом для показа и ввода текста. Дополнительные сведения см. в разделе ««Основы работы с текстом» на странице 401».
- Класс TextLine, включенный в пакет flash.text.engine, представляет собой отображаемый объект, используемый для отображения линий текста, которые образованы компонентами Flash Text Engine и Text Layout Framework. Дополнительные сведения см. в разделах ««Использование Flash Text Engine» на странице 428» и ««Использование Text Layout Framework» на странице 458».
- Класс Video, включенный в пакет flash.media, является экранным объектом, используемым для показа видеофайлов. Дополнительные сведения см. в разделе ««Работа с видео» на странице 508».

Далее перечислены классы в пакете flash.display, которые являются расширениями класса DisplayObject. Однако невозможно создать их экземпляры. Вместо этого они служат родительскими классами для других экранных объектов, сочетая общие функциональные возможности в пределах одного класса.

- AVM1Movie — класс AVM1Movie используется для представления загруженных SWF-файлов, разработанных на языках ActionScript 1.0 и 2.0.
- DisplayObjectContainer — классы Loader, Stage, Sprite и MovieClip являются расширениями класса DisplayObjectContainer. Дополнительные сведения см. в разделе ««Работа с контейнерами экранных объектов» на странице 172».
- InteractiveObject — InteractiveObject является базовым классом для всех объектов, используемых для взаимодействия с мышью и клавиатурой. Объекты SimpleButton, TextField, Loader, Sprite, Stage и MovieClip являются подклассами класса InteractiveObject. Дополнительные сведения о создании взаимодействий с мышью и клавиатурой см. в разделе ««Основы взаимодействия пользователя с системой» на странице 597».
- MorphShape — эти объекты создаются при создании анимации формы в средстве разработки Flash. С помощью ActionScript невозможно создать их экземпляры, но их можно вызвать из списка отображения.
- Stage — класс Stage является расширением класса DisplayObjectContainer. В каждом приложении существует один экземпляр Stage, который находится на верхнем уровне иерархии списка отображения. Чтобы вызвать рабочую область, воспользуйтесь свойством `stage` любого экземпляра DisplayObject. Дополнительные сведения см. в разделе ««Установка свойств рабочей области» на странице 177».

Кроме того, класс StaticText в пакете flash.text является расширением класса DisplayObject. Однако создать его экземпляр в коде невозможно. Поля статического текста создаются только в Flash.

Следующие классы не являются отображаемыми объектами и контейнерами отображаемых объектов, а также не включаются в список отображения, но обеспечивают показ графики в рабочей области. Эти классы рисуют прямоугольник, называемый окном просмотра, который позиционируется относительно рабочей области.

- StageVideo — класс StageVideo отображает видеосодержимое, используя по мере возможности аппаратное ускорение. Этот класс стал доступен в Flash Player 10.2. Дополнительные сведения см. в разделе «Использование класса StageVideo для визуализации с аппаратным ускорением» на странице 549.

- StageWebView — класс StageWebView отображает HTML-содержимое. Этот класс доступен, начиная с версии AIR 2.5. Дополнительные сведения см. в разделе ««Объекты StageWebView» на странице 1104».

Следующие классы fl.display обеспечивают такие же функции, что и классы flash.display.Loader и LoaderInfo. Используйте эти классы вместо их аналогов flash.display при разработке приложений в среде Flash Professional (CS5.5 или более поздних версий). В этой среде данные классы помогают решить проблемы, связанные с TLF с предварительной загрузкой RSL. Дополнительную информацию см. в разделе ««Использование классов ProLoader и ProLoaderInfo» на странице 219».

- fl.display.ProLoader — аналог flash.display.Loader
- fl.display.ProLoaderInfo — аналог flash.display.LoaderInfo

# Преимущества подхода с использованием списка отображения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В ActionScript 3.0 отсутствуют отдельные классы для различных типов экранных объектов. В ActionScript 1.0 и 2.0 многие из однотипных объектов включены в один и тот же класс MovieClip.

Обособление классов и иерархической структуры списков отображения имеет следующие преимущества:

- более эффективная визуализация и пониженные требования к памяти;
- Усовершенствованное управление глубиной
- Полный обход списка отображения
- Экранные объекты, не вошедшие в список
- Упрощенное распределение экранных объектов по подклассам

## Более эффективная визуализация и файлы меньшего размера

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В ActionScript 1.0 и 2.0 рисовать фигуры можно только в объекте MovieClip. В ActionScript 3.0 существуют более простые классы экранных объектов, в которых можно рисовать фигуры. Поскольку эти классы экранных объектов ActionScript 3.0 не включают полный набор методов и свойств, которые содержит объект MovieClip, они используют меньше ресурсов процессора и памяти.

Например, любой объект MovieClip включает свойства для временной шкалы фрагмента ролика, тогда как объект Shape не включает их. Свойства, относящиеся к управлению временной шкалой, могут требовать много ресурсов процессора и памяти. В ActionScript 3.0 использование объекта Shape приводит к повышению производительности. Объект Shape имеет меньше служебных данных, чем сложный объект MovieClip. Flash Player и AIR не требуется оперировать неиспользованными свойствами объекта MovieClip. Это повышает скорость его обработки и снижает объем используемой памяти.

## Усовершенствованное управление глубиной

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В ActionScript 1.0 и 2.0 глубина регулировалась с помощью линейной схемы и таких методов, как `getNextHighestDepth()`.

ActionScript 3.0 включает класс DisplayObjectContainer, который обладает более удобными методами и свойствами для управления глубиной экранных объектов.

В ActionScript 3.0 при перемещении экранного объекта на новое положение в списке дочерних объектов экземпляра DisplayObjectContainer другие дочерние объекты в контейнере экранного объекта автоматически смещаются. Им присваиваются соответствующие позиции в указателе дочерних объектов, относящемся к контейнеру экранного объекта.

Также в ActionScript 3.0 всегда можно обнаружить все дочерние объекты любого контейнера экранного объекта. Каждый экземпляр DisplayObjectContainer обладает свойством `numChildren`, в котором перечислено количество дочерних элементов в контейнере экранного объекта. И поскольку список дочерних объектов контейнера экранных объектов всегда является индексированным, вы можете изучить любой объект в списке, начиная с позиции указателя 0 до последней позиции (`numChildren - 1`). Это было невозможно сделать с помощью методов и свойств объекта MovieClip в ActionScript 1.0 и 2.0.

В ActionScript 3.0 можно легко выполнить последовательный обход списка отображения; номера в указателе списка дочерних объектов для контейнера экранного объекта идут один за другим. Обход списка отображения и управление глубиной объектов сильно упрощены по сравнению с ActionScript 1.0 и 2.0. В ActionScript 1.0 и 2.0 фрагмент ролика мог содержать объекты с перемежающимися разрывами в порядке глубины, которые могут усложнить обход списка объекта. В ActionScript 3.0 каждый список дочерних объектов контейнера экранного объекта кэшируется внутри в качестве массива, что приводит к резкому ускорению поиска по указателю. Циклический просмотр всех дочерних объектов контейнера экранного объекта также происходит очень быстро.

В ActionScript 3.0 дочерние объекты можно также вызвать в контейнере экранного объекта путем использования метода `getChildByName()` класса DisplayObjectContainer.

## Полный обход списка отображения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В ActionScript 1.0 и 2.0 некоторые объекты (например, векторные фигуры, отрисованные в инструменте разработки Flash) вызвать невозможно. В ActionScript 3.0 все объекты в списке отображения доступны (как объекты, созданные с помощью ActionScript, так и все объекты, созданные в инструменте разработки Flash. Дополнительные сведения см. в разделе ««Обход списка отображения» на странице 175».

## Экранные объекты, не вошедшие в список

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В ActionScript 3.0 можно создавать экранные объекты, которые не находятся в выводимом на экран списке отображения. Они называются экранными объектами, *не вошедшими в список*. Экранный объект добавляется в видимый список отображения только при вызове метода `addChild()` или `addChildAt()` экземпляра DisplayObjectContainer, который уже был добавлен в список отображения.

Экранные объекты, не входящие в список, можно использовать для сборки сложных экранных объектов (например, объектов, имеющих ряд контейнеров экранных объектов, в которых содержится несколько экранных объектов). Не внося экранные объекты в список, можно собирать сложные объекты без затрат времени обработки при визуализации этих экранных объектов. После этого можно добавить экранный объект, который входит в список отображения, когда необходимо. Кроме того, можно переместить дочерний объект контейнера экранного объекта в список отображения, выбрав любую позицию списка отображения по желанию.

## Упрощенное распределение экранных объектов по подклассам

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В ActionScript 1.0 и 2.0 часто приходилось добавлять новые объекты MovieClip в SWF-файл, чтобы создать основные фигуры или показать растровые изображения. В ActionScript 3.0 класс DisplayObject содержит многие встроенные подклассы, включая Shape и Bitmap. Поскольку классы в ActionScript 3.0 больше специализируются на конкретных типах объектов, легче создавать базовые подклассы на основе встроенных классов.

Например, чтобы нарисовать круг в ActionScript 2.0, можно было бы создать класс CustomCircle, который является расширением класса MovieClip при создании экземпляра объекта, принадлежащего к заказному классу. Однако этот класс будет также включать ряд свойств и методов из класса MovieClip (например, `totalFrames`), которые к нему не относятся. Однако в ActionScript 3.0 можно создать класс CustomCircle, который является расширением объекта Shape, и в качестве такового не включает несвязанные свойства и методы, которые содержатся в классе MovieClip. Следующий код является примером класса CustomCircle:

```
import flash.display.*;

public class CustomCircle extends Shape
{
    var xPos:Number;
    var yPos:Number;
    var radius:Number;
    var color:uint;
    public function CustomCircle(xInput:Number,
                                 yInput:Number,
                                 rInput:Number,
                                 colorInput:uint)
    {
        xPos = xInput;
        yPos = yInput;
        radius = rInput;
        color = colorInput;
        this.graphics.beginFill(color);
        this.graphics.drawCircle(xPos, yPos, radius);
    }
}
```

# Работа с экранными объектами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Теперь, когда вы разобрались с основными понятиями, которые связаны с рабочей областью, экранными объектами, контейнерами экранных объектов и списком отображения, в этом разделе можно ознакомиться с более конкретными сведениями о работе с экранными объектами в ActionScript 3.0.

## Свойства и методы класса DisplayObject

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Все экранные объекты являются подклассами класса DisplayObject и в качестве таковых наследуют свойства и методы класса DisplayObject. Унаследованные свойства являются основными свойствами, применяемыми ко всем экранным объектам. Например, у каждого экранного объекта существуют свойства `x` и `y`, которые указывают на позицию объекта в контейнере экранного объекта.

Экземпляр DisplayObject невозможно создать с помощью конструктора класса DisplayObject. Необходимо создать другой тип объекта (объект, который является подклассом класса DisplayObject, например, Sprite), чтобы создать экземпляр объекта с оператором `new`. Кроме того, если необходимо создать заказной класс экранного объекта, вы должны создать подкласс одного из подклассов экранного объекта, у которого есть пригодная к использованию функция конструктора (например, класс Shape или класс Sprite). Дополнительные сведения см. в описании класса DisplayObject в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

## Добавление экранных объектов в список отображения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При создании экземпляра экранного объекта он не появится на экране (в рабочей области) до тех пор, пока вы не добавите экземпляр экранного объекта в контейнер экранного объекта, находящийся в списке отображения. Например, в следующем коде объект `myText` TextField не будет виден, если вы пропустили последнюю строку кода. В последней строке кода ключевое слово `this` должно относиться к контейнеру экранного объекта, добавленному в список отображения.

```
import flash.display.*;
import flash.text.TextField;
var myText:TextField = new TextField();
myText.text = "Buenos dias.";
this.addChild(myText);
```

При добавлении любого визуального элемента в рабочую область этот элемент становится *дочерним объектом* объекта Stage. Первый SWF-файл, загруженный в приложение (например, встроенный в страницу HTML), автоматически добавляется в качестве дочернего объекта рабочей области. Это может быть объект любого типа, который является расширением класса Sprite.

Любые экранные объекты, созданные *без* использования ActionScript (например, за счет добавления тега MXML в файл Flex MXML или путем размещения элемента в рабочей области Flash Professional), добавляются в список отображения. Хотя эти экранные объекты не добавляются с помощью ActionScript, их можно вызывать благодаря ему. Например, следующий код регулирует ширину объекта `button1`, добавленного в инструмент разработки (не с помощью ActionScript):

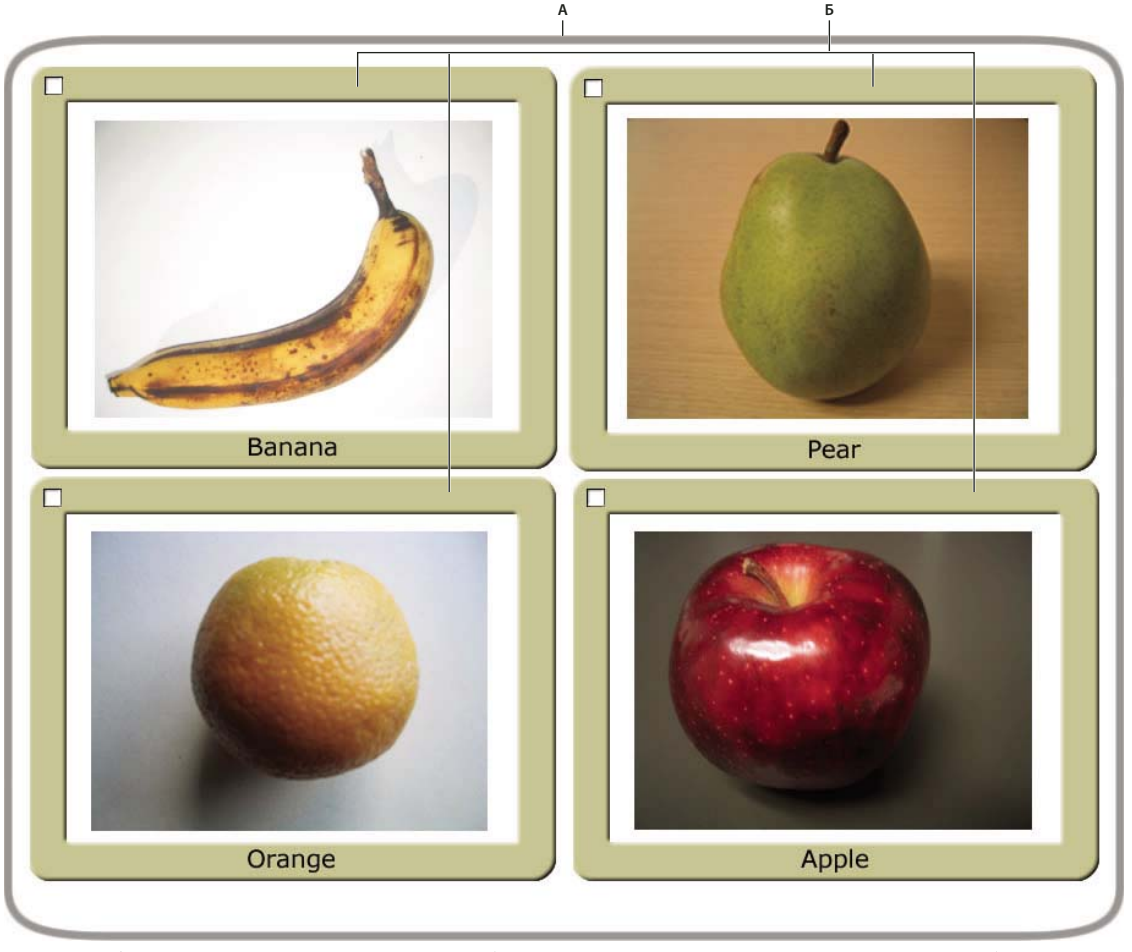
```
button1.width = 200;
```

## Работа с контейнерами экранных объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если объект DisplayObjectContainer удаляется из списка отображения, а также перемещается или иным способом преобразуется, то каждый экранный объект в DisplayObjectContainer также удаляется, перемещается или преобразуется.

Сам по себе контейнер экранного объекта является типом экранного объекта, и его можно добавить в контейнер экранного объекта. Например, на следующем изображении показан контейнер экранного объекта (`pictureScreen`), который содержит одну контурную фигуру и четыре других контейнера экранных объектов (принадлежащих типу PictureFrame):




***А.** Фигура, задающая границу контейнера экранного объекта pictureScreen **Б.** Четыре контейнера экранных объектов, являющихся дочерними объектами объекта pictureScreen*

Чтобы экранный объект появился в списке отображения, его нужно добавить в контейнер экранного объекта, находящийся в списке отображения. Это делается с помощью метода `addChild()` или метода `addChildAt()` объекта контейнера. Например, объект `myTextField` не удастся вывести на экран без последней строки следующего кода:

```
var myTextField:TextField = new TextField();
myTextField.text = "hello";
this.root.addChild(myTextField);
```

В этом примере кода `this.root` указывает на контейнер экранного объекта MovieClip, содержащий код. В вашей собственной версии кода можно указать другой контейнер.

Воспользуйтесь методом `addChildAt()` для добавления дочернего объекта на конкретную позицию в списке дочерних объектов контейнера экранного объекта. Эти позиции указателя, начинающегося с нуля, в списке дочерних объектов, относятся к расположению экранных объектов (от переднего слоя к заднему). Например, рассмотрите следующие три экранных объекта. Каждый объект создавался на основе заказного класса Ball.

Расположение этих экранных объектов по слоям в контейнере можно отрегулировать с помощью метода `addChildAt()`. Например, рассмотрим следующий код:

```
ball_A = new Ball(0xFFCC00, "a");
ball_A.name = "ball_A";
ball_A.x = 20;
ball_A.y = 20;
container.addChild(ball_A);

ball_B = new Ball(0xFFCC00, "b");
ball_B.name = "ball_B";
ball_B.x = 70;
ball_B.y = 20;
container.addChild(ball_B);

ball_C = new Ball(0xFFCC00, "c");
ball_C.name = "ball_C";
ball_C.x = 40;
ball_C.y = 60;
container.addChildAt(ball_C, 1);
```

После выполнения этого кода экранные объекты располагаются в объекте `container` DisplayObjectContainer следующим образом. Обратите внимание на слои расположения объектов.

Чтобы переместить объект в верхнюю часть списка отображения, просто снова добавьте его в список. Например, после предыдущего кода, чтобы переместить `ball_A` в верхнюю часть стопки, воспользуйтесь следующей строкой кода:

```
container.addChild(ball_A);
```

Этот код удаляет `ball_A` с позиции в списке отображения `container` и снова добавляет его в верхнюю часть списка. В конечном итоге он перемещается в верхнюю часть стопки.

Можно использовать метод `getChildAt()` для проверки порядка слоев, в которых располагаются экранные объекты. Метод `getChildAt()` возвращает дочерние объекты контейнера на основе переданного ему номера указателя. Например, следующий код обнаруживает имена экранных объектов на различных позициях в списке дочерних объектов объекта `container` DisplayObjectContainer:

```
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_C
trace(container.getChildAt(2).name); // ball_B
```

При удалении экранного объекта из списка дочерних объектов родительского контейнера позиции родительских элементов списка смещаются вниз в указателе дочерних объектов. Например, в продолжение разговора о предыдущем коде, в следующем коде показано, как экранный объект, который находился на позиции 2 в `container` DisplayObjectContainer, перемещается на позицию 1 при удалении экранного объекта, который находится ниже в списке дочерних объектов:

```
container.removeChild(ball_C);
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_B
```

Методы `removeChild()` и `removeChildAt()` не удаляют экземпляр объекта целиком. Они просто удаляют его из списка дочерних объектов контейнера. При этом данный экземпляр может быть представлен другой переменной. Чтобы полностью удалить объект, воспользуйтесь оператором `delete`.

Поскольку экранный объект имеет только один родительский контейнер, экземпляр экранного объекта можно добавить только в один контейнер экранного объекта. Например, следующий код демонстрирует, что экранный объект `tf1` может существовать только в одном контейнере (в данном случае это Sprite, являющийся расширением класса DisplayObjectContainer):

```
tf1:TextField = new TextField();
tf2:TextField = new TextField();
tf1.name = "text 1";
tf2.name = "text 2";

container1:Sprite = new Sprite();
container2:Sprite = new Sprite();

container1.addChild(tf1);
container1.addChild(tf2);
container2.addChild(tf1);

trace(container1.numChildren); // 1
trace(container1.getChildAt(0).name); // text 2
trace(container2.numChildren); // 1
trace(container2.getChildAt(0).name); // text 1
```

Если вы добавляете экранный объект, который содержится в одном контейнере экранных объектов, в другой контейнер экранных объектов, он удаляется из списка дочерних объектов первого контейнера экранных объектов.

Помимо описанных выше методов класс DisplayObjectContainer определяет ряд методов для работы с дочерними экранными объектами, включая указанные ниже.

- `contains()`: определяет, является ли объект дочерним объектом DisplayObjectContainer.
- `getChildByName()`: извлекает экранный объект по имени.

- `getChildIndex()`: возвращает позицию индекса для экранного объекта.
- `setChildIndex()`: изменяет позицию дочернего экранного объекта.
- `removeChildren()`: удаляет несколько дочерних объектов отображения.
- `swapChildren()`: меняет местами два экранных объекта, ставя один из них перед другим или позади него.
- `swapChildrenAt()`: меняет местами два экранных объекта, ставя один из них перед другим или позади него в зависимости от их значений в указателе.

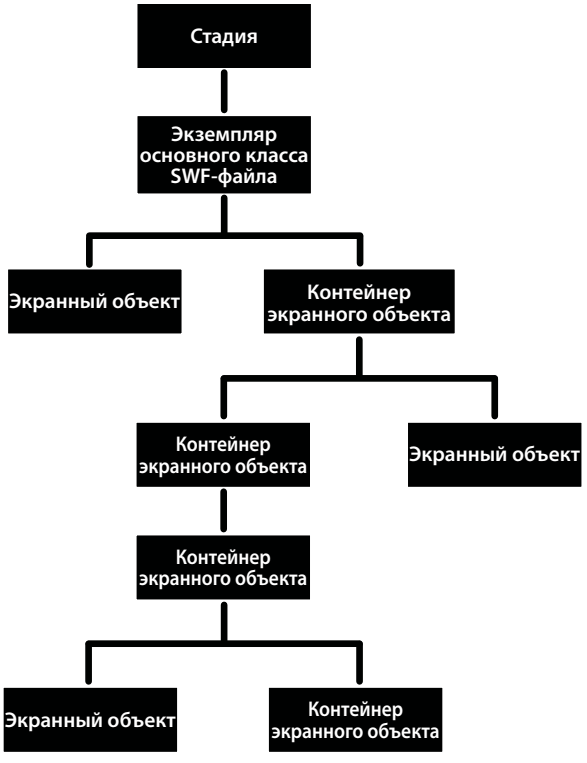
Дополнительные сведения см. в соответствующих разделах справочника ActionScript® 3.0 для платформы Adobe® Flash® Platform.

Вспомните, что экранный объект, который отсутствует в списке отображения (то есть не включенный в контейнер экранного объекта, являющийся дочерним объектом рабочей сцены), известен как экранный объект, *не входящий в список*.

## Обход списка отображения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Как вы заметили, список отображения представляет собой древовидную структуру. В верхней части дерева находится рабочая область, которая может содержать несколько экранных объектов. Те экранные объекты, которые сами являются контейнерами экранных объектов, могут содержать другие экранные объекты или их контейнеры.



Класс DisplayObjectContainer включает свойства и методы для обхода списка отображения с помощью списков дочерних объектов с контейнерами экранных объектов. Например, рассмотрим следующий код, который добавляет два экранных объекта (`title` и `pict`) в объект `container` (который относится к классу Sprite, являющемуся расширением класса DisplayObjectContainer):

```
var container:Sprite = new Sprite();
var title:TextField = new TextField();
title.text = "Hello";
var pict:Loader = new Loader();
var url:URLRequest = new URLRequest("banana.jpg");
pict.load(url);
pict.name = "banana loader";
container.addChild(title);
container.addChild(pict);
```

Метод `getChildAt()` возвращает дочерний объект списка отображения, который находится на определенной позиции указателя:

```
trace(container.getChildAt(0) is TextField); // true
```

Доступ к дочерним объектам может также осуществляться по имени. Каждый экранный объект обладает свойством имени, и если его не присвоить, Flash Player или AIR присваивает значение по умолчанию (например, `"instance1"`). На примере следующего кода демонстрируется применение метода `getChildByName()` для доступа к дочернему экранному объекту с именем `"banana loader"`:

```
trace(container.getChildByName("banana loader") is Loader); // true
```

Использование метода `getChildByName()`, в отличие от метода `getChildAt()`, может привести к снижению производительности.

Так как контейнер экранного объекта может содержать другие контейнеры экранных объектов в качестве дочерних объектов в своем списке отображения, весь список отображения приложения можно обойти как дерево. Например, в указанном ранее фрагменте кода после завершения операции загрузки объекта `pict` Loader объект `pict` будет иметь один дочерний экранный объект, в качестве которого выступает загруженное растровое изображение. Чтобы вызвать этот экранный объект растрового изображения, можно написать `pict.getChildAt(0)`. Кроме того, можно ввести `container.getChildAt(0).getChildAt(0)` (так как `container.getChildAt(0) == pict`).

Следующая функция приводит к выводу с отступом результата выполнения метода `trace()`, относящегося к списку отображения из контейнера экранного объекта:

```
function traceDisplayList(container:DisplayObjectContainer,indentString:String = ""):void
{
    var child:DisplayObject;
    for (var i:uint=0; i < container.numChildren; i++)
    {
        child = container.getChildAt(i);
        trace(indentString, child, child.name);
        if (container.getChildAt(i) is DisplayObjectContainer)
        {
            traceDisplayList(DisplayObjectContainer(child), indentString + "")
        }
    }
}
```

**Adobe Flex**

Если вы пользуетесь программой Flex, вы должны знать, что в ней определяются многие классы экранных объектов, и эти классы переопределяют методы доступа к списку отображения для класса DisplayObjectContainer. Например, класс Container из пакета mx.core переопределяет метод `addChild()` и другие методы класса DisplayObjectContainer (расширением которого является класс Container). В случае метода `addChild()` этот класс переопределяет его таким образом, что вы не можете добавить все типы экранных объектов в экземпляр контейнера Container во Flex. Переопределенный метод в данном случае требует, чтобы добавляемый дочерний объект относился к типу объекта mx.core.UIComponent.

---

## Установка свойств рабочей области

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс Stage переопределяет большинство свойств и методов класса DisplayObject. При вызове переопределенных свойств или методов Flash Player и AIR выдают исключение. Например, объект Stage лишен свойств `x` или `y`, поскольку его положение в качестве основного контейнера приложения фиксировано. Свойства `x` и `y` связаны с позицией экранного объекта по отношению к его контейнеру. Поскольку Stage не содержится в другом контейнере экранного объекта, эти свойства неприменимы.

***Примечание.*** *Некоторые свойства и методы класса Stage доступны только для экранных объектов, которые находятся в той же изолированной программной среде, что и загруженный SWF-файл. Дополнительные сведения см. в разделе ««Безопасность рабочей области» на странице 1147».*

### Управление частотой кадров при воспроизведении

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Свойство `framerate` класса Stage используется для установки частоты кадров для всех SWF-файлов, загруженных в приложение. Дополнительные сведения см. в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

### Управление масштабом рабочей области

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Когда изменяется размер части экрана, занимаемой проигрывателями Flash Player или AIR, содержимое среды выполнения автоматически корректируется в соответствии с изменениями. Свойство `scaleMode` класса Stage определяет метод корректировки содержимого. Это свойство может принимать 4 разных значения, которые определяются как константы в классе flash.display.StageScaleMode:

- `StageScaleMode.EXACT_FIT` изменяет масштаб SWF для заполнения рабочей области с новыми размерами без учета исходного соотношения сторон. Коэффициенты масштабирования по ширине и высоте могут отличаться, поэтому если соотношение сторон изменится, содержимое может выглядеть сжатым или растянутым.
- `StageScaleMode.SHOW_ALL` изменяет масштаб SWF для заполнения всей рабочей области с новыми размерами без изменения соотношения сторон содержимого. При данном режиме масштабирования отображается все содержимое, но возможно появление границ «letterbox», например черных полос, которые появляются при просмотре широкоэкранного фильма на стандартном телевизоре.

- `StageScaleMode.NO_BORDER` изменяет масштаб SWF для заполнения всей рабочей области с новыми размерами без изменения соотношения сторон содержимого. Данный режим масштабирования позволяет полностью использовать область отображения рабочей области, однако содержимое при этом может обрезаться.
- `StageScaleMode.NO_SCALE` не изменяет масштаб SWF. Если новая рабочая область имеет меньшие размеры, содержимое обрезается. Если она имеет большие размеры, добавленное пространство будет пустым.

  В режиме масштабирования `StageScaleMode.NO_SCALE` и ни в каком другом свойства `Width` и `Height` класса Stage можно использовать для определения фактических размеров измененной рабочей области в пикселах. При других режимах масштабирования свойства `stageWidth` и `stageHeight` всегда отражают исходную длину и ширину SWF-файла. Кроме того, когда свойству `scaleMode` присвоено значение `StageScaleMode.NO_SCALE`, а размер SWF-файла изменяется, отправляется событие `resize` класса Stage, позволяющее внести корректировки соответствующим образом.

  Следовательно, если `scaleMode` имеет значение `StageScaleMode.NO_SCALE`, это позволяет при желании точнее управлять корректировкой содержимого экрана, зависящей от изменения размеров экрана. Например, в SWF-файле, содержащем видеоролик и строку управления, может потребоваться сохранять размер строки управления при изменении размера рабочей области. При этом размер окна показа видеофайла изменяется в соответствии с изменением размера рабочей области. Это демонстрируется в следующем примере.

```
// mainContent is a display object containing the main content;
// it is positioned at the top-left corner of the Stage, and
// it should resize when the SWF resizes.

// controlBar is a display object (e.g. a Sprite) containing several
// buttons; it should stay positioned at the bottom-left corner of the
// Stage (below mainContent) and it should not resize when the SWF
// resizes.

import flash.display.Stage;
import flash.display.StageAlign;
import flash.display.StageScaleMode;
import flash.events.Event;

var swfStage:Stage = mainContent.stage;
swfStage.scaleMode = StageScaleMode.NO_SCALE;
swfStage.align = StageAlign.TOP_LEFT;
swfStage.addEventListener(Event.RESIZE, resizeDisplay);

function resizeDisplay(event:Event):void
{
    var swfWidth:int = swfStage.stageWidth;
    var swfHeight:int = swfStage.stageHeight;

    // Resize the main content area
    var newContentHeight:Number = swfHeight - controlBar.height;
    mainContent.height = newContentHeight;
    mainContent.scaleX = mainContent.scaleY;

    // Reposition the control bar.
    controlBar.y = newContentHeight;
}
```

**Настройка режима масштабирования рабочей области для окон AIR**

Свойство `scaleMode` объекта Stage определяет, как рабочая область масштабирует и обрезает дочерние экранные объекты при изменении размеров окна. В AIR следует использовать только режим `noScale`. В этом режиме рабочая область не масштабируется. Вместо этого размер рабочей области изменяется непосредственно вместе с границами окна. При уменьшении размера окна, объекты могут обрезаться.

Режимы масштабирования рабочей области предназначены для использования в таких средах, как веб-браузер, где не всегда есть возможность управлять размером или соотношением сторон рабочей области. Эти режимы позволяют выбирать наименее искаженный вариант, когда рабочая область не соответствует идеальному размеру или соотношению сторон окна приложения. В приложении AIR вы всегда управляете рабочей областью, поэтому в большинстве случаев повторное размещение содержимого или просто корректировка размеров окна дадут лучшие результаты, чем масштабирование рабочей области.

В браузере и в начальном окне AIR соотношение между размером окна и начальным фактором масштабирования считывается из загруженного SWF-файла. Однако при создании объекта NativeWindow среда AIR выбирает произвольное соотношение между размером окна ил фактором масштабирования 72:1. Таким образом, если размер окна составляет 72x72 пиксела, квадрат 10x10, добавляемый в окно, отрисовывается с правильным размером 10x10 пикселов. Однако если размер окна составляет 144x144 пиксела, квадрат 10x10 масштабируется до размера 20x20 пикселов. Если все же требуется использовать режим `scaleMode`, а не `noScale` для рабочей области окна, в качестве компенсации можно задать для всех экранных объектов в окне фактор масштабирования 72 пикселов к текущей ширине и высоте рабочей области. Например, следующий код вычисляет необходимый фактор масштабирования для экранного объекта с именем `client`.

```
if(newWindow.stage.scaleMode != StageScaleMode.NO_SCALE){
client.scaleX = 72/newWindow.stage.stageWidth;
client.scaleY = 72/newWindow.stage.stageHeight;
}
```

***Примечание.*** *Окна Flex и HTML автоматически задают свойству `scaleMode` объекта Stage значение `noScale`. Изменение свойства `scaleMode` мешает работе механизмов автоматической компоновки, применяемых в окнах этих типов.*

---

## Работа в полноэкранном режиме

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Полноэкранный режим позволяет рабочей области ролика заполнить весь экран без каких-либо границ контейнера или меню. Свойство класса Stage `displayState` используется для входа и выхода из полноэкранного режима показа SWF-файла. Свойству `displayState` можно присвоить одно из значений, определенных для констант в классе flash.display.StageDisplayState. Чтобы включить полноэкранный режим, присвойте свойству `displayState` значение `StageDisplayState.FULL_SCREEN`:

```
stage.displayState = StageDisplayState.FULL_SCREEN;
```

Чтобы включить полноэкранный интерактивный режим (новая возможность Flash Player 11.3), задайте свойство `displayState` со значением `StageDisplayState.FULL_SCREEN_INTERACTIVE`:

```
stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
```

В Flash Player к полноэкранному режиму можно перейти с помощью ActionScript только в ответ на щелчок мышью (включая правую кнопку) или нажатие клавиши. Содержимое AIR, которое выполняется в изолированной программной среде, не требует перехода к полноэкранному режиму в ответ на жест пользователя.

Чтобы выключить полноэкранный режим, присвойте свойству `displayState` значение `StageDisplayState.NORMAL`.

```
stage.displayState = StageDisplayState.NORMAL;
```

Кроме того, пользователь может выходить из полноэкранного режима путем смещения фокуса на другое окно или с помощью одного из нескольких сочетаний клавиш: Esc (все платформы), Control-W (Windows), Command-W (Mac) или Alt-F4 (Windows).

### Включение полноэкранного режима в Flash Player

Для включения полноэкранного режима для SWF-файла, встроенного в HTML-страницу, код HTML, встраиваемый в Flash Player, должен включать тег `param`, атрибут `embed` с именем `allowFullScreen` и значение `true`, как указано ниже:

```
<object>
    ...
    <param name="allowFullScreen" value="true" />
    <embed ... allowFullScreen="true" />
</object>
```

В инструменте разработки Flash выберите «Файл» > «Параметры публикации» и в открывшемся окне на вкладке «HTML» укажите шаблон «Только Flash — разрешить полноэкранный режим».

В Flex проверьте, что шаблон HTML включает теги `<object>` и `<embed>`, поддерживающие полноэкранный режим.

Если на веб-странице при создании тегов для встраивания SWF-файлов используется язык JavaScript, то для добавления тега `allowFullScreen param` и атрибута необходимо лишь изменить JavaScript. Например, если в вашей HTML-странице используется функция `AC_FL_RunContent()` (которая применяется на страницах HTML, созданных с помощью Flash Professional и Flash Builder), необходимо следующим образом добавить параметр `allowFullScreen` к этому вызову функции:

```
AC_FL_RunContent(
    ...
    'allowFullScreen','true',
    ...
    ); //end AC code
```

Это не относится к SWF-файлам, запущенным в автономной программе Flash Player.

***Примечание.*** *Если для режима окна (wmode в HTML) выбрать значение «Непрозрачный безоконный» (opaque) или «Прозрачный безоконный» (transparent), окно в полноэкранном режиме всегда будет непрозрачным.*

Существуют также ограничения системы безопасности на использование полноэкранного режима Flash Player в обозревателе. Эти ограничения описаны в разделе ««Безопасность» на странице 1121».

### Включение полноэкранного интерактивного режима в проигрывателе Flash Player 11.3 и более поздних версий

Проигрыватель Flash Player 11.3 и более поздних версий поддерживает полноэкранный интерактивный режим, который обеспечивает полную поддержку для всех клавиш (кроме **Esc**, которая завершает работу в полноэкранном интерактивном режиме). Полноэкранный интерактивный режим полезен в игровых приложениях (например, для включения чата в многопользовательской игре или элементов управления клавиатуры WASD в стрелялках с видом от первого лица).

Для включения полноэкранного интерактивного режима для SWF-файла, встроенного в HTML-страницу, код HTML, встраиваемый в Flash Player, должен включать тег `param`, атрибут `embed` с именем `allowFullScreenInteractive` и значение `true`, как указано ниже:

```
<object>
    ...
    <param name="allowFullScreenInteractive" value="true" />
    <embed ... allowFullScreenInteractive="true" />
</object>
```

В инструменте разработки Flash выберите «Файл» > «Параметры публикации» и в открывшемся окне на вкладке «HTML» укажите шаблон «Только Flash — разрешить полноэкранный интерактивный режим».

В Flash Builder и Flex убедитесь, что шаблоны HTML содержат теги `<object>` и `<embed>`, поддерживающие полноэкранный интерактивный режим.

Если на веб-странице при создании тегов для встраивания SWF-файлов используется язык JavaScript, то для добавления тега `allowFullScreenInteractive param` и атрибута необходимо лишь изменить JavaScript. Например, если в вашей HTML-странице используется функция `AC_FL_RunContent()` (которая применяется на страницах HTML, созданных с помощью Flash Professional и Flash Builder), необходимо следующим образом добавить параметр `allowFullScreenInteractive` к этому вызову функции:

```
AC_FL_RunContent(
    ...
    'allowFullScreenInteractive','true',
    ...
    ); //end AC code
```

Это не относится к SWF-файлам, запущенным в автономной программе Flash Player.

### Полноэкранный режим и масштабирование

Свойства `Stage.fullScreenHeight` и `Stage.fullScreenWidth` возвращают высоту и ширину монитора, которые используются при переходе к размеру во весь экран, если он выполняется немедленно. Эти значения могут быть неправильными, если пользователь имеет возможность переместить браузер с одного монитора на другой после получения этих значений, но до перехода в полноэкранный режим. Если эти значения были получены в том же обработчике событий, где свойству `Stage.displayState` было присвоено значение `StageDisplayState.FULL_SCREEN`, то значения являются верными. При наличии нескольких подключенных мониторов, содержимое SWF полностью займет только один из них. Flash Player и AIR используют метрику, чтобы определить, какой из мониторов содержит большую часть SWF-файла, выбирая этот монитор для полноэкранного режима. Свойства fullScreenHeight и fullScreenWidth только отражают размер монитора, используемого для полноэкранного режима. Дополнительные сведения см. в описаниях свойств `Stage.fullScreenHeight` и `Stage.fullScreenWidth` в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

Масштабирование рабочей области в полноэкранном режиме протекает так же, как и в обычном режиме. Масштабирование управляется свойством `scaleMode` класса Stage. Если свойству `scaleMode` присвоено значение `StageScaleMode.NO_SCALE`, свойства `stageWidth` и `stageHeight` рабочей области изменяются, отражая размер области экрана, занятой SWF-файлом (в данном случае это весь экран). При просмотре в обозревателе параметр HTML управляет этой настройкой.

Событие `fullScreen` класса Stage используется, чтобы узнать, включен или выключен полноэкранный режим, и среагировать соответствующим образом. Например, может потребоваться изменить положение, добавить или удалить элементы с экрана при переходе к полноэкранному режиму или выходе из него как в этом примере:

```
import flash.events.FullScreenEvent;

function fullScreenRedraw(event:FullScreenEvent):void
{
    if (event.fullScreen)
    {
        // Remove input text fields.
        // Add a button that closes full-screen mode.
    }
    else
    {
        // Re-add input text fields.
        // Remove the button that closes full-screen mode.
    }
}

mySprite.stage.addEventListener(FullScreenEvent.FULL_SCREEN, fullScreenRedraw);
```

Как показано в этом коде, объект события для события `fullScreen` является экземпляром класса flash.events.FullScreenEvent, включающего свойство `fullScreen`, которое указывает, включен ли полноэкранный режим (`true`) или нет (`false`).

### Поддержка клавиатуры в полноэкранном режиме

При запуске Flash Player в обозревателе в полноэкранном режиме все сценарии ActionScript, связанные с использованием клавиатуры, например события клавиатуры или ввод текста в экземпляры TextField, отключаются. Исключениями (клавиши, которые остаются доступными) являются:

- Клавиши, не содержащие букв или цифр, а именно стрелки, пробел и клавиша табуляции
- Сочетания клавиш, используемые для выхода из полноэкранного режима: Esc (Windows и Mac), Control-W (Windows), Command-W (Mac), и Alt-F4

Данные ограничения не действуют для содержимого SWF, выполняемого в автономном Flash Player или в приложении AIR. AIR поддерживает интерактивный полноэкранный режим, который допускает ввод с клавиатуры.

### Поддержка мыши в полноэкранном режиме

По умолчанию события мыши в полноэкранном режиме происходят таким же образом, как и не в полноэкранном режиме. Однако в полноэкранном режиме можно установить необязательное свойство `Stage.mouseLock`, чтобы включить блокировку мыши. При блокировке мыши курсор отключается, и включается несвязанное движение мыши.

***Примечание.*** *Блокировку мыши можно включить только в полноэкранном режиме для настольных приложений. При установке блокировки не в полноэкранном режиме или для приложений на мобильных устройствах будет возникать исключение.*

Блокировка мыши автоматически отключается и курсор мыши снова становится видимым в следующих случаях:

- Пользователь отключает полноэкранный режим нажатием клавиш Escape (все платформы), Control-W (Windows), Command-W (Mac) или Alt-F4 (Windows).
- Окно приложения теряет фокус.
- Любой пользовательский интерфейс настроек является видимым, включая все диалоговые окна конфиденциальности.

- Показано собственное диалоговое окно, такое как диалоговое окно отправки файла.

События, связанные с движением мыши, такие как `mouseMove`, используют класс MouseEvent для представления объекта события. Если блокировка мыши отключена, для определения положения мыши используйте свойства `MouseEvent.localX` и `MouseEvent.localY`. Если блокировка мыши включена, для определения положения мыши используйте свойства `MouseEvent.movementX` и `MouseEvent.movementY`. Свойства `movementX` и`movementY` содержат изменения положения мыши, произошедшие с последнего события, вместо абсолютных координат положения мыши.

### Аппаратное масштабирование в полноэкранном режиме

Свойство `fullScreenSourceRect` класса Stage можно использовать для того, чтобы заставить Flash Player или AIR масштабировать определенную зону рабочей области, переведя ее в режим полноэкранного показа. При наличии такой возможности Flash Player и AIR используют аппаратное масштабирование средствами доступного в системе пользователя видеоадаптера, что позволяет отображать содержимое быстрее, чем при использовании программного масштабирования.

Чтобы использовать аппаратное масштабирование, необходимо задать полноэкранный режим для всей рабочей области или ее части. Следующий код ActionScript 3.0 задает полноэкранный режим для всей рабочей области.

```
import flash.geom.*;
{
    stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

Когда этому свойству присвоено значение допустимого прямоугольника, а свойство `displayState` настроено на полноэкранный режим, проигрыватель Flash Player и AIR масштабируют заданную область. Фактический размер рабочей области в пикселах в ActionScript не изменяется. Flash Player и AIR используют ограничение минимального размера прямоугольника при показе стандартного сообщения «Нажмите «Esc» для выхода из полноэкранного режима». Обычно минимальный размер составляет около 260 на 30 пикселов, но он может варьироваться в зависимости от платформы и версии Flash Player.

*Свойство `fullScreenSourceRect` можно настраивать, только когда проигрыватель Flash Player или AIR не находится в полноэкранном режиме. Для правильной работы этого свойства, необходимо сначала настроить именно его, а затем настроить свойство `displayState` на работу в полноэкранном режиме.*

Чтобы включить масштабирование, задайте для свойства `fullScreenSourceRect` объект прямоугольника.

```
stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
```

Чтобы отключить масштабирование, присвойте свойству `fullScreenSourceRect` значение `null`.

```
stage.fullScreenSourceRect = null;
```

Чтобы задействовать все возможные аппаратные функции ускорения в Flash Player, включите их в диалоговом окне «Свойства Flash Player». Чтобы открыть диалоговое окно с настройками, щелкните правой кнопкой мыши (Windows) или осуществите щелчок правой кнопкой мыши при зажатой клавише Control (Mac) по содержимому Flash Player в вашем обозревателе. Перейдите на первую вкладку, «Отображение», и установите флажок «Включить аппаратное ускорение».

**Режимы окна: прямой и совмещенный с использованием графического процессора**

В проигрыватель Flash Player 10 добавлено два режима окна: прямой и совмещенный с использованием графического процессора, которые можно включить через параметры публикации в инструменте разработки Flash. Эти режимы не поддерживаются в AIR. Для использования этих режимов необходимо включить аппаратное ускорение в проигрывателе Flash Player.

Прямой режим использует самый быстрый и наиболее прямой путь для вывода графики на экран, что полезно при воспроизведении видео.

Режим совмещения с использованием графического процессора ускоряет совмещение, используя графический процессор видеокарты. Совмещение видео — это процесс наложения нескольких слоев изображения для получения одного видеоизображения. Когда совмещение ускоряется с помощью графического процессора, оно может улучшить производительность YUV-преобразования, коррекции цвета, поворота или масштабирования и наложения YUV-преобразованием называется преобразование цветов композитных аналоговых сигналов, используемых для передачи, в модель цвета RGB (красный, зеленый, синий), которая используется видеокамерами и дисплеями. Использование графического процессора для ускорения совмещения сокращает потребление ресурсов памяти и процессора. Кроме того, это улучшает воспроизведение видео со стандартным разрешением.

Соблюдайте осторожность, применяя эти режимы окна. Совмещение с использованием графического процессора может интенсивно использовать память и ресурсы центрального процессора. Если некоторые операции (такие как режимы наложения, фильтрация, обрезание или маскирование) не могут выполняться графическим процессором, они выполняются программным обеспечением. Компания Adobe рекомендует добавлять не более одного SWF-файла на HTML-страницу, когда используются эти режимы. Кроме того, их не следует включать для баннеров. Функция «Тестировать ролик» инструмента разработки Flash не использует аппаратное ускорение, но его можно использовать, выбрав опцию «Просмотр публикации».

Не имеет смысла задавать в SWF-файле частоту кадров выше 60, то есть максимальной частоты обновления экрана. При использовании частоты кадров от 50 до 55 допускается пропуск кадров, который может время от времени происходить по разным причинам.

Для прямого режима требуется Microsoft DirectX 9 и 128 МБ видеопамяти на платформе Windows и OpenGL для Apple Macintosh, Mac OS X 10.2 и более поздних версий. Для режима совмещения с использованием графического процессора требуется поддержка Microsoft DirectX 9 и Pixel Shader 2.0 на платформе Windows с 128 МБ видеопамяти. На платформе Mac OS X и Linux для режима совмещения с использованием графического процессора требуется OpenGL 1.5 и несколько расширений OpenGL (объект буфера кадра, мультитекстурирование, объекты шейдеров, язык шейдера, шейдер фрагмента).

Режимы ускорения (`direct` и `gpu`) для отдельного SWF-файла можно активировать через диалоговое окно «Параметры публикации» в Flash в меню «Аппаратное ускорение» на вкладке «Flash». Если выбрать вариант «Нет», режим окна возвращается к значению `default`, `transparent` или `opaque` в соответствии с вариантом, выбранным на вкладке «HTML».

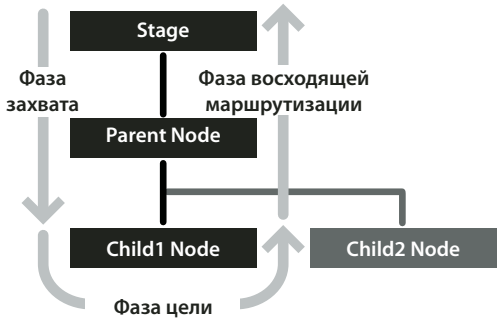
## Обработка событий для экранных объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс DisplayObject наследует классу EventDispatcher. Это означает, что каждый экранный объект может в полной мере участвовать в модели события (описанной в разделе ««Обработка событий» на странице 134»). Каждый экранный объект может использовать свой метод `addEventListener()`, унаследованный из класса EventDispatcher, для ожидания определенного события, но лишь в том случае, если этот объект принадлежит к потоку событий для этого события.

Когда Flash Player или AIR отправляет объект события, этот объект события возвращается в рабочую область к экранному объекту, где это событие произошло. Например, если пользователь щелкает экранный объект `child1`, Flash Player отправляет объект события из рабочей области с помощью иерархии списка отображения вниз к экранному объекту `child1`.

Поток событий делится на три фазы, как показано на этой диаграмме:




Дополнительные сведения см. в разделе ««Обработка событий» на странице 134».

При работе с событиями экранных объектов необходимо учитывать влияние, которое прослушиватели событий могут иметь на то, удаляются ли экранные объекты автоматически из памяти («сбор мусора») при удалении из списка отображения. Если у экранного объекта есть объекты, зарегистрированные в качестве прослушивателей его событий, то он не будет удаляться из памяти даже при удалении из списка отображения, поскольку у него все равно будут иметься ссылки на объекты прослушивателя. Дополнительные сведения см. в разделе ««Управление прослушивателями событий» на странице 149».

## Выбор подкласса DisplayObject

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Имея несколько параметров на выбор, при работе с экранными объектами вам предстоит принять среди прочих важное решение относительно того, какой экранный объект какой цели соответствует. Ниже приводится ряд рекомендаций, которые помогут вам определиться с выбором. Те же самые советы применяются тогда, когда необходим экземпляр класса, или при выборе базового класса для создаваемого класса:

- Если вам не нужен объект, который может быть контейнером для других экранных объектов (то есть просто необходим объект, который служит автономным экранным элементом), выберите один из этих подклассов DisplayObject или InteractiveObject в зависимости от того, для чего он будет использоваться:
  - Bitmap для показа растрового изображения.
  - TextField для добавления текста.
  - Video для показа видеофайлов.
  - Shape для создания «холста», позволяющего рисовать содержимое на экране. В частности, если вы хотите создать экземпляр для отрисовки фигур на экране, и он не будет контейнером для других экранных объектов, вы получите значительный прирост производительности, используя Shape вместо Sprite или MovieClip.

    - MorphShape, StaticText или SimpleButton для элементов, созданных в инструменте разработки Flash. (Экземпляры этих классов нельзя создать программным методом, но вы можете создавать переменные с этими типами данных, чтобы обращаться к элементам, созданным с помощью инструмента разработки Flash.)
- Если необходима переменная, относящаяся к основной рабочей области, в качестве типа данных выберите класс Stage.
- Если вам необходим контейнер для загрузки внешнего SWF-файла или файла изображения, воспользуйтесь экземпляром Loader. Загруженное содержимое будет добавлено в список отображения в качестве дочернего объекта экземпляра Loader. Его тип данных будет зависеть от характера загруженного содержания следующим образом:
    - Загруженное изображение станет экземпляром Bitmap.
    - Загруженный SWF-файл, написанный на ActionScript 3.0, будет экземпляром Sprite или MovieClip (или экземпляром подкласса этих классов, как указано создателем содержимого).
    - Загруженный SWF-файл, написанный на языках ActionScript 1.0 или ActionScript 2.0, будет экземпляром AVM1Movie.
- Если вам необходим объект, который будет служить контейнером для других экранных объектов (рисование на экранном объекте с помощью ActionScript), выберите один из подклассов DisplayObjectContainer:
    - Sprite (если объект будет создан только с помощью ActionScript или в качестве базового класса для заказного экранного объекта, который будет создаваться и обрабатываться исключительно средствами ActionScript).
    - MovieClip (если вы создаете переменную, относящуюся к символу фрагмента ролика, созданному в инструменте разработки Flash).
- Если вы создаете класс, который будет связан с символом фрагмента ролика в библиотеке Flash, выберите один из указанных ниже подклассов DisplayObjectContainer в качестве базового класса вашего класса:
    - MovieClip (если связанный символ фрагмента ролика имеет содержимое более чем в одном кадре)
    - Sprite (если связанный символ фрагмента ролика имеет содержимое только в первом кадре)

# Операции с экранными объектами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Независимо от того, какой экранный объект вы выберете, существует ряд операций, применимых ко всем экранным объектам, которые выступают в роли элементов, отображаемых на экране. Например, их все можно расположить на экране, переместить вперед или назад, изменив порядок наложения экранных объектов, изменить масштаб, выполнить вращение и так далее. Поскольку все экранные объекты наследуют эти функциональные возможности из общего базового класса (DisplayObject), они действуют одинаково при операциях с экземплярами TextField, Video, Shape или любого другого экранного объекта. В следующих разделах подробно описаны эти общие операции с экранными объектами.

## Изменение положения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Наиболее основная операция, применяющаяся к любому экранному объекту, заключается в размещении его на экране. Чтобы задать положение экранного объекта, измените свойства `x` и `y` объекта.

```
myShape.x = 17;
myShape.y = 212;
```

Система размещения экранных объектов воспринимает рабочую область как декартову систему координат (обычная сетка с горизонтальной осью x и вертикальной осью y). Начальная точка системы координат (точка с координатами 0,0, где перекрещиваются оси x и y) находится в верхнем левом углу рабочей области. Отсюда по оси x вправо идут положительные значения, а влево — отрицательные, в то время как положительные значения по оси y направлены вниз, а отрицательные — вверх (в отличие от типичных графических систем). Например, предыдущие строки кода приводят к перемещению объекта `myShape` в точку с координатой 17 по оси x (17 пикселов вправо от начальной точки) и с координатой 212 по оси y (212 пикселов вниз от начальной точки).

Когда экранный объект создается с помощью ActionScript, свойства `x` и `y` по умолчанию принимают значение 0, а объект помещается в левом верхнем углу родительского содержимого.

### Изменение положения относительно рабочей области

Важно помнить, что свойства `x` и `y` всегда относятся к позиции экранного объекта относительно точки с координатами 0,0 относительно осей родительского экранного объекта. Для экземпляра Shape (например, круга), содержащегося в экземпляре Sprite, присваивание свойствам `x` и `y` значения 0 приводит к размещению круга в верхнем левом углу спрайта, который не обязательно совпадает с левым верхним углом рабочей области. Чтобы разместить объект относительно глобальных координат рабочей области, можно воспользоваться методом `globalToLocal()` любого экранного объекта, чтобы преобразовать координаты из глобальных (рабочая область) в локальные (контейнер экранного объекта). Например, следующим образом:

```
// Position the shape at the top-left corner of the Stage,
// regardless of where its parent is located.

// Create a Sprite, positioned at x:200 and y:200.
var mySprite:Sprite = new Sprite();
mySprite.x = 200;
mySprite.y = 200;
this.addChild(mySprite);

// Draw a dot at the Sprite's 0,0 coordinate, for reference.
mySprite.graphics.lineStyle(1, 0x000000);
mySprite.graphics.beginFill(0x000000);
mySprite.graphics.moveTo(0, 0);
mySprite.graphics.lineTo(1, 0);
mySprite.graphics.lineTo(1, 1);
mySprite.graphics.lineTo(0, 1);
mySprite.graphics.endFill();

// Create the circle Shape instance.
var circle:Shape = new Shape();
mySprite.addChild(circle);

// Draw a circle with radius 50 and center point at x:50, y:50 in the Shape.
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0xff0000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();

// Move the Shape so its top-left corner is at the Stage's 0, 0 coordinate.
var stagePoint:Point = new Point(0, 0);
var targetPoint:Point = mySprite.globalToLocal(stagePoint);
circle.x = targetPoint.x;
circle.y = targetPoint.y;
```

Кроме того, можно воспользоваться методом `localToGlobal()` класса DisplayObject, чтобы преобразовать локальные координаты в координаты рабочей области.

## Перемещение экранных объектов с помощью мыши

Пользователю можно предоставить возможность перемещать экранные объекты с помощью мыши, используя два разных приема в ActionScript. В обоих случаях применяется два события мыши. Когда кнопка мыши нажата, объекту передается инструкция следовать за курсором, а когда она отпущена, передается инструкция о прекращении следования за курсором.

***Примечание.*** *Flash Player 11.3 и AIR 3.3 или более поздних версий: можно также использовать событие MouseEvent.RELEASE_OUTSIDE на тот случай, если пользователь отпускает кнопку мыши за границами содержащего объекта Sprite.*

Первый прием предполагает использование метода `startDrag()`. Он проще, но имеет больше ограничений. При нажатии кнопки мыши вызывается метод `startDrag()` экранного объекта, который необходимо перетащить. Когда кнопка отпущена, вызывается метод `stopDrag()`. Класс Sprite определяет эти две функции, поэтому перемещаемый объект должен быть экземпляром класса Sprite или одного из его подклассов.

```
// This code creates a mouse drag interaction using the startDrag()
// technique.
// square is a MovieClip or Sprite instance).

import flash.events.MouseEvent;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    square.startDrag();
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    square.stopDrag();
}

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

У этого метода есть одно довольно существенное ограничение: с помощью метода `startDrag()` можно перетаскивать только один объект за раз. Если перетаскивается один экранный объект, а метод `startDrag()` вызывается для другого экранного объекта, первый экранный объект сразу же перестает следовать за мышью. Например, если функция `startDragging()` изменяется так, как здесь показано, будет перетаскиваться только объект `circle`, несмотря на вызов метода `square.startDrag()`:

```
function startDragging(event:MouseEvent):void
{
    square.startDrag();
    circle.startDrag();
}
```

Из-за того, что одновременно с помощью метода `startDrag()` можно перетащить только один объект, метод `stopDrag()` можно вызвать по отношению к любому экранному объекту. При этом перетаскивание любого другого объекта останавливается.

Если вам необходимо перетащить более одного экранного объекта или избежать возможных конфликтов, которые возникают, когда метод `startDrag()` используется более чем одним объектом, используйте технику следования за мышью, чтобы добиться эффекта перетаскивания. В случае данной техники когда нажимается кнопка, функция регистрируется в качестве прослушивателя события `mouseMove` рабочей области. Эта функция, которая затем вызывается при каждом движении мыши, приводит к тому, что перетаскиваемый объект переходит к точке расположения курсора с координатами x и y. После отпускания кнопки мыши функция перестает восприниматься как прослушиватель. Это означает, что она больше не вызывается при перемещениях мыши, и объект перестает следовать за курсором. Ниже приведен код, иллюстрирующий данный метод:

```
// This code moves display objects using the mouse-following
// technique.
// circle is a DisplayObject (e.g. a MovieClip or Sprite instance).

import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // Record the difference (offset) between where
    // the cursor was when the mouse button was pressed and the x, y
    // coordinate of the circle when the mouse button was pressed.
    offsetX = event.stageX - circle.x;
    offsetY = event.stageY - circle.y;

    // tell Flash Player to start listening for the mouseMove event
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragCircle(event:MouseEvent):void
{
    // Move the circle to the location of the cursor, maintaining
    // the offset between the cursor's location and the
    // location of the dragged object.
    circle.x = event.stageX - offsetX;
    circle.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Часто требуется, чтобы экранный объект не просто следовал за курсором мыши, но и оставался на переднем плане во время перетаскивания. При этом создается впечатление, что он плавает поверх всех остальных объектов. Предположим, что имеется два объекта, круг и квадрат, которые можно перетаскивать с помощью мыши. Если круг находится под квадратом в списке отображения, и вы щелкнете и перетащите его, чтобы курсор оказался над квадратом, вам покажется, что круг зашел за квадрат. Это вызовет нарушение иллюзии. Вместо этого можно сделать так, что при нажатии круга он переместится на вершину списка отображения, и при этом всегда будет находиться над остальным содержимым.

Следующий код (адаптированный из предыдущего примера) позволяет перемещать с помощью мыши два экранных объекта, круг и квадрат. Каждый раз, когда на любой из них наводится курсор и нажимается кнопка мыши, соответствующий элемент перемещается в верхнюю часть списка отображения рабочей области и из-за этого при перетаскивании он всегда находится сверху. (Новый код или измененный код из предыдущего примера выделен полужирным.)

```
// This code creates a drag-and-drop interaction using the mouse-following
// technique.
// circle and square are DisplayObjects (e.g. MovieClip or Sprite
// instances).

import flash.display.DisplayObject;
import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;
var draggedObject:DisplayObject;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // remember which object is being dragged
    draggedObject = DisplayObject(event.target);

    // Record the difference (offset) between where the cursor was when
    // the mouse button was pressed and the x, y coordinate of the
    // dragged object when the mouse button was pressed.
    offsetX = event.stageX - draggedObject.x;
    offsetY = event.stageY - draggedObject.y;

    // move the selected object to the top of the display list
    stage.addChild(draggedObject);

    // Tell Flash Player to start listening for the mouseMove event.
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}
```

```
// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragObject(event:MouseEvent):void
{
    // Move the dragged object to the location of the cursor, maintaining
    // the offset between the cursor's location and the location
    // of the dragged object.
    draggedObject.x = event.stageX - offsetX;
    draggedObject.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

Чтобы расширить сферу применения этого эффекта (например, для игры, в которой фишки или карты перемещаются среди стопок, можно добавить перетаскиваемый объект в список отображения рабочей области, когда его «поднимают», а затем добавить его в другой список отображения (например, «стопку», в которую его следует уронить) при отпускании кнопки мыши.

И, наконец, чтобы усилить этот эффект, можно применить фильтр тени к экранному объекту при нажатии (когда вы начнете его перетаскивать) и удалить тень при отпускании объекта. Сведения об использовании фильтра тени и других фильтров экранного объекта в ActionScript см. в разделе ««Фильтрация экранных объектов» на странице 289».

## Панорамирование и прокрутка экранных объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если у вас есть экранный объект, который является слишком большим для области, в которой его требуется показать, можно воспользоваться свойством `scrollRect` для определения видимой области экранного объекта. Кроме того, путем изменения свойства `scrollRect` в ответ на данные, введенные пользователем, можно включить панорамирование содержимого влево или вправо, а также прокрутку вверх и вниз.

Свойство `scrollRect` является экземпляром класса Rectangle. В этом классе сочетаются значения, которые необходимы для определения прямоугольной области в качестве единичного объекта. Чтобы определить вначале видимую область экранного объекта, создайте новый экземпляр Rectangle и присвойте его свойству `scrollRect` экранного объекта. Впоследствии для панорамирования или прокрутки свойство `scrollRect` считывается и заносится в отдельную переменную Rectangle. Требуемое свойство изменяется (например, измените свойство `x` экземпляра Rectangle, чтобы выполнить панорамирование, или свойство `y`, чтобы выполнить прокрутку). Затем необходимо переназначить экземпляр Rectangle свойству `scrollRect`, чтобы сообщить объекту об изменении значения.

Например, следующий код определяет обозримую область для объекта TextField с именем `bigText`, который слишком высок, чтобы вместиться в границы SWF-файла. При нажатии двух кнопок (`up` и `down`) они вызывают функции, отвечающие за прокрутку содержимого объекта TextField вверх и вниз путем изменения свойства `y` экземпляра Rectangle`scrollRect`.

```
import flash.events.MouseEvent;
import flash.geom.Rectangle;

// Define the initial viewable area of the TextField instance:
// left: 0, top: 0, width: TextField's width, height: 350 pixels.
bigText.scrollRect = new Rectangle(0, 0, bigText.width, 350);

// Cache the TextField as a bitmap to improve performance.
bigText.cacheAsBitmap = true;

// called when the "up" button is clicked
function scrollUp(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Decrease the y value of the rectangle by 20, effectively
    // shifting the rectangle down by 20 pixels.
    rect.y -= 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

// called when the "down" button is clicked
function scrollDown(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Increase the y value of the rectangle by 20, effectively
    // shifting the rectangle up by 20 pixels.
    rect.y += 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

up.addEventListener(MouseEvent.CLICK, scrollUp);
down.addEventListener(MouseEvent.CLICK, scrollDown);
```

Как показано в этом примере, при работе со свойством `scrollRect` экранного объекта лучше всего указать, что Flash Player или AIR должны кэшировать содержимое объекта в качестве растрового изображения с помощью свойства `cacheAsBitmap`. При этом Flash Player и AIR не требуется повторно отрисовывать все содержимое экранного объекта после каждой прокрутки. Вместо этого можно использовать кэшированное растровое изображение, чтобы визуализировать необходимую часть напрямую на экран. Сведения см. в разделе ««Кэширование экранных объектов» на странице 196».

## Изменение размера и масштабирование объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Измерить размер экранного объекта и манипулировать им можно двумя способами: либо с помощью свойств измерений (`width` и `height`), либо с помощью свойств масштаба (`scaleX` и `scaleY`).

Каждый экранный объект имеет свойства `width` и `height`, которые исходно приравниваются к размеру объекта в пикселах. Значения этих свойств можно считать, чтобы измерить размер экранного объекта. Можно также указать новые значения, чтобы изменить размер объекта следующим образом:

```
// Resize a display object.
square.width = 420;
square.height = 420;

// Determine the radius of a circle display object.
var radius:Number = circle.width / 2;
```

Изменение свойств `height` или `width` экранного объекта приводит к масштабированию объекта. При этом его содержимое растягивается или сжимается, чтобы вместиться в новую область. Если экранный объект содержит только векторные фигуры, то они будут отрисованы с новым масштабом без потери качества. Любые растровые графические элементы в экранном объекте будут отрисованы вместо масштабирования. Итак, например, цифровая фотография, ширина и высота которой увеличиваются, превосходя фактические размеры, заданные для пикселной информации в изображении, будет пикселизирована, а изображение станет зубчатым.

При изменении свойств `width` или `height` экранного объекта Flash Player и AIR обновляют также свойства `scaleX` и `scaleY` объекта.

***Примечание.*** *Объекты TextField являются исключением из этого поведения при масштабировании. Текстовые поля должны изменять свой размер, чтобы вместить перенос текста и размеры шрифта, поэтому после изменения размеров они сбрасывают свои значения scaleX и scaleY до единицы. Однако при изменении значений scaleX или scaleY объекта TextField значения width и height (ширина и высота) также изменяются, чтобы применить заданные значения масштабирования.*

Эти свойства отражают относительный размер экранного объекта в сравнении с его исходным размером. Свойства `scaleX` и `scaleY` используют дробные десятичные значения при расчете процентных значений. Например, если значение `width` экранного объекта изменилось так, что оно составляет половину исходного размера, свойству `scaleX` будет присвоено значение `.5`, равное 50 процентам. В случае удвоения высоты его свойству `scaleY` будет присвоено значение `2`, равное 200 процентам.

```
// circle is a display object whose width and height are 150 pixels.
// At original size, scaleX and scaleY are 1 (100%).
trace(circle.scaleX); // output: 1
trace(circle.scaleY); // output: 1

// When you change the width and height properties,
// Flash Player changes the scaleX and scaleY properties accordingly.
circle.width = 100;
circle.height = 75;
trace(circle.scaleX); // output: 0.6622516556291391
trace(circle.scaleY); // output: 0.4966887417218543
```

Изменения размера непропорциональны. Иными словами, если вы изменяете высоту (`height`) квадрата, не трогая ширину (`width`), его пропорции больше не будут прежними, и вместо квадрата вы получите прямоугольник. Если вы хотите внести относительные изменения в размер экранного объекта, можно установить значения свойств `scaleX` и `scaleY`, чтобы изменить размер объекта вместо установки свойств `width` или `height`. Например, этот код изменяет свойство `width` экранного объекта `square`, а затем изменяет масштаб по вертикали (`scaleY`), приравнивая его к масштабу по горизонтали, чтобы размеры квадрата оставались пропорциональными.

```
// Change the width directly.
square.width = 150;

// Change the vertical scale to match the horizontal scale,
// to keep the size proportional.
square.scaleY = square.scaleX;
```

## Управление искажениями при масштабировании

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Обычно при масштабировании экранного объекта (например, при растягивании по горизонтали) итоговое искажение равномерно распределяется по объекту, и каждая часть растягивается в одинаковой мере. Если речь идет о графических элементах и элементах дизайна, то, возможно, именно это и требуется. Однако иногда более предпочтительно выбирать растягиваемые зоны экранного объекта, оставляя другие области без изменений. Одним из распространенных примеров такой ситуации является прямоугольная кнопка со скругленными углами. В случае обычного масштабирования углы кнопки растягиваются. Из-за этого радиус угла изменяется при изменении масштаба кнопки.

Однако при этом предпочтительно управлять масштабированием, чтобы иметь возможность выделять масштабируемые (прямые стороны и середину) и немасштабируемые области (углы). Это позволяет выполнять масштабирование без появления видимых искажений.

Для создания экранных объектов, масштабированием которых управляет пользователь, можно использовать девятислойное масштабирование (Scale-9). В случае девятислойного масштабирования экранный объект делится на 9 отдельных прямоугольников (сетка 3x3, как в игре «крестики-нолики»). Прямоугольники не обязательно должны быть одного размера: размещением линий сетки управляет сам пользователь. Любое содержимое, находящееся в четырехугольных прямоугольниках (например, скругленные углы кнопки), не будет растягиваться или сжиматься при масштабировании экранного объекта. Верхний и нижний центральные прямоугольники будут масштабироваться по горизонтали, но не по вертикали, а левый средний и правый средний прямоугольники будут масштабироваться по вертикали, но не по горизонтали. Центральный прямоугольник будет масштабироваться как по горизонтали, так и по вертикали.

С учетом этого, если вы создаете экранный объект и хотите избежать масштабирования определенного содержания, достаточно просто убедиться в том, что линии, разделяющие сетку девятислойного масштабирования, размещены так, что содержимое не выходит за пределы одного из прямоугольников.

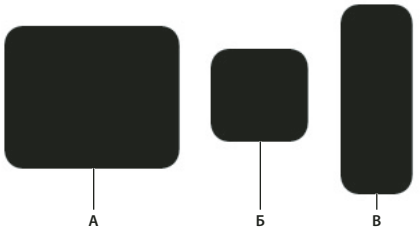
В ActionScript присвоение значения свойству `scale9Grid` экранного объекта приводит к включению девятислойного масштабирования объекта и определяет размер прямоугольников в сетке объекта формата Scale-9. Экземпляр класса Rectangle используется в качестве значения свойства `scale9Grid` следующим образом:

```
myButton.scale9Grid = new Rectangle(32, 27, 71, 64);
```

В конструкторе Rectangle имеется 4 параметра: координата x, координата y, ширина и высота. В данном примере верхний левый угол прямоугольника размещается в точке с координатами (x: 32, y: 27) в экранном объекте с именем `myButton`. Ширина этого прямоугольника составляет 71 пиксел, а высота — 64 пиксела (его правый край имеет координату (x:103), а нижний край — координату (y:92) в экранном объекте).

Фактическая область, которая содержится в пространстве, определенном экземпляром объекта Rectangle, представляет собой центральный прямоугольник сетки Scale-9. Другие прямоугольники рассчитываются Flash Player и AIR путем удлинения сторон экземпляра Rectangle, как показано ниже:

В этом случае при увеличении или уменьшении масштаба кнопки скругленные углы не будут растягиваться или сжиматься, но другие области будут скорректированы в соответствии с масштабом.




***А.*** *myButton.width = 131;myButton.height = 106;* ***Б.*** *myButton.width = 73;myButton.height = 69;* ***В.*** *myButton.width = 54;myButton.height = 141;*

## Кэширование экранных объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

По мере увеличения размера проектов Flash необходимо учитывать производительность и оптимизацию независимо от того, создаете ли вы приложения или сложные анимации на основе сценариев. Если содержимое остается статическим (как экземпляр прямоугольного объекта Shape), проигрыватель Flash Player и среда AIR не оптимизируют его. Поэтому при изменении позиции прямоугольника Flash Player и AIR отрисовывают весь экземпляр Shape заново.

Вы можете поместить в кэш указанные экранные объекты, чтобы повысить производительность своего SWF-файла. Экранный объект представляет собой *подложку*. Это обычно растровая версия векторных данных экземпляра, которые не планируется сильно изменять в пределах SWF-файла. Следовательно, экземпляры с включенным кэшированием не перерисовываются постоянно по мере воспроизведения SWF-файла, что позволяет быстро визуализировать этот файл.

***Примечание.*** *Векторные данные можно обновить. При этом подложка будет воссоздана. Исходя из этого, векторные данные, которые кэшированы в подложке, не должны оставаться неизменными для всего SWF-файла.*

Присвоение свойству `cacheAsBitmap` экранного объекта значения `true` приводит к тому, что объект кэширует собственное растровое представление. Проигрыватель Flash Player и среда AIR создают для экземпляра объект подложки, который представляет собой кэшированное растровое изображение, а не векторные данные. При изменении границ экранного объекта выполняется воссоздание подложки вместо изменения размера. Подложки могут быть вложены в другие подложки. Дочерняя подложка копирует свое растровое изображение на родительскую подложку. Дополнительные сведения см. в разделе ««Включение кэширования растрового изображения» на странице 199».

Свойства `opaqueBackground` и `scrollRect` класса DisplayObject относятся к кэшированию растрового изображения с помощью свойства `cacheAsBitmap`. Хотя три этих свойства независимы друг от друга, свойства `opaqueBackground` и `scrollRect` лучше всего функционируют, когда объект кэширован в качестве растрового изображения. Выигрыш в производительности при использовании свойств `opaqueBackground` и `scrollRect` виден только тогда, когда `cacheAsBitmap` присвоено значение `true`. Дополнительные сведения о прокрутке содержимого экранного объекта см. в разделе ««Панорамирование и прокрутка экранных объектов» на странице 192». Дополнительные сведения об установке непрозрачного фона см. в разделе ««Установка цвета непрозрачного фона» на странице 200».

Сведения о маскировании альфа-канала, которое требует присвоения свойству `cacheAsBitmap` значения `true`, см. в разделе ««Маскирование экранных объектов» на странице 205».

## Когда следует применять кэширование

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При включении кэширования для экранного объекта создается подложка, которая обладает несколькими преимуществами (например, позволяет быстрее визуализировать сложную векторную анимацию). Существует ряд сценариев, требующих включения кэширования. Может показаться, что включать кэширование следует всегда, когда требуется повысить производительность своих SWF-файлов. Однако существуют ситуации, при которых включение кэширования не приводит к повышению производительности или даже вызывает ее снижение. В данном разделе описываются сценарии, в которых необходимо применять кэширование, и случаи использования обычных экранных объектов.

Общая производительность кэшированных данных зависит от сложности векторных данных в экземплярах, от объема измененных данных и от того, включено ли свойство `opaqueBackground`. При изменении небольших регионов разница между использованием подложки и векторных данных может оказаться незначительной. Перед развертыванием приложения может быть полезно проверить оба сценария со своей работой.

### Случаи применения кэширования растровых изображений

Ниже приведены типичные сценарии, в которых кэширование растровых изображений может оказаться очень выгодным.

- Изображение со сложным фоном: приложение, содержащее векторное изображение со сложным и детально проработанным фоном (возможно, изображение, к которому применена команда trace bitmap, или работа, созданная в Adobe Illustrator®). Можно анимировать фоновые символы. Это замедлит анимацию, поскольку для фона необходимо постоянно воссоздавать векторные данные. Чтобы повысить производительность, можно присвоить свойству `opaqueBackground` фонового экранного объекта значение `true`. Фон визуализируется как растровое изображение и поддерживает быструю перерисовку, что повышает скорость воспроизведения анимации.

- Текстовое поле с прокруткой: приложение, отображающее большой объем текста в текстовом поле с прокруткой. Текстовое поле можно разместить в экранном объекте, для которого задана возможность прокрутки с помощью границ прокрутки (свойство `scrollRect`). Это позволяет быстро прокручивать пикселы в указанном экземпляре. При прокрутке экземпляра экранного объекта все текстовое поле не формируется заново. Вместо этого прокручиваемые пикселы сдвигаются вверх, после чего формируется новая область.
- Система управления окнами: приложение со сложной системой перекрывающихся окон. Каждое окно можно открыть или закрыть (например, окна веб-обозревателя). Если каждое окно пометить как подложку (задав для свойства `cacheAsBitmap` значение `true`), то все помеченные окна будут изолированными и будут кэшироваться. Пользователи могут перетаскивать окна, перекрывая ими друг друга, при этом векторное содержимое в каждом окне не приходится формировать заново.
- Маскирование альфа-канала: при использовании маскирования альфа-канал необходимо присвоить свойству `cacheAsBitmap` значение `true`. Дополнительные сведения см. в разделе ««Маскирование экранных объектов» на странице 205».

Включение кэширования растровых изображений во всех этих сценариях улучшает способность к реагированию и интерактивность приложения путем оптимизации векторной графики.

Кроме того, каждый раз при применении фильтра к экранному объекту `cacheAsBitmap` автоматически присваивается значение `true` даже в том случае, если вы явным образом выберете значение `false`. Если вы удалите все фильтры из экранного объекта, свойству `cacheAsBitmap` будет назначено последнее из присвоенных значений.

**Когда не следует применять кэширование растровых изображений**

Использование данной функции при неверных обстоятельствах может негативно сказаться на производительности SWF-файла. При использовании кэширования растровых изображений запомните следующие рекомендации.

- Не используйте подложки (экранные объекты с включенным кэшированием) больше, чем это необходимо. Любая подложка требует больше памяти, чем обычный экранный объект. Это означает, что подложки необходимо применять только тогда, когда нужно повысить производительность визуализации.

  Кэшированные растровые изображения могут занимать значительно больше памяти, чем обычный экранный объект. Например, если размер экземпляра Sprite в рабочей области составляет 250x250 пикселов, то кэширование может потребовать 250 КБ вместо 1 КБ в случае обычного (не помещенного в кэш) экземпляра Sprite.
- Избегайте увеличения кэшированных подложек. Если кэширование растровых изображений используется чрезмерно активно, то потребляется большой объем памяти (см. предыдущий пункт), особенно при увеличении масштаба.
- Подложки следует использовать для тех экземпляров экранных объектов, которые, в основном, статичны (без анимации). Такой экземпляр можно перетаскивать или перемещать, но его содержимое не должно быть анимированным или значительно изменяться. Анимация или изменение содержимого более вероятно в случае экземпляра MovieClip с анимацией или экземпляра Video. Например, если экземпляр вращается или преобразуется, то используется попеременно подложка и векторные данные, что затрудняет расчеты и негативно влияет на работу SWF-файла.
- Если подложки используются вперемешку с векторными данными, то возрастает объем вычислений, выполняемых приложениями Flash Player и AIR (а иногда и компьютером). Группируйте подложки везде, где это возможно (например, при создании приложений с окнами).

- Не кэшируйте объекты, графика которых часто изменяется. Каждый раз при изменении масштаба, вращении отображаемого объекта, изменении свойства alpha, преобразовании цвета, перемещении дочерних отображаемых объектов и рисовании с использованием свойства графики перерисовывается кэшированное растровое изображение. Если это происходит в каждом кадре, среда выполнения должна добавить объект в растровое изображение, а затем скопировать это растровое изображение в рабочую область, что приводит к повышению объема работы по сравнению с рисованием некэшированного объекта в рабочей области. Соотношение между оптимизацией производительности при кэшировании и частотой обновления зависит от сложности и размера отображаемого объекта и может быть определено только путем тестирования конкретного содержимого.

## Включение кэширования растрового изображения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы включить кэширование растрового изображения для экранного объекта, свойству `cacheAsBitmap` необходимо присвоить значение `true`.

```
mySprite.cacheAsBitmap = true;
```

После присвоения свойству `cacheAsBitmap` значения `true` можно заметить, что экранный объект автоматически привязывает свои пикселы к целым значениям координат. В ходе проверки SWF-файла видно, что сложные анимированные векторные изображения визуализируются гораздо быстрее.

Подложка (кэшированное растровое изображение) не создается даже в том случае, когда `cacheAsBitmap` присваивается значение `true`, при наличии одного или нескольких указанных ниже условий.

- Размер растрового изображения превышает 2880 в ширину или в высоту.
- Не удается найти растровое изображение (ошибка «Недостаточно памяти»).

### Матрицы преобразования кэшированных растровых изображений

**Adobe AIR 2.0 и более поздние версии (мобильный профиль)**

В приложениях AIR для мобильных устройств следует устанавливать свойство `cacheAsBitmapMatrix` во всех случаях, когда задается свойство `cacheAsBitmap`. Установка этого свойства позволит применять более широкий набор преобразований для отображаемых объектов без повторной визуализации.

```
mySprite.cacheAsBitmap = true;
mySprite.cacheAsBitmapMatrix = new Matrix();
```

Если установлено свойство матрицы, к отображаемому объекту можно будет применять дополнительное преобразование без повторного кэширования объекта.

- Перемещение и перенос без привязки к пикселам
- Вращать
- Масштаб
- Наклон
- Изменить значение альфа (от 0 до 100 % прозрачности)

Эти преобразования применяются к кэшированному растровому изображению.

## Установка цвета непрозрачного фона

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Вы можете установить непрозрачный фон для экранного объекта. Например, когда у SWF-файла есть фон, содержащий сложную векторную графику, свойству `opaqueBackground` можно присвоить заданный цвет (обычно выбирается цвет рабочей области). Цвет выражается числом (обычно это шестнадцатеричное значение цвета). Затем этот фон воспринимается как растровое изображение, что помогает добиться оптимальной производительности.

Когда `cacheAsBitmap` присваивается значение `true`, а свойству `opaqueBackground` назначается заданный цвет, свойство `opaqueBackground` дает внутреннему растровому изображению возможность стать непрозрачным и быстрее визуализироваться. Если `cacheAsBitmap` не присвоить значение `true`, свойство `opaqueBackground` добавляет непрозрачную векторную квадратную фигуру к фону экранного объекта. При этом растровое изображение не создается автоматически.

В следующем примере показан метод установки фона экранного объекта для оптимизации производительности:

```
myShape.cacheAsBitmap = true;
myShape.opaqueBackground = 0xFF0000;
```

В данном случае цветом фона объекта Shape с именем `myShape` является красный (`0xFF0000`). Предположим, что экземпляр Shape содержит рисунок с зеленым треугольником в рабочей области с белым фоном. На экране это будет выглядеть как зеленый треугольник в красном незаполненном пространстве ограничивающего поля (прямоугольника, полностью охватывающего экземпляр Shape).

Конечно, этот код имел бы больше смысла при использовании в рабочей области со сплошным красным фоном. При выборе фона другого цвета этот цвет необходимо было бы указать. Например, в SWF-файле с белым фоном свойству `opaqueBackground`, скорее всего, был бы присвоен цвет `0xFFFFFF` или чистый белый оттенок.

## Применение режимов смешивания

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Режимы смешивания предполагают сочетание цветов одного изображения (базовое изображение) с цветами другого изображения (изображение смешивания), приводящее к появлению третьего итогового изображения, которое появляется на экране. Каждое значение пиксела на изображении обрабатывается с учетом соответствующего пикселного значения другого изображения. В результате получается значение пиксела для идентичной позиции.

У каждого экранного объекта существует свойство `blendMode`, для которого можно задать один из указанных ниже режимов смешивания. Эти константы определены в классе BlendMode. Кроме того, можно воспользоваться строковыми значениями (в скобках), которые являются фактическими значениями констант.

- `BlendMode.ADD` (`"add"`): обычно используется для создания анимированного осветляющего эффекта растворения двух изображений.
- `BlendMode.ALPHA` (`"alpha"`): обычно используется для применения прозрачности переднего плана к фону. (Не поддерживается при визуализации с использованием графического процессора.)
- `BlendMode.DARKEN` (`"darken"`): обычно применяется для наложения типа. (Не поддерживается при визуализации с использованием графического процессора.)
- `BlendMode.DIFFERENCE` (`"difference"`): обычно используется для оживления цветов.
- `BlendMode.ERASE` (`"erase"`): как правило, применяется для вырезания (стирания) части переднего плана с использованием альфа-канала переднего плана. (Не поддерживается при визуализации с использованием графического процессора.)
- `BlendMode.HARDLIGHT` (`"hardlight"`): обычно применяется для создания эффектов затенения. (Не поддерживается при визуализации с использованием графического процессора.)
- `BlendMode.INVERT` (`"invert"`): используется для инверсии фона.
- `BlendMode.LAYER` (`"layer"`): применяется для принудительного создания временного буфера с целью предварительного формирования определенного экранного объекта. (Не поддерживается при визуализации с использованием графического процессора.)
- `BlendMode.LIGHTEN` (`"lighten"`): обычно используется для наложения типа. (Не поддерживается при визуализации с использованием графического процессора.)
- `BlendMode.MULTIPLY` (`"multiply"`): обычно применяется для создания теней и эффектов глубины.
- `BlendMode.NORMAL` (`"normal"`): используется для того, чтобы указать, что значения пикселов изображения смешивания имеют приоритет перед соответствующими значениями базового изображения.
- `BlendMode.OVERLAY` (`"overlay"`): обычно используется для создания эффектов тени. (Не поддерживается при визуализации с использованием графического процессора.)
- `BlendMode.SCREEN` (`"screen"`): обычно используется для создания подсветки и бликов объектива.
- `BlendMode.SHADER` (`"shader"`): используется для того, чтобы указать, что шейдер Pixel Bender используется для создания заказного эффекта смешивания. Дополнительные сведения об использовании шейдеров см. в разделе ««Работа с шейдерами Pixel Bender» на странице 324». (Не поддерживается при визуализации с использованием графического процессора.)
- `BlendMode.SUBTRACT` (`"subtract"`): обычно используется для создания анимированного затемняющего эффекта растворения двух изображений.

## Корректировка цветов DisplayObject

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для корректировки цвета экранного объекта можно воспользоваться методами класса ColorTransform (flash.geom.ColorTransform). У каждого экранного объекта есть свойство `transform`, которое является экземпляром класса Transform и содержит сведения о различных преобразованиях, применяемых к экранному объекту (например, вращение, изменения масштаба и положения и т. д.) Помимо информации о геометрических преобразованиях класс Transform включает также свойство `colorTransform`, которое является экземпляром класса ColorTransform и обеспечивает доступ для внесения цветовых корректировок в экранный объект. Чтобы перейти к сведениям о цветовых преобразованиях экранного объекта, можно воспользоваться следующим кодом:

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;
```

После создания экземпляра ColorTransform можно считать его значения свойств, чтобы выяснить, какие преобразования цветов уже применены, либо задать эти значения, чтобы внести цветовые корректировки в экранный объект. Чтобы обновить экранный объект после внесения любых изменений, можно повторно назначить экземпляр ColorTransform свойству `transform.colorTransform`.

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;

// Make some color transformations here.

// Commit the change.
myDisplayObject.transform.colorTransform = colorInfo;
```

### Установка значений цвета с помощью кода

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Свойство `color` класса ColorTransform можно использовать для присвоения экранному объекту значения цвета RGB (красный, зеленый, синий). Свойство `color` используется в следующем примере для замены цвета экранного объекта `square` синим, когда пользователь нажимает кнопку `blueBtn`:

```
// square is a display object on the Stage.
// blueBtn, redBtn, greenBtn, and blackBtn are buttons on the Stage.

import flash.events.MouseEvent;
import flash.geom.ColorTransform;

// Get access to the ColorTransform instance associated with square.
var colorInfo:ColorTransform = square.transform.colorTransform;

// This function is called when blueBtn is clicked.
function makeBlue(event:MouseEvent):void
{
    // Set the color of the ColorTransform object.
    colorInfo.color = 0x003399;
    // apply the change to the display object
    square.transform.colorTransform = colorInfo;
}

blueBtn.addEventListener(MouseEvent.CLICK, makeBlue);
```

Учтите, что при изменении цвета экранного объекта с помощью свойства `color` полностью меняется цвет всего объекта, независимо от того, что раньше его окраска состояла из нескольких цветов. Например, при наличии экранного объекта с зеленым кругом, в верхней части которого есть черный текст, выбор красного оттенка с помощью свойства `color` для экземпляра ColorTransform этого объекта приведет к заполнению всего объекта (круга и текста) красным цветом (при этом текст станет неотличим от остального содержимого объекта).

## Изменение эффектов цвета и яркости с помощью кода

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Предположим, что имеется экранный объект с несколькими цветами (например, цифровой снимок), и полностью перекрашивать объект не требуется. Нужно просто скорректировать цвет экранного объекта, исходя из имеющихся цветов. В данном сценарии класс ColorTransform включает ряд свойств умножителя и смещения, которые можно использовать для выполнения этого типа коррекции. Свойства умножителя `redMultiplier`, `greenMultiplier`, `blueMultiplier` и `alphaMultiplier` действуют как цветные фотофильтры (или окрашенные солнечные очки), усиливая или уменьшая выраженность определенных цветов в экранном объекте. Свойства смещения (`redOffset`, `greenOffset`, `blueOffset` и `alphaOffset`) можно использовать для добавления в объект дополнительных порций того или иного цвета или для указания минимально допустимого значения определенного цвета.

Эти свойства умножителя и смещения идентичны дополнительным установкам цвета, доступным для символов фрагмента ролика в инструменте разработки Flash. К ним можно перейти, выбрав «Дополнительно» во всплывающем меню «Цвет» инспектора свойств.

Следующий код загружает изображение JPEG и применяет к нему преобразование цвета, которое корректирует красный и зеленый каналы по мере перемещения курсора вдоль осей X и Y. В этом случае из-за того, что значения смещения не заданы, значение цвета каждого канала цвета, отображенного на экране, будет выражаться в процентах исходного значения цвета в изображении. Это означает, что по большей части красный и зеленый цвета в любом заданном пикселе будут основаны на исходной доле красного и зеленого цвета в пикселе.

```
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.geom.Transform;
import flash.geom.ColorTransform;
import flash.net.URLRequest;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// This function is called when the mouse moves over the loaded image.
function adjustColor(event:MouseEvent):void
{
    // Access the ColorTransform object for the Loader (containing the image)
    var colorTransformer:ColorTransform = loader.transform.colorTransform;

    // Set the red and green multipliers according to the mouse position.
    // The red value ranges from 0% (no red) when the cursor is at the left
    // to 100% red (normal image appearance) when the cursor is at the right.
    // The same applies to the green channel, except it's controlled by the
    // position of the mouse in the y axis.
    colorTransformer.redMultiplier = (loader.mouseX / loader.width) * 1;
    colorTransformer.greenMultiplier = (loader.mouseY / loader.height) * 1;

    // Apply the changes to the display object.
    loader.transform.colorTransform = colorTransformer;
}

loader.addEventListener(MouseEvent.MOUSE_MOVE, adjustColor);
```

## Поворот объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Экранные объекты можно вращать с помощью свойства `rotation`. Это значение можно считать, чтобы выяснить, был ли повернут объект. А чтобы повернуть объект, можно присвоить этому свойству число (в градусах), отражающее угол поворота объекта. Так, к примеру, следующая строка кода приводит к повороту объекта `square` на 45 градусов (одна восьмая от полного оборота):

```
square.rotation = 45;
```

Можно также повернуть объект с помощью матрицы преобразования, описанной в разделе «Работа с геометрией» на странице 227.

## Снижение яркости объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Можно управлять прозрачностью экранного объекта, чтобы сделать его частично или полностью прозрачным, либо изменить прозрачность таким образом, чтобы казалось, что яркость объекта снижается или повышается. Свойство `alpha` класса DisplayObject определяет прозрачность (точнее говоря, непрозрачность) экранного объекта. Свойству `alpha` можно присвоить любое значение от 0 до 1, где 0 обозначает полную прозрачность, а 1 — абсолютную непрозрачность. Например, следующие строки кода приводят к тому, что объект `myBall` становится наполовину (50 %) прозрачным при щелчке мышью:

```
function fadeBall(event:MouseEvent):void
{
    myBall.alpha = .5;
}
myBall.addEventListener(MouseEvent.CLICK, fadeBall);
```

Кроме того, можно изменить прозрачность экранного объекта с помощью корректировок цвета, доступных для класса ColorTransform. Дополнительные сведения см. в разделе ««Корректировка цветов DisplayObject» на странице 202».

## Маскирование экранных объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Экранный объект можно применить в качестве маски, чтобы создать отверстие, сквозь которое видно содержимое другого экранного объекта.

### Определение маски

Чтобы указать, что экранный объект будет маской для другого экранного объекта, сделайте маскирующий объект свойством `mask` маскируемого экранного объекта:

```
// Make the object maskSprite be a mask for the object mySprite.
mySprite.mask = maskSprite;
```

Маскированный экранный объект находится под всеми непрозрачными областями экранного объекта, выполняющего роль маски. Например, следующий код создает экземпляр Shape, содержащий квадрат размером 100x100 пикселов, и экземпляр Sprite, содержащий синий круг с радиусом 25 пикселов. При нажатии на круг он задается в качестве маски для квадрата так, что видна лишь та часть квадрата, которая закрыта сплошной частью круга. Иными словами, виден будет только красный круг.

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.Shape;

// Draw a square and add it to the display list.
var square:Shape = new Shape();
square.graphics.lineStyle(1, 0x000000);
square.graphics.beginFill(0xff0000);
square.graphics.drawRect(0, 0, 100, 100);
square.graphics.endFill();
this.addChild(square);

// Draw a circle and add it to the display list.
var circle:Sprite = new Sprite();
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0x0000ff);
circle.graphics.drawCircle(25, 25, 25);
circle.graphics.endFill();
this.addChild(circle);

function maskSquare(event:MouseEvent):void
{
    square.mask = circle;
    circle.removeEventListener(MouseEvent.CLICK, maskSquare);
}

circle.addEventListener(MouseEvent.CLICK, maskSquare);
```

Экранный объект, выступающий в роли маски, можно перетаскивать и анимировать, а также динамически изменять его размер. Внутри одной маски могут использоваться отдельные фигуры. Маскирующий экранный объект не обязательно добавлять в список отображения. Однако если требуется, чтобы масштаб маскирующего объекта изменялся при изменении масштаба рабочей области, или если необходимо включить возможность взаимодействия пользователя с маской (например, управляемое пользователем перетаскивание и изменение размера), маскирующий объект необходимо добавить в список отображения. Фактический z-индекс (порядок расположения слоев) экранных объектов не играет никакой роли, если маскирующий объект добавлен в список отображения. Маскирующий объект будет появляться на экране только в виде маски. Если маскирующий объект является экземпляром MovieClip с несколькими кадрами, он воспроизводит все кадры на временной шкале. То же самое происходит и тогда, когда он не выступает в роли маски. Удалить маску можно, присвоив свойству `mask` значение `null`:

```
// remove the mask from mySprite
mySprite.mask = null;
```

Маску нельзя применять для маскирования другой маски. Невозможно задать свойство `alpha` маскирующего экранного объекта. В экранном объекте, выбранного в качестве маски, используются только заливки; обводки игнорируются.

**AIR 2**

Если экранный маскируемый объект кэшируется путем установки свойств `cacheAsBitmap` и `cacheAsBitmapMatrix`, маска должна быть дочерней по отношению к этому объекту. Аналогичным образом, если маскируемый экранный объект является потомком контейнера экранных объектов, которые кэшируются, маска и экранный объект должны быть потомками этого контейнера. Если маскируемый объект является потомком контейнера нескольких кэшированных экранных объектов, маска должна быть потомком кэшированного контейнера, ближайшего к маскируемому объекту в списке отображения.

---

## О маскировании шрифтов устройства

Экранный объект можно использовать в целях маскирования текста, набранного шрифтом устройства. Когда экранный объект используется для маскирования текста, набранного шрифтом устройства, прямоугольное ограничивающее поле маски используется в качестве маскирующей фигуры. Иными словами, если в среде разработчика создать непрямоугольную маску экранного объекта для текста со шрифтом устройства, то маска в SWF-файле приобретает форму прямоугольной ограничительной рамки маски, а не форму самой маски.

## Маскирование альфа-канала

Маскирование альфа-канала поддерживается, если и маска, и маскированные экранные объекты используют кэширование растровых изображений, как показано ниже.

```
// maskShape is a Shape instance which includes a gradient fill.
mySprite.cacheAsBitmap = true;
maskShape.cacheAsBitmap = true;
mySprite.mask = maskShape;
```

Например, маскирование альфа-канала можно применять для применения фильтра к маскирующему объекту независимо от фильтра, который применен к маскированному экранному объекту.

В следующем примере в рабочую область загружен внешний файл с изображением. Это изображение (точнее говоря, экземпляр Loader, в который оно загружено) становится маскированным экранным объектом. Овал с градиентом (центр, окрашенный сплошным черным цветом, который переходит к прозрачному по краям) нарисован поверх изображения; это маска альфа-канала. У обоих экранных объектов отключено кэширование растровых изображений. Овал задан в качестве маски для изображения, и это придает ему способность к перетаскиванию.

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Sprite;
import flash.geom.Matrix;
import flash.net.URLRequest;

// Load an image and add it to the display list.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create a Sprite.
var oval:Sprite = new Sprite();
// Draw a gradient oval.
var colors:Array = [0x000000, 0x000000];
var alphas:Array = [1, 0];
var ratios:Array = [0, 255];
var matrix:Matrix = new Matrix();
matrix.createGradientBox(200, 100, 0, -100, -50);
oval.graphics.beginGradientFill(GradientType.RADIAL,
                                colors,
                                alphas,
                                ratios,
                                matrix);
oval.graphics.drawEllipse(-100, -50, 200, 100);
oval.graphics.endFill();
// add the Sprite to the display list
this.addChild(oval);

// Set cacheAsBitmap = true for both display objects.
loader.cacheAsBitmap = true;
oval.cacheAsBitmap = true;
// Set the oval as the mask for the loader (and its child, the loaded image)
loader.mask = oval;

// Make the oval draggable.
oval.startDrag(true);
```

# Анимация объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Анимация — это процесс приведения объектов в движение или изменение объектов с течением времени. Анимация на основе сценариев является основополагающей частью видеоигр. Она часто применяется при добавлении удобных и полезных меток интерактивности в другие приложения.

В основе анимации на основе сценариев лежит идея о том, что необходимо внести изменение, которое следует разделить на временные фрагменты. В ActionScript легко добиться повторения с помощью обычной инструкции повтора. Однако перед обновлением экрана цикл должен будет повториться заданное число раз. Чтобы создать анимацию на основе сценариев, необходимо создать код ActionScript, в котором определенное действие повторяется с течением времени и при каждом повторении происходит обновление экрана.

Например, представьте, что необходимо создать простую анимацию (например, запустить по экрану мяч). ActionScript включает простой механизм, позволяющий следить за ходом времени и соответствующим образом обновлять экран. Это значит, что можно составить код, который передвигает мяч каждый раз на небольшое расстояние до тех пор, пока он не достигнет пункта назначения. Экран обновляется после завершения каждого движения, позволяя зрителю увидеть перемещение мяча по рабочей области.

С практической точки зрения имеет смысл синхронизировать частоту кадров в анимации на основе сценариев и SWF-файла (иными словами, заставить анимацию меняться с каждым новым кадром), поскольку это определяет частоту обновления экрана в AIR или проигрывателе Flash Player. У каждого экранного объекта существует событие `enterFrame`, которое отправляется в соответствии с частотой кадров SWF-файла (по одному событию на кадр). Большинство разработчиков, создающих анимацию на основе сценариев, используют событие `enterFrame` в качестве средства для создания повторяющихся действий. Можно написать код, который ожидает появления события `enterFrame` и передвигает анимированный мяч на определенное расстояние в каждом кадре. По мере обновления экрана (в каждом кадре) этот мяч будет отрисовываться на новой позиции, что вызывает эффект движения.

***Примечание.*** *Другим методом выполнения повторяющегося действия является использование класса Timer. Экземпляр Timer инициирует уведомление о событии каждый раз после истечения определенного промежутка времени. Можно написать код, определяющий анимацию путем обработки события таймера в классе Timer. При этом необходимо выбрать маленький промежуток времени (долю секунды). Дополнительные сведения об использовании класса Timer см. в разделе ««Управление временными интервалами» на странице 4».*

В следующем примере в рабочей области создается круговой экземпляр Sprite под именем `circle`. Когда пользователь щелкает по кругу, начинает выполняться последовательность из анимации на основе сценария. Благодаря этому яркость `circle` снижается (значение свойства `alpha` уменьшается) до тех пор, пока он не станет совсем прозрачным:

```
import flash.display.Sprite;
import flash.events.Event;
import flash.events.MouseEvent;

// draw a circle and add it to the display list
var circle:Sprite = new Sprite();
circle.graphics.beginFill(0x990000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();
addChild(circle);

// When this animation starts, this function is called every frame.
// The change made by this function (updated to the screen every
// frame) is what causes the animation to occur.
function fadeCircle(event:Event):void
{
    circle.alpha -= .05;

    if (circle.alpha <= 0)
    {
        circle.removeEventListener(Event.ENTER_FRAME, fadeCircle);
    }
}

function startAnimation(event:MouseEvent):void
{
    circle.addEventListener(Event.ENTER_FRAME, fadeCircle);
}

circle.addEventListener(MouseEvent.CLICK, startAnimation);
```

Когда пользователь щелкает круг, функция `fadeCircle()` регистрируется в качестве прослушивателя события `enterFrame`. Это означает, что она вызывается один раз за кадр. Эта функция снижает яркость `circle` путем изменения его свойства `alpha`. Один раз за кадр свойство `alpha` круга уменьшается на 0,05 (5 %), а экран обновляется. В итоге, когда значение `alpha` доходит до 0 (`circle` полностью прозрачен), функция `fadeCircle()`, играющая роль прослушивателя событий, удаляется, и анимация завершается.

Тот же самый код можно использовать, например, для создания анимированного движения вместо ослабления яркости. Путем замены свойства `alpha` другим свойством в функции, которая является прослушивателем события `enterFrame`, это свойство будет анимировано вместо нее. Например, замена строки

```
    circle.alpha -= .05;
```

кодом

```
    circle.x += 5;
```

приведет к анимации свойства `x`, что приведет к перемещению круга вправо через рабочую область. Условие, завершающее анимацию, можно изменить так, чтобы она завершалась (то есть, чтобы отменить регистрацию прослушивателя `enterFrame`) при достижении нужной координаты по оси x.

# Ориентация рабочей области

**AIR 2.0 и более новые версии**

Мобильные устройства обычно изменяют ориентацию пользовательского интерфейса, чтобы экран оставался в вертикальном положении, когда пользователь поворачивает устройство. Если в приложении включена автоматическая ориентация, устройство поддерживает правильную ориентацию экрана, но при этом нужно убедиться, что содержимое отображается нормально при изменении соотношения сторон рабочей области. Если автоматическая ориентация отключена, экран устройства сохраняет фиксированную ориентацию до тех пор, пока ориентация не будет изменена вручную.

Приложения AIR работают на различных мобильных устройствах и в операционных системах. Поведение ориентации может отличаться для разных операционных систем и даже для разных устройств с одинаковой операционной системой. Простая стратегия, хорошо работающая на всех устройствах и операционных системах, заключается в том, чтобы включить автоматическую ориентацию и прослушивать события `resize` для рабочей области, чтобы определить, когда требуется обновить макет приложения.

В качестве альтернативного решения, если установлена поддержка соотношения сторон только для книжной или только для альбомной ориентации, автоматическую ориентацию можно отключить и задать поддерживаемое соотношение сторон в дескрипторе приложения AIR. Данная стратегия разработки позволяет добиться предсказуемого поведения и выбрать «лучшую» ориентацию для заданного соотношения сторон. Например, если задать альбомное соотношение сторон, выбранная ориентация подходит для устройств с выдвигающейся клавиатурой в альбомном режиме.

## Получение текущей ориентации и соотношения сторон рабочей области

Ориентация указывается относительно нормального положения устройства. Для большинства устройств имеется четкое вертикальное положение. Данное положение считается ориентацией *по умолчанию*. Три других возможных ориентации включают: *повернуто влево*, *повернуто вправо* и *перевернуто*. Класс StageOrientation определяет строковые константы, которые применяются при настройке и сравнении значений.

Класс Stage определяет два свойства, которые позволяют получить сведения об ориентации:

- Stage.deviceOrientation — указывает физическую ориентацию устройства относительно положения по умолчанию.

  ***Примечание.*** *Свойство deviceOrientation не всегда доступно при первом запуске приложения или когда устройство находится в горизонтальном положении. В этих случаях сообщается, что ориентация неизвестна.*

- Stage.orientation — указывает ориентацию рабочей области относительно положения по умолчанию. Если автоматическая ориентация включена, рабочая область поворачивается в противоположном направлении, как если бы устройство оставалось в вертикальном положении. Таким образом, положения с поворотом вправо и влево, указываемые свойством `orientation`, противоположны положениям, о которых сообщает свойство `deviceOrientation`. Например, когда свойство `deviceRotation` указывает ориентацию *повернуто вправо*, свойство `orientation` указывает ориентацию *повернуто влево*.

Соотношение сторон рабочей области может быть получено путем сравнения текущей ширины и высоты рабочей области:

```
var aspect:String = this.stage.stageWidth >= this.stage.stageHeight ?
StageAspectRatio.LANDSCAPE : StageAspectRatio.PORTRAIT;
```

## Автоматическая ориентация

Если автоматическая ориентация включена и пользователь поворачивает устройство, операционная система изменяет ориентацию всего интерфейса, включая панель задач и окно вашего приложения. В результате этого соотношение сторон рабочей области меняется с книжного на альбомное или наоборот. При изменении соотношения сторон также изменяются размеры рабочей области.

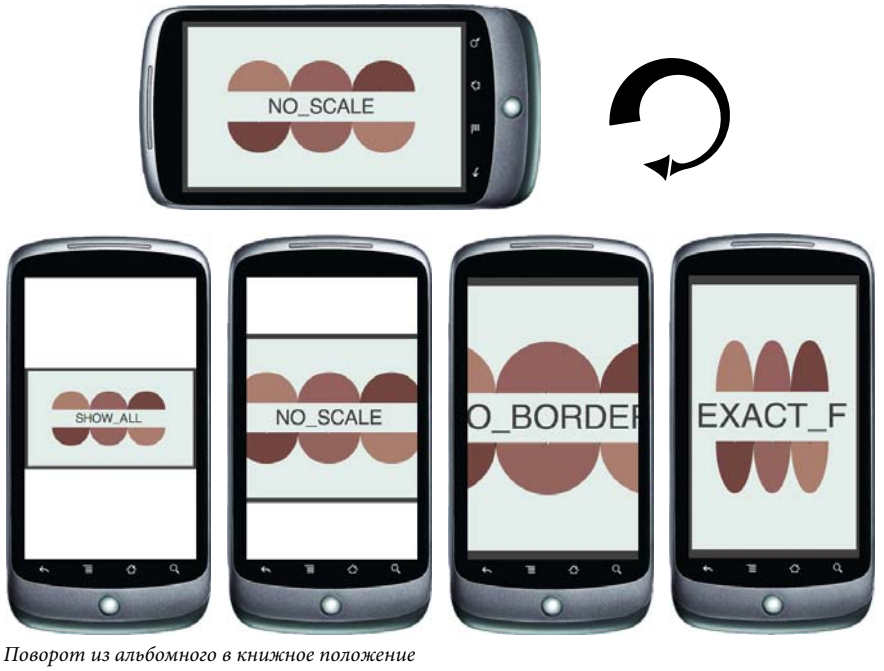
Для включения и отключения автоматической ориентации во время выполнения установите для свойства рабочей области `autoOrients` значение `true` или `false`. Можно задать исходное значение для этого свойства, определив в дескрипторе приложения AIR элемент `<autoOrients>`. (обратите внимание, что до версии AIR 2.6 свойство `autoOrients` доступно только для чтения и его можно установить только в дескрипторе приложения).

Если указаны пропорции альбомной или книжной ориентации и включен параметр автоматической установки ориентации, в среде AIR автоматическая ориентация устанавливается только в пределах указанных пропорций.

### Изменение размеров рабочей области

При изменении размера рабочей области изменяется масштаб и положение содержимого в соответствии со значениями свойств `scaleMode` и `align` объекта Stage. В большинстве случаев использование автоматического поведения настроек рабочей области `scaleMode` не дает хороших результатов. Вместо этого следует выполнять перекомпоновку или перерисовку графики и компонентов, чтобы обеспечить поддержку более чем одного соотношения сторон (реализация гибкой схемы компоновки также улучшит качество работы приложения на разных устройствах с разными размерами экранов и соотношением сторон).

На следующей иллюстрации показаны эффекты различных настроек `scaleMode` при вращении стандартного мобильного устройства.




*Поворот из альбомного в книжное положение*

На иллюстрации показано поведение масштабирования при повороте устройства из альбомного в книжное положение в разных режимах масштабирования. Поворот из книжного положения в альбомное дает аналогичные эффекты.

**События изменения ориентации**

Объект Stage отправляет два типа событий, которые можно использовать, чтобы обнаруживать изменения ориентации и реагировать на них. Оба события рабочей области `resize` и `orientationChange` отправляются, когда автоматическая ориентация включена.

Событие *resize* лучше всего использовать, когда для сохранения вертикального расположения экрана применяется функция автоматической ориентации. Когда рабочая область отправляет событие `resize`, выполните перекомпоновку или перерисовку содержимого. Событие `resize` отправляется, только когда установлен режим масштабирования `noScale`.

Событие `orientationChange` также можно использовать для выявления изменений ориентации. Событие `orientationChange` отправляется, только когда включена автоматическая ориентация.

***Примечание.*** *На некоторых мобильных платформах рабочая область отправляет событие `orientationChanging` перед отправкой событий resize и orientationChange. Поскольку событие поддерживается не на всех платформах, его использования следует избегать.*

## Ручная ориентация

**AIR 2.6 и более новые версии**

Управлять ориентацией рабочей области можно с помощью метода `setOrientation()` или `setAspectRatio()` объекта Stage.

**Установка ориентации рабочей области**

Ориентацию рабочей области можно задать во время выполнения с помощью метода`setOrientation()` объекта Stage. Чтобы выбрать нужную ориентацию используйте строковые константы, определенные в классе StageOrientation.

```
this.stage.setOrientation( StageOrientation.ROTATED_RIGHT );
```

Не все устройства и операционные системы поддерживают все возможные ориентации. Например, Android 2.2 не поддерживает программный выбор ориентации с поворотом влево на устройствах со стандартной книжной ориентацией и вообще не поддерживает ориентацию вверх ногами. Свойство `supportedOrientations` рабочей области содержит список вариантов ориентации, которые могут быть переданы методу `setOrientation()`:

```
var orientations:Vector.<String> = this.stage.supportedOrientations;
for each( var orientation:String in orientations )
{
    trace( orientation );
}
```

**Установка соотношения сторон рабочей области**

Если основное значение отдается соотношению сторон рабочей области, можно задать для нее книжное или альбомное соотношение сторон. Соотношение сторон можно задать в дескрипторе приложения AIR или во время выполнения с помощью метода `setAspectRatio()` объекта Stage.

```
this.stage.setAspectRatio( StageAspectRatio.LANDSCAPE );
```

Среда выполнения выбирает одну из двух возможных ориентаций для указанного соотношения сторон. Она может не соответствовать текущей ориентации устройства. Например, по умолчанию в настройках выбрана перевернутая ориентация (AIR 3.2 и более ранних версий), а для выдвижной клавиатуры в настройках выбрана противоположная ориентация.

**(AIR 3.3 и более поздних версий)** Начиная с версии AIR 3.3 (SWF версии 16) можно также использовать константу `StageAspectRatio.ANY`. Если для параметра `Stage.autoOrients` установлено значение `true` и вызывается `setAspectRatio(StageAspectRatio.ANY)`, приложение может переориентироваться по всем направлениям (альбомная-влево, альбомная-вправо, книжная и книжная-перевернутая). Кроме того, нововведением AIR 3.3 является то, что пропорции являются постоянными, и в дальнейшем поворот устройства выполняется только с указанной ориентацией.

**Пример: установка ориентации рабочей области в соответствии с ориентацией устройства.**
В следующем примере демонстрируется функция, которая обновляет ориентацию рабочей области в соответствии с текущей ориентацией устройства. Свойство рабочей области `deviceOrientation` указывает физическую ориентацию устройства даже в том случае, если отключена автоматическая ориентация.

```
function refreshOrientation( theStage:Stage ):void
{
    switch ( theStage.deviceOrientation )
    {
        case StageOrientation.DEFAULT:
            theStage.setOrientation( StageOrientation.DEFAULT );
            break;
        case StageOrientation.ROTATED_RIGHT:
            theStage.setOrientation( StageOrientation.ROTATED_LEFT );
            break;
        case StageOrientation.ROTATED_LEFT:
            theStage.setOrientation( StageOrientation.ROTATED_RIGHT );
            break;
        case StageOrientation.UPSIDE_DOWN:
            theStage.setOrientation( StageOrientation.UPSIDE_DOWN );
            break;
        default:
            //No change
    }
}
```

Изменение ориентации выполняется асинхронно. Можно прослушивать событие `orientationChange`, отправляемое объектом Stage, для определения завершения изменения. Если устройство не поддерживает ориентацию, вызов `setOrientation()` завершается неудачей без появления ошибки.

# Динамическая загрузка содержимого для показа

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В приложение на языке ActionScript 3.0 можно загрузить любые внешние активы для показа:

- SWF-файл, разработанный в ActionScript 3.0 — этот файл может относиться к классу Sprite, MovieClip или любому классу, являющемуся расширением Sprite. В приложениях AIR в ОС iOS можно загружать только SWF-файлы, не содержащие байт-код ActionScript. Это означает, что можно загрузить SWF-файлы, содержащие внедренные данные, такие как изображения и звук, но нельзя загрузить SWF-файлы, содержащие исполняемый код.
- Файл изображения — файлы в форматах JPG, PNG и GIF.
- Файл AVM1 SWF — это SWF-файл, созданный на языке ActionScript 1.0 или 2.0. (не поддерживается в приложениях AIR для мобильных устройств)

Загрузка этих активов производится с помощью класса Loader.

## Загрузка экранных объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Объекты Loader используются для загрузки SWF-файлов и файлов графики в приложение. Класс Loader — это подкласс класса DisplayObjectContainer. Объект Loader может содержать в своем списке отображения только один дочерний экранный объект, который представляет собой загружаемый SWF-файл или файл графики. При добавлении объекта Loader в список отображения (как в следующем коде) в него после завершения загрузки добавляется также загруженный дочерний экранный объект:

```
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
this.addChild(pictLdr);
```

После загрузки SWF-файла или изображения можно переместить загруженный экранный объект в другой контейнер экранного объекта (например, в данном примере это `container` DisplayObjectContainer):

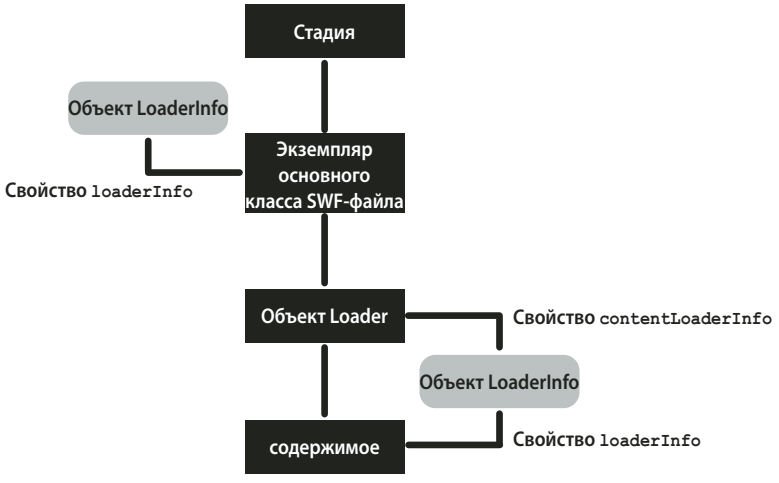
```
import flash.display.*;
import flash.net.URLRequest;
import flash.events.Event;
var container:Sprite = new Sprite();
addChild(container);
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
pictLdr.contentLoaderInfo.addEventListener(Event.COMPLETE, imgLoaded);
function imgLoaded(event:Event):void
{
    container.addChild(pictLdr.content);
}
```

## Просмотр хода загрузки

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

После начала загрузки файла создается объект LoaderInfo. Объект LoaderInfo предоставляет информацию о процессе загрузки, URL-адресах загружающего объекта и загружаемого содержимого, общем количестве байтов для носителя, а также номинальной высоте и ширине содержимого. Объект LoaderInfo также отправляет события, позволяющие следить за ходом загрузки.

На следующей диаграмме показаны различные применения объекта LoaderInfo (как экземпляр основного класса SWF-файла, как объект Loader и как объект, загруженный объектом Loader):




Объект LoaderInfo можно вызвать как свойство одновременно объекта Loader и загруженного экранного объекта. Как только начнется загрузка, объект LoaderInfo можно вызвать с помощью свойства `contentLoaderInfo` объекта Loader. По завершении загрузки экранного объекта объект LoaderInfo можно также вызвать в качестве свойства загруженного экранного объекта с помощью свойства `loaderInfo` экранного объекта. Свойство `loaderInfo` загруженного экранного объекта относится к тому же объекту LoaderInfo, что и свойство `contentLoaderInfo` объекта Loader. Другими словами, объект LoaderInfo совместно используется загруженным объектом и загрузившим его объектом Loader.

Чтобы вызвать свойства загруженного содержимого, необходимо добавить прослушиватель событий в объект LoaderInfo так, как это сделано в следующем коде:

```
import flash.display.Loader;
import flash.display.Sprite;
import flash.events.Event;

var ldr:Loader = new Loader();
var urlReq:URLRequest = new URLRequest("Circle.swf");
ldr.load(urlReq);
ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, loaded);
addChild(ldr);

function loaded(event:Event):void
{
    var content:Sprite = event.target.content;
    content.scaleX = 2;
}
```

Дополнительные сведения см. в разделе ««Обработка событий» на странице 134».

## Указание контекста загрузки

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При загрузке внешнего файла в Flash Player или AIR с помощью метода `load()` или `loadBytes()` класса Loader можно при желании указать параметр `context`. Этот параметр является объектом LoaderContext.

Класс LoaderContext включает три свойства, которые позволяют определить контекст применения загруженного содержимого.

- `checkPolicyFile`: используйте это свойство только при загрузке файла изображения (а не SWF-файла). Если этому свойству задано значение `true`, объект Loader проверяет наличие файла политики на исходном сервере (см. раздел «Элементы управления веб-сайта (файлы политики)» на странице 1132). Это необходимо выполнить только для содержимого, взятого из доменов, которые отличаются от домена SWF-файла, содержащего объект Loader. Если сервер дает разрешение на доступ к домену Loader, ActionScript из SWF-файлов в домене Loader может вызывать данные в загруженном изображении. Иными словами, можно использовать команду `BitmapData.draw()` для доступа к данным в загруженном изображении.

  Учтите, что SWF-файл из доменов, которые отличаются от домена объекта Loader, могут вызвать `Security.allowDomain()`, чтобы разрешить определенный домен.

- `securityDomain`: используйте это свойство только при загрузке SWF-файлов (но не изображений). Его следует указывать для SWF-файла из домена, который отличается от домена файла, содержащего объект Loader. Когда указывается этот параметр, Flash Player проверяет наличие файла политики. Если он существует, SWF-файлы из доменов, разрешенных файлом междоменной политики, могут применять загруженное содержимое SWF в разных сценариях. В качестве данного параметра можно указать `flash.system.SecurityDomain.currentDomain`.

- `applicationDomain`: используйте это свойство только при загрузке SWF-файлов, написанных на языке ActionScript 3.0 (а не изображений или SWF-файлов, написанных на ActionScript 1.0 или 2.0). При загрузке этого файла можно указать, чтобы файл был включен в тот же домен приложений, что и объект Loader, присвоив параметру `applicationDomain` значение `flash.system.ApplicationDomain.currentDomain`. Разместив загруженный SWF-файл в том же домене приложений, можно вызывать его классы напрямую. Это может оказаться полезным, если загружается SWF-файл, содержащий встроенные мультимедийные файлы, доступ к которым может осуществляться по связанным именам классов. Дополнительные сведения см. в разделе ««Работа с доменами приложений» на странице 158».

Вот пример проверки наличия файла политики при загрузке растрового изображения из другого домена:

```
var context:LoaderContext = new LoaderContext();
context.checkPolicyFile = true;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/photo11.jpg");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

Вот пример проверки наличия файла политики при загрузке SWF-файла из другого домена в целях размещения файла в той же изолированной программной среде, что и объект Loader. Кроме того, этот код добавляет классы в загруженном SWF-файле в тот же домен приложения, что и у объекта Loader:

```
var context:LoaderContext = new LoaderContext();
context.securityDomain = SecurityDomain.currentDomain;
context.applicationDomain = ApplicationDomain.currentDomain;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/library.swf");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

Дополнительные сведения см. в описании класса LoaderContext в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

## Загрузка SWF-файлов в среде AIR для iOS

**Adobe AIR 3.6 и более поздние версии, только в ОС iOS**

На устройствах под управлением iOS имеются ограничения на загрузку и компиляцию кода во время выполнения. Из-за этих ограничений возникают следующие сложности при загрузке внешних SWF-файлов в приложение.

- Все SWF-файлы, содержащие код ActionScript, должны входить в пакет приложения. Загрузка SWF с кодом из внешнего ресурса (например, по сети) невозможна. При создании пакета приложения весь код ActionScript в SWF-файлах пакета компилируется в собственный код устройств iOS.
- Загрузка, выгрузка и повторная загрузка SWF-файла невозможна. При попытке выполнить эти действия возникнет ошибка.
- Поведение среды при загрузке в память и выгрузке из нее не отличается от поведения настольных платформ. После выгрузки загруженного SWF-файла все визуальные ресурсы, содержащиеся в нем, будут выгружены из памяти. Однако все ссылки на классы ActionScript из загруженного SWF-файла останутся в памяти и могут быть использованы кодом ActionScript.
- Все загружаемые SWF-файлы должны быть загружены в том же домене приложения, что и основной SWF-файл. По умолчанию такого не происходит, поэтому для каждого загружаемого SWF-файла следует создавать объект LoaderContext с указанием на основной домен приложения и передавать этот объект LoaderContext при вызове метода Loader.load(). При попытке загрузить SWF-файл в домен приложения, отличный от домена приложения основного SWF-файла, возникает ошибка. Это правило также выполняется, если загружаемый SWF-файл содержит только визуальные ресурсы и не содержит кода ActionScript.

  В следующем примере показан код, которым осуществляется загрузка SWF-файла из пакета приложения в домен приложения основного SWF-файла.

```
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("swfs/SecondarySwf.swf");
var loaderContext:LoaderContext = new LoaderContext(false, ApplicationDomain.currentDomain,
null);
loader.load(url, loaderContext);
```

SWF-файл, содержащий только ресурсы без кода, может быть загружен как из пакета приложения, так и по сети. В обоих случаях этот SWF-файл так же необходимо загружать в основной домен приложения.

Если среда AIR имеет более раннюю версию, чем AIR 3.6, в процессе компиляции игнорируется весь код во всех SWF-файлах, кроме основного файла приложения. В пакет приложения для загрузки во время выполнения можно включать только SWF-Файлы, содержащие визуальные ресурсы и не содержащие код. При попытке загрузить SWF-файл, содержащий код ActionScript, возникнет ошибка. Эта ошибка приведет к появлению в приложении диалогового окна Uncompiled ActionScript (Нескомпилированный ActionScript).

**См. также**

Включение в пакет и загрузка несколькихSWF-файлов в приложениях AIR на iOS

## Использование классов ProLoader и ProLoaderInfo

**В средах Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий требуется Flash Professional CS5.5**

В среде Flash Professional CS5.5 представлены новые классы fl.display.ProLoader и fl.display.ProLoaderInfo, помогающие выполнить предварительную загрузку удаленной общей библиотеки (RSL). Эти классы зеркально воспроизводят классы flash.display.Loader и flash.display.LoaderInfo, но обеспечивают более последовательное взаимодействие при загрузке.

В частности, ProLoader помогает загрузить файлы SWF, использующие Text Layout Framework (TLF) с предварительной загрузкой RSL. Во время выполнения файлам SWF, предварительно загружающим другие файлы SWF или SWZ, например TLF, требуется только внутренний файл оболочки SWF. Дополнительный уровень сложности, связанный с использованием файла оболочки SWF, может привести к нежелательному поведению. ProLoader позволяет решить проблемы, связанные с загрузкой этих файлов, загружая их как обычные файлы SWF. Решение, используемое классом ProLoader, является прозрачным для пользователей и не требует специальной обработки в среде ActionScript. Помимо этого, класс ProLoader правильно загружает обычное содержимое SWF.

В среде Flash Professional CS5.5 и более поздних версий можно без последствий заменить все используемые экземпляры класса Loader классом ProLoader. Затем необходимо экспортировать программу в среду Flash Player 10.2 или более поздней версии, чтобы класс ProLoader мог получать доступ к необходимым функциям ActionScript. Класс ProLoader также можно использовать при разработке приложений для более ранних версий Flash Player, поддерживающих ActionScript 3.0. Однако полный функциональный набор класса ProLoader доступен только в среде Flash Player 10.2 или более поздних версий. Всегда используйте класс ProLoader вместе с TLF в среде Flash Professional CS5.5 или более поздних версий. Класс ProLoader не требуется в средах, отличных от Flash Professional.

***Важная информация.*** *Для файлов SWF, опубликованных во Flash Professional CS5.5 и более поздних версий, всегда можно использовать классы fl.display.ProLoader и fl.display.ProLoaderInfo вместо классов flash.display.Loader и flash.display.LoaderInfo.*

### Проблемы, которые решает класс ProLoader

Класс ProLoader решает проблемы, обработка которых не предусмотрена существующим классом Loader. Эти проблемы возникают в связи с предварительной загрузкой RSL библиотек TLF. В частности, они относятся к файлам SWF, использующим объект Loader для загрузки других файлов SWF. В числе решаемых проблем:

- **Сценарии между загружающим файлом и загруженным файлом не выполняются надлежащим образом.** Класс ProLoader автоматически задает загружающий SWF-файл в качестве родительского элемента загружаемого SWF-файла. В результате сообщения из загружающего файла SWF передаются непосредственно в загруженный файл SWF.
- **Программа SWF должна активно управлять процессом загрузки.** Для этого требуется реализация дополнительных событий, таких как `added`, `removed`, `addedToStage` и `removedFromStage`. Если приложение предназначено для проигрывателя Flash Player 10.2 или более поздних версий, класс ProLoader исключает необходимость выполнения дополнительной работы.

### Обновление кода для использования класса ProLoader вместо класса Loader

Поскольку класс ProLoader является зеркальным повторением класса Loader, можно без труда поочередно использовать в коде эти два класса. Следующий пример демонстрирует обновление существующего кода для использования нового класса.

```
import flash.display.Loader;
import flash.events.Event;
var l:Loader = new Loader();

addChild(l);
l.contentLoaderInfo.addEventListener(Event.COMPLETE, loadComplete);
l.load("my.swf");
function loadComplete(e:Event) {
    trace('load complete!');
}
```

Этот код можно обновить для использования класса ProLoader следующим образом:

```
import fl.display.ProLoader;
import flash.events.Event;
var l:ProLoader = new ProLoader();

addChild(l);
l.contentLoaderInfo.addEventListener(Event.COMPLETE, loadComplete);
l.load("my.swf");
function loadComplete(e:Event) {
    trace('load complete!');
}
```

# Пример экранного объекта: SpriteArranger

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Образец приложения SpriteArranger создан на основе образца приложения Geometric Shapes, описание которого приводится в документе *Изучение ActionScript 3.0*.

Образец приложения SpriteArranger иллюстрирует ряд понятий, используемых при операциях с экранными объектами:

- Расширение классов экранного объекта
- Добавление объектов в список отображения
- Расположение экранных объектов по слоям и работа с контейнерами экранного объекта
- Ответ на события экранного объекта
- Использование свойств и методов экранных объектов

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения SpriteArranger находятся в папке Examples/SpriteArranger. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| SpriteArranger.mxml<br>или<br>SpriteArranger.fla | Основной файл приложения Flash (FLA) или Flex (MXML). |
| com/example/programmingas3/SpriteArranger/CircleSprite.as | Класс, определяющий тип объекта Sprite, который визуализирует на экране круг. |
| com/example/programmingas3/SpriteArranger/DrawingCanvas.as | Класс, определяющий холст, который является контейнером экранного объекта, содержащим объекты GeometricSprite. |
| com/example/programmingas3/SpriteArranger/SquareSprite.as | Класс, определяющий тип объекта Sprite, который визуализирует на экране квадрат. |
| com/example/programmingas3/SpriteArranger/TriangleSprite.as | Класс, определяющий тип объекта Sprite, который визуализирует на экране треугольник. |
| com/example/programmingas3/SpriteArranger/GeometricSprite.as | Класс, являющийся расширением объекта Sprite и используемый для определения фигуры на экране. Расширениями данного класса являются CircleSprite, SquareSprite и TriangleSprite. |
| com/example/programmingas3/geometricshapes/IGeometricShape.as | Базовый интерфейс, определяющий методы, которые необходимо реализовать во всех классах геометрических фигур. |
| com/example/programmingas3/geometricshapes/IPolygon.as | Интерфейс, определяющий методы, которые должны быть реализованы классами геометрических фигур с несколькими сторонами. |
| com/example/programmingas3/geometricshapes/RegularPolygon.as | Тип геометрической фигуры с равными сторонами, которые симметрично расположены относительно центра фигуры. |
| com/example/programmingas3/geometricshapes/Circle.as | Тип геометрической фигуры, определяющей круг. |
| com/example/programmingas3/geometricshapes/EquilateralTriangle.as | Подкласс класса RegularPolygon, определяющий равносторонний треугольник. |
| com/example/programmingas3/geometricshapes/Square.as | Подкласс класса RegularPolygon, определяющий равносторонний прямоугольник. |
| com/example/programmingas3/geometricshapes/GeometricShapeFactory.as | Класс, содержащий «фабричный метод» создания фигур заданного типа и размера. |

## Определение классов SpriteArranger

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Приложение SpriteArranger дает пользователю возможность добавить ряд экранных объектов на экранный «холст».

Класс DrawingCanvas определяет область рисования и тип контейнера для экранного объекта, в который пользователь может добавить экранные фигуры. Эти экранные фигуры являются экземплярами одного из подклассов класса GeometricSprite.

### Класс DrawingCanvas

В программе Flex все дочерние экранные объекты, добавленные в объект Container, должны принадлежать классу, который наследует классу mx.core.UIComponent. Это приложение добавляет экземпляр класса DrawingCanvas в качестве дочернего объекта объекта mx.containers.VBox (см. определение в коде MXML, указанном в файле SpriteArranger.mxml). Это наследование определено в объявлении класса DrawingCanvas следующим образом:

```
public class DrawingCanvas extends UIComponent
```

Класс UIComponent наследует классам DisplayObject, DisplayObjectContainer и Sprite, а код в классе DrawingCanvas использует свойства и методы этих классов.

Класс DrawingCanvas является расширением класса Sprite, и это наследование определено в объявлении класса DrawingCanvas следующим образом:

```
public class DrawingCanvas extends Sprite
```

Класс Sprite представляет собой подкласс классов DisplayObjectContainer и DisplayObject, а класс DrawingCanvas использует методы и свойства этих классов.

Метод конструктора `DrawingCanvas()` настраивает объект Rectangle (`bounds`), который является свойством, используемым позднее при отрисовке контура холста. После этого он вызывает метод `initCanvas()` следующим образом:

```
this.bounds = new Rectangle(0, 0, w, h);
initCanvas(fillColor, lineColor);
```

Как показано в следующем примере, метод `initCanvas()` определяет различные свойства объекта DrawingCanvas, которые были переданы в качестве аргументов функции конструктора:

```
this.lineColor = lineColor;
this.fillColor = fillColor;
this.width = 500;
this.height = 200;
```

Затем метод `initCanvas()` вызывает метод `drawBounds()`, который отрисовывает холст с помощью свойства `graphics` класса DrawingCanvas. Свойство `graphics` унаследовано из класса Shape.

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
this.graphics.drawRect(bounds.left - 1,
                       bounds.top - 1,
                       bounds.width + 2,
                       bounds.height + 2);
this.graphics.endFill();
```

В зависимости от взаимодействия пользователя с приложением вызываются следующие дополнительные методы класса DrawingCanvas:

- методы `addShape()` и `describeChildren()`, описанные в разделе «Добавление на холст экранных объектов» на странице 223;
- методы `moveToBack()`, `moveDown()`, `moveToFront()` и `moveUp()`, описанные в разделе «Изменение слоя для экранного объекта» на странице 226;
- метод `onMouseUp()`, описанный в разделе «Нажатие и перетаскивание экранных объектов» на странице 225.

**Класс GeometricSprite и его подклассы**

Любой экранный объект, который может быть добавлен пользователем на холст, является экземпляром одного из следующих подклассов класса GeometricSprite:

- CircleSprite
- SquareSprite
- TriangleSprite

Класс GeometricSprite является расширением класса flash.display.Sprite:

```
public class GeometricSprite extends Sprite
```

Класс GeometricSprite включает ряд свойств, являющихся общими для всех объектов GeometricSprite. Они задаются в функции конструктора на основе параметров, переданных этой функции. Например:

```
this.size = size;
this.lineColor = lColor;
this.fillColor = fColor;
```

Свойство `geometricShape` класса GeometricSprite определяет интерфейс IGeometricShape, который в свою очередь определяет только математические, но не визуальные свойства фигуры. Классы, реализующие интерфейс IGeometricShape, определены в образце приложения GeometricShapes, описание которого приводится в документе *Изучение ActionScript 3.0*.

Класс GeometricSprite определяет метод `drawShape()`, более подробные описания которого приведены в определениях переопределения для каждого подкласса GeometricSprite. Дополнительные сведения см. в разделе «Добавление на холст экранных объектов» ниже.

Класс GeometricSprite предоставляет также другие методы:

- методы `onMouseDown()` и `onMouseUp()`, описанные в разделе «Нажатие и перетаскивание экранных объектов» на странице 225;
- методы `showSelected()` и `hideSelected()`, описанные в разделе «Нажатие и перетаскивание экранных объектов» на странице 225.

## Добавление на холст экранных объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Когда пользователь нажимает кнопку «Добавить фигуру», приложение вызывает метод `addShape()` класса DrawingCanvas. Он создает экземпляр нового объекта GeometricSprite путем вызова соответствующей функции конструктора одного из подклассов GeometricSprite, как показано в следующем примере:

```
public function addShape(shapeName:String, len:Number):void
{
    var newShape:GeometricSprite;
    switch (shapeName)
    {
        case "Triangle":
            newShape = new TriangleSprite(len);
            break;

        case "Square":
            newShape = new SquareSprite(len);
            break;

        case "Circle":
            newShape = new CircleSprite(len);
            break;
    }
    newShape.alpha = 0.8;
    this.addChild(newShape);
}
```

Каждый метод конструктора вызывает метод `drawShape()`, который использует свойство `graphics` данного класса (унаследованное из класса Sprite) для отрисовки соответствующей векторной графики. Например, метод `drawShape()` класса CircleSprite включает следующий код:

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
var radius:Number = this.size / 2;
this.graphics.drawCircle(radius, radius, radius);
```

Предпоследняя строка функции `addShape()` задает свойство `alpha` экранного объекта (унаследованное из класса DisplayObject) таким образом, что каждый добавленный на холст экранный объект слегка прозрачен, что дает пользователю возможность увидеть объекты за ним.

Последняя строка метода `addChild()` добавляет новый экранный объект в список дочерних объектов экземпляра класса DrawingCanvas, который уже находится в списке отображения. Это приводит к тому, что в рабочей области появляется новый экранный объект.

Интерфейс приложения включает два текстовых поля (`selectedSpriteTxt` и `outputTxt`). Свойства текста этих текстовых полей обновляются благодаря информации об объектах GeometricSprite, которые добавлены на холст или выбраны пользователем. Класс GeometricSprite обрабатывает это задание по предоставлению информации путем переопределения метода `toString()`:

```
public override function toString():String
{
    return this.shapeType + " of size " + this.size + " at " + this.x + ", " + this.y;
}
```

Свойству `shapeType` присваивается соответствующее значение в методе конструктора каждого из подклассов GeometricSprite. Например, метод `toString()` может вернуть следующее значение для экземпляра CircleSprite, недавно добавленного к экземпляру DrawingCanvas:

```
Circle of size 50 at 0, 0
```

Метод `describeChildren()` класса DrawingCanvas обеспечивает циклическую обработку списка дочерних объектов холста. При этом свойство `numChildren` (унаследованное из класса DisplayObjectContainer) применяется, чтобы задать предел цикла `for`. Он генерирует следующую строку со списком всех дочерних объектов:

```
var desc:String = "";
var child:DisplayObject;
for (var i:int=0; i < this.numChildren; i++)
{
child = this.getChildAt(i);
desc += i + ": " + child + '\n';
}
```

Эта строка применяется для задания свойства `text` текстового поля `outputTxt`.

## Нажатие и перетаскивание экранных объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Когда пользователь щелкает по экземпляру GeometricSprite, приложение вызывает обработчик событий `onMouseDown()`. Как указано далее, этот обработчик событий настроен на прослушивание событий нажатия кнопки мыши в функции конструктора класса GeometricSprite:

```
this.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
```

Метод `onMouseDown()` вызывает затем метод `showSelected()` объекта GeometricSprite. Если этот метод вызывается для объекта впервые, он создает новый объект Shape с именем `selectionIndicator` и использует свойство `graphics` объекта Shape для отрисовки прямоугольника с красной подсветкой следующим образом:

```
this.selectionIndicator = new Shape();
this.selectionIndicator.graphics.lineStyle(1.0, 0xFF0000, 1.0);
this.selectionIndicator.graphics.drawRect(-1, -1, this.size + 1, this.size + 1);
this.addChild(this.selectionIndicator);
```

Если метод `onMouseDown()` вызван уже не в первый раз, он просто задает для фигуры `selectionIndicator` свойство `visible` (унаследованное из класса DisplayObject) следующим образом:

```
this.selectionIndicator.visible = true;
```

Метод `hideSelected()` скрывает фигуру `selectionIndicator` ранее выбранного объекта путем присвоения свойству `visible` значения `false`.

Метод обработчика событий `onMouseDown()` вызывает также метод `startDrag()` (унаследованный из класса Sprite), который включает следующий код:

```
var boundsRect:Rectangle = this.parent.getRect(this.parent);
boundsRect.width -= this.size;
boundsRect.height -= this.size;
this.startDrag(false, boundsRect);
```

Это дает пользователю возможность перетащить выбранный объект через холст в пределах, заданных прямоугольником `boundsRect`.

Когда пользователь отпускает кнопку мыши, отправляется событие `mouseUp`. Метод конструктора DrawingCanvas устанавливает следующий обработчик событий:

```
this.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
```

Этот обработчик событий задается для объекта DrawingCanvas, а не для отдельных объектов GeometricSprite. Это происходит потому, что при перетаскивании объекта GeometricSprite он может оказаться задвинутым за другой экранный объект (другой объект GeometricSprite), когда будет отпущена кнопка мыши. Экранный объект на переднем плане зарегистрирует событие отпускания кнопки мыши (mouse up) в отличие от экранного объекта, перетаскиваемого пользователем. Добавление прослушивателя к объекту DrawingCanvas гарантирует обработку события.

Метод `onMouseUp()` вызывает метод `onMouseUp()` объекта GeometricSprite, который в свою очередь вызывает метод `stopDrag()` объекта GeometricSprite.

## Изменение слоя для экранного объекта

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Пользовательский интерфейс приложения включает кнопки Move Back («Переместить назад»), Move Down («Переместить вниз»), Move Up («Переместить вверх») и Move to Front («Переместить вперед»). Когда пользователь нажимает одну из этих кнопок, приложение вызывает соответствующий метод класса DrawingCanvas: `moveToBack()`, `moveDown()`, `moveUp()` или `moveToFront()`. Например, метод `moveToBack()` включает следующий код:

```
public function moveToBack(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, 0);
    }
}
```

Этот метод использует метод `setChildIndex()` (унаследованный из класса DisplayObjectContainer) для размещения экранного объекта на позиции указателя 0 в списке дочерних объектов экземпляра DrawingCanvas (`this`).

Метод `moveDown()` действует похожим образом, за исключением того, что он уменьшает позицию указателя для экранного объекта на 1 в списке дочерних объектов экземпляра DrawingCanvas:

```
public function moveDown(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, index - 1);
    }
}
```

Методы `moveUp()` и `moveToFront()` аналогичны методам `moveToBack()` и `moveDown()`.

# Глава 11. Работа с геометрией

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Пакет flash.geom содержит классы, определяющие геометрические объекты, такие как точки, прямоугольники и матрицы преобразований. Эти классы используются для определения свойств объектов, которые применяются в других классах.

### Дополнительные разделы справки

пакет flash.geom

## Основы геометрии

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Пакет flash.geom содержит классы, определяющие геометрические объекты, такие как точки, прямоугольники и матрицы преобразований. Эти классы не обязательно обладают самостоятельной функциональностью, тем не менее, они используются для определения свойств объектов, которые применяются в других классах.

Все геометрические классы базируются на том, что положение объекта на экране представляется в двумерной плоскости. Экран рассматривается как плоская диаграмма с горизонтальной (X) и вертикальной (Y) осями. Местоположение любой *точки* на экране представляется двумя значениями x и y, которые называются *координатами* местоположения.

Каждый экранный объект, включая Stage, имеет собственное *координатное пространство*. Координатное пространство представляет собой собственный график объекта для нанесения местоположений дочерних экранных объектов, чертежей и других объектов. *Начало координат* — это точка с координатами 0, 0 (на пересечении осей X и Y), которая находится в верхнем левом углу экранного объекта. Это начало координат всегда справедливо для рабочей области, но не обязательно для остальных экранных объектов. Значение на оси X увеличиваются при перемещении вправо и уменьшаются при перемещении влево. Точки слева от начала координат имеют отрицательную координату X. Однако в отличие от традиционных координатных систем, в среде выполнения Flash значения на оси Y увеличиваются при перемещении вниз и уменьшаются при перемещении вверх. Точки выше начала координат имеют отрицательную координату Y. Так как верхний левый угол рабочей области является началом координат ее координатного пространства, для большинства расположенных в ней объектов координата X больше нуля и меньше ширины рабочей области. А координата Y этого же объекта больше 0 меньше высоты рабочей области.

Для представления отдельных точек в координатном пространстве можно воспользоваться экземплярами класса Point. Для представления прямоугольной области в координатном пространстве можно создать экземпляр класса Rectangle. Опытные пользователи могут использовать экземпляр класса Matrix для применения комплексных преобразований к экранному объекту. Многие преобразования, такие как поворот, изменения положения и масштаба, могут применяться к экранному объекту непосредственно с использованием свойств этого объекта. Для получения дополнительной информации о преобразованиях с использованием свойств экранных объектов см. раздел «Операции с экранными объектами» на странице 186.

**Важные понятия и термины**

В следующем справочном списке содержатся важные геометрические термины.

**Декартовы координаты** Координаты, которые обычно выражаются парой чисел (например, 5, 12 или 17, -23). Указанные два числа являются x-координатой и y-координатой соответственно.

**Координатное пространство** Диаграмма координат, содержащаяся в экранном объекте, на которой располагаются его дочерние элементы.

**Начало координат** Точка координатного пространства, в которой пересекаются оси X и Y. Эта точка имеет координаты 0, 0.

**Точка** Единичный пункт в координатном пространстве. В двухмерной системе координат, используемой в ActionScript, точка определяется как местоположение, расположенное вдоль осей X и Y (координаты точки).

**Точка регистрации** Начало отсчета координатного пространства (координаты 0, 0) в экранном объекте.

**Масштаб** Размер объекта по отношению к исходному размеру. Масштабирование — изменение размера объекта за счет его сжатия или расширения.

**Перемещение** Изменение координат точек при переходе из одного координатного пространства в другое.

**Преобразование** Корректировка визуальных характеристик графики, например: поворот, масштабирование, сдвиг, деформация или изменение цвета объекта.

**Ось X** Горизонтальная ось в двумерной системе координат, используемой в ActionScript.

**Ось Y** Вертикальная ось в двумерной системе координат, используемой в ActionScript.

# Использование объектов Point

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Точечный объект Point определяется парой декартовых координат. Он представляет точку в двумерной системе координат, где *x* является значением по горизонтальной оси, а *y* — значением по вертикальной оси.

Для определения объекта Point необходимо задать его свойства `x` и `y` следующим образом:

```
import flash.geom.*;
var pt1:Point = new Point(10, 20); // x == 10; y == 20
var pt2:Point = new Point();
pt2.x = 10;
pt2.y = 20;
```

## Определение расстояния между двумя точками

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В классе Point для нахождения расстояния между двумя точками в координатном пространстве используется метод `distance()`. Например, следующий код позволяет найти расстояние между точками регистрации двух экранных объектов `circle1` и `circle2`, расположенных в одном контейнере:

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
var pt2:Point = new Point(circle2.x, circle2.y);
var distance:Number = Point.distance(pt1, pt2);
```

## Перемещение координатных пространств

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если два экранных объекта находятся в различных контейнерах, они могут быть в различных координатных пространствах. Для класса DisplayObject можно использовать метод `localToGlobal()` для перевода координат в единое (глобальное) координатное пространство, применяемое для рабочей области. Например, следующий код позволяет найти расстояние между точками регистрации двух экранных объектов `circle1` и `circle2`, расположенных в различных контейнерах:

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
pt1 = circle1.localToGlobal(pt1);
var pt2:Point = new Point(circle2.x, circle2.y);
pt2 = circle2.localToGlobal(pt2);
var distance:Number = Point.distance(pt1, pt2);
```

Аналогичным образом можно найти расстояние от точки регистрации экранного объекта `target` до заданной точки рабочей области, используя метод `localToGlobal()` класса DisplayObject:

```
import flash.geom.*;
var stageCenter:Point = new Point();
stageCenter.x = this.stage.stageWidth / 2;
stageCenter.y = this.stage.stageHeight / 2;
var targetCenter:Point = new Point(target.x, target.y);
targetCenter = target.localToGlobal(targetCenter);
var distance:Number = Point.distance(stageCenter, targetCenter);
```

## Перемещение экранного объекта на заданный угол и расстояние

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для перемещения экранного объекта на заданный угол и расстояние можно использовать метод `polar()` из класса Point. Например, следующий код позволяет переместить объект `myDisplayObject` на 100 пикселов и 60°:

```
import flash.geom.*;
var distance:Number = 100;
var angle:Number = 2 * Math.PI * (90 / 360);
var translatePoint:Point = Point.polar(distance, angle);
myDisplayObject.x += translatePoint.x;
myDisplayObject.y += translatePoint.y;
```

## Другое использование класса Point

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Объекты Point можно использовать со следующими функциями и свойствами.

| Класс | Методы или свойства | Описание |
|---|---|---|
| DisplayObjectContainer | `areInaccessibleObjectsUnderPoint()` `getObjectsUnderPoint()` | Используется для возвращения списка объектов под точкой в контейнер экранных объектов. |
| BitmapData | `hitTest()` | Используется для определения пиксела в объекте BitmapData, а также точки, проверяемой на попадание. |
| BitmapData | `applyFilter()`<br>`copyChannel()`<br>`merge()`<br>`paletteMap()`<br>`pixelDissolve()`<br>`threshold()` | Используется для определения позиций прямоугольников, определяющих операции. |
| Matrix | `deltaTransformPoint()`<br>`transformPoint()` | Используется для определения точек, к которым требуется применить преобразование. |
| Rectangle | `bottomRight`<br>`size`<br>`topLeft` | Используется для определения этих свойств. |

# Использование объектов Rectangle

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Объект Rectangle используется для определения прямоугольных областей. Положение объекта Rectangle определяется координатами его верхнего левого угла *x* и *y*, а также свойствами ширины `width` и высоты `height` . Чтобы задать эти свойства для нового объекта Rectangle, необходимо вызвать функцию конструктора `Rectangle()` следующим образом:

```
import flash.geom.Rectangle;
var rx:Number = 0;
var ry:Number = 0;
var rwidth:Number = 100;
var rheight:Number = 50;
var rect1:Rectangle = new Rectangle(rx, ry, rwidth, rheight);
```

## Изменение размеров и положения объектов Rectangle

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Существует несколько способов для изменения размеров и положения объектов Rectangle.

Можно непосредственно изменить положение объекта Rectangle, изменив его свойства `x` и `y`. Это изменение не окажет никакого влияния на ширину и высоту объекта Rectangle.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.x = 20;
rect1.y = 30;
trace(rect1); // (x=20, y=30, w=100, h=50)
```

Как демонстрирует следующий пример, при изменении свойства `left` или `top` объекта Rectangle изменяется местоположение прямоугольника. Свойства `x` и `y` прямоугольника соответствуют свойствам `left` и `top` соответственно. Тем не менее, положение нижнего левого угла объекта Rectangle остается прежним, хотя размеры его меняются.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.left = 20;
rect1.top = 30;
trace(rect1); // (x=20, y=30, w=80, h=20)
```

Аналогичным образом, как показано в следующем примере, при изменении свойства `bottom` и `right` объекта Rectangle положение верхнего левого угла прямоугольника остается прежним, а размеры меняются.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.right = 60;
trect1.bottom = 20;
trace(rect1); // (x=0, y=0, w=60, h=20)
```

Изменить положение объекта Rectangle можно также с помощью метода `offset()`, как показано ниже:

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.offset(20, 30);
trace(rect1); // (x=20, y=30, w=100, h=50)
```

Метод `offsetPt()` работает аналогичным образом, за исключением того, что в нем в качестве параметра используется объект Point вместо значений смещения *x* и *y*.

Изменить положение объекта Rectangle можно также с помощью метода `inflate()`, в котором используются два параметра, `dx` и `dy`. Параметр `dx` обозначает число пикселов, на которое правая и левая стороны прямоугольника перемещаются относительно центра. Параметр `dy` обозначает число пикселов, на которое верхняя и нижняя стороны прямоугольника перемещаются относительно центра.

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.inflate(6,4);
trace(rect1); // (x=-6, y=-4, w=112, h=58)
```

Метод `inflatePt()` работает аналогичным образом, за исключением того, что в нем в качестве параметра используется объект Point вместо значений смещения `dx` и `dy`.

## Нахождение объединений и пересечений объектов Rectangle

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для нахождения прямоугольной области, сформированной границами двух прямоугольников, используется функция объединения `union()`:

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(120, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.union(rect2)); // (x=0, y=0, w=220, h=160)
```

Для нахождения прямоугольной области, сформированной перекрытием двух прямоугольников, используется функция пересечения `intersection()`:

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(80, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.intersection(rect2)); // (x=80, y=60, w=20, h=40)
```

Метод `intersects()` используется для определения факта пересечения двух прямоугольников. Метод `intersects()` используется также для определения факта нахождения определенного экранного объекта на заданном участке рабочей области. Например, в следующем коде предполагается, что координатное пространство контейнера, содержащего экранный объект `circle`, совпадает с пространством рабочей области. Пример показывает, как использовать метод `intersects()` для определения факта пересечения экранным объектом `circle` определенных участков рабочей области, заданных объектами Rectangle `target1` и `target2`:

```
import flash.display.*;
import flash.geom.Rectangle;
var circle:Shape = new Shape();
circle.graphics.lineStyle(2, 0xFF0000);
circle.graphics.drawCircle(250, 250, 100);
addChild(circle);
var circleBounds:Rectangle = circle.getBounds(stage);
var target1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(circleBounds.intersects(target1)); // false
var target2:Rectangle = new Rectangle(0, 0, 300, 300);
trace(circleBounds.intersects(target2)); // true
```

Аналогичным образом можно использовать метод `intersects()` для определения факта перекрытия смежных прямоугольников двух экранных объектов. Используйте метод `getRect()` класса DisplayObject для включения дополнительного пространства, которое обводки экранного объекта могут добавлять к граничной области.

## Другое использование объектов Rectangle

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Объекты Rectangle можно использовать со следующими функциями и свойствами:

| Класс | Методы или свойства | Описание |
|---|---|---|
| BitmapData | `applyFilter()`, `colorTransform()`, `copyChannel()`, `copyPixels()`, `draw()`, `drawWithQuality()`, `encode()`, `fillRect()`, `generateFilterRect()`, `getColorBoundsRect()`, `getPixels()`, `merge()`, `paletteMap()`, `pixelDissolve()`, `setPixels()`, `threshold()` | Используется как тип для некоторых параметров, определяющих область объекта BitmapData. |
| DisplayObject | `getBounds()`, `getRect()`, `scrollRect`, `scale9Grid` | Используется как тип данных для свойства или как возвращенный тип данных. |
| PrintJob | `addPage()` | Используется для определения параметра `printArea`. |
| Sprite | `startDrag()` | Используется для определения параметра `bounds`. |
| TextField | `getCharBoundaries()` | Используется как тип возвращенного значения. |
| Transform | `pixelBounds` | Используется как тип данных. |

# Использование объектов Matrix

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс Matrix представляет матрицу преобразования, определяющую способ сопоставления точек из одного пространства координат с точками в другом пространстве координат. Можно выполнять различные графические преобразования экранного объекта путем задания свойств объекта Matrix, применения этого объекта Matrix к свойству `matrix` объекта Transform и последующего применения объекта Transform в качестве свойства `transform` к экранному объекту. Эти функции преобразования включают перемещение (изменение положения осей *x* и *y*), поворот, масштабирование и наклон.

Несмотря на то, что матрицу можно определить непосредственно, задав свойства объекта Matrix (`a`, `b`, `c`, `d`, `tx`, `ty`), легче использовать для этого метод `createBox()`. Этот метод включает параметры, позволяющие непосредственно задавать эффекты масштабирования поворота и перемещения для результирующей матрицы. Например, следующий код позволяет создать объект Matrix, который выполняет масштабирование объекта по вертикали с коэффициентом 2 и по горизонтали с коэффициентом 3, поворот на 45°, перемещение на 10 пикселов направо и на 20 пикселов вниз:

```
var matrix:Matrix = new Matrix();
var scaleX:Number = 2.0;
var scaleY:Number = 3.0;
var rotation:Number = 2 * Math.PI * (45 / 360);
var tx:Number = 10;
var ty:Number = 20;
matrix.createBox(scaleX, scaleY, rotation, tx, ty);
```

Используя методы `scale()`, `rotate()` и `translate()`, можно также откорректировать эффекты масштабирования, поворота и перемещения объекта Matrix. Обратите внимание, что в этих методах используется комбинация данных со значениями существующего объекта Matrix. Например, следующий код задает объект Matrix, который будет включать эффекты масштабирования с коэффициентом 4 и поворота на 60°, поскольку методы `scale()` и `rotate()` использованы дважды:

```
var matrix:Matrix = new Matrix();
var rotation:Number = 2 * Math.PI * (30 / 360); // 30°
var scaleFactor:Number = 2;
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);
matrix.scale(scaleX, scaleY);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

Для применения преобразования наклона к объекту Matrix необходимо задать его свойства `b` или `c`. Свойство `b` задает вертикальный наклон матрицы, свойство `c` задает горизонтальный наклон. Следующий код позволяет наклонить матричный объект `myMatrix` по вертикали с коэффициентом 2:

```
var skewMatrix:Matrix = new Matrix();
skewMatrix.b = Math.tan(2);
myMatrix.concat(skewMatrix);
```

Матричное преобразование можно применить к свойству `transform` экранного объекта. Например, следующий код позволяет применить матричное преобразование к экранному объекту с именем `myDisplayObject`:

```
var matrix:Matrix = myDisplayObject.transform.matrix;
var scaleFactor:Number = 2;
var rotation:Number = 2 * Math.PI * (60 / 360); // 60°
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

Первая строка задает объект Matrix в существующем матричном преобразовании как экранный объект `myDisplayObject` (свойство `matrix` свойства `transformation` экранного объекта `myDisplayObject`). Таким образом, используемые методы класса Matrix имеют кумулятивный эффект на существующие положение, масштаб и поворот экранного объекта.

***Примечание.*** *Класс ColorTransform также включен в пакет flash.geometry. Этот класс используется для задания свойства* `colorTransform` *для объекта Transform. Данный класс не имеет отношения к каким-либо геометрическим преобразованиям, поэтому он здесь подробно не рассматривается. Дополнительные сведения см. в описании класса ColorTransform в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.*

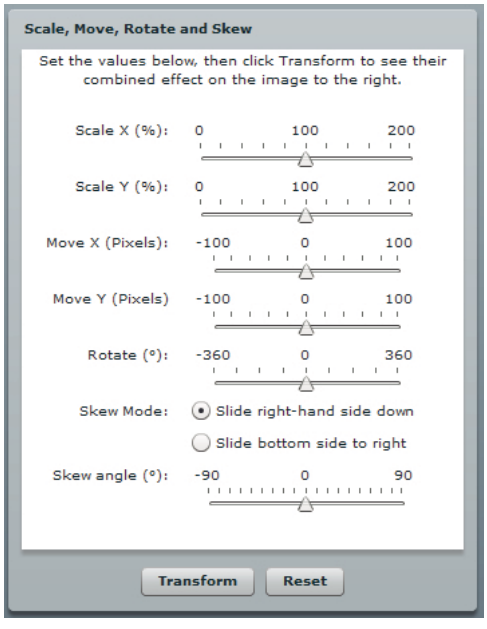
# Пример геометрии: применение матричного преобразования к экранному объекту

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В качестве примера приводится приложение DisplayObjectTransformer, которое демонстрирует ряд функций, используемых в классе Matrix для преобразования экранного объекта. Среди этих функций следующие:

- Поворот экранного объекта
- Масштабирование экранного объекта
- Перемещение экранного объекта
- Наклон экранного объекта

Приложение предоставляет интерфейс для настройки параметров матричного преобразования, как показано ниже.




При нажатии кнопки Transform (Преобразовать) приложение выполняет заданные преобразования.

*Исходный экранный объект и экранный объект, повернутый на -45° и уменьшенный на 50 %*

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения DisplayObjectTransformer находятся в папке Samples/DisplayObjectTransformer. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| DisplayObjectTransformer.mxml<br>или<br>DisplayObjectTransformer.fla | Основной файл приложения Flash (FLA) или Flex (MXML) |
| com/example/programmingas3/geometry/MatrixTransformer.as | Класс, содержащий методы для применения матричного преобразования. |
| img/ | Папка, содержащая файлы образцов изображений, используемых в приложении. |

## Определение класса MatrixTransformer

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс MatrixTransformer включает статические методы, которые применяют геометрические преобразования к объектам Matrix.

### Метод transform()

Метод `transform()` включает параметры для нижеследующего:

- `sourceMatrix` — входная матрица, которая преобразуется с помощью этого метода
- `xScale` и `yScale` — масштабные коэффициенты *x* и *y*
- `dx` и `dy` — величины перемещения *x* и *y* в пикселах
- `rotation` — угол поворота в градусах
- `skew` — коэффициент наклона в процентах
- `skewType` — направление наклона вправо или влево (`"right"` или `"left"`)

Возвращаемое значение — результирующая матрица.

Метод `transform()` вызывает следующие статические методы этого класса:

- `skew()`
- `scale()`
- `translate()`

- `rotate()`

Каждый из них возвращается в исходную матрицу с примененным преобразованием.

### Метод skew()

Метод `skew()` вносит в матрицу корректировку наклона, задавая свойства матрицы `b` и `c`. Дополнительный параметр `unit` задает единицы измерения угла наклона — при необходимости в этом методе возможно конвертирование значения угла `angle` в радианы.

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
```

Матричный объект `skewMatrix` создается и корректируется для применения преобразования наклона. Первоначально это единичная матрица:

```
var skewMatrix:Matrix = new Matrix();
```

Параметр `skewSide` определяет сторону, к которой применяется преобразование наклона. Если задана правая сторона `"right"`, свойство матрицы `b` устанавливает следующий код:

```
skewMatrix.b = Math.tan(angle);
```

Нижний край наклоняется заданием свойства матрицы `c`:

```
skewMatrix.c = Math.tan(angle);
```

Результирующий наклон применяется к исходной матрице за счет объединения двух матриц, как показано в следующем примере:

```
sourceMatrix.concat(skewMatrix);
return sourceMatrix;
```

### Метод scale()

Как показано в следующем примере, метод `scale()` сначала задает коэффициент масштабирования, если он представлен в процентах, затем метод `scale()` применяется к объекту Matrix:

```
if (percent)
{
    xScale = xScale / 100;
    yScale = yScale / 100;
}
sourceMatrix.scale(xScale, yScale);
return sourceMatrix;
```

### Метод translate()

Метод `translate()` просто применяет коэффициенты преобразования `dx` и `dy` путем вызова метода `translate()` матричного объекта, как показано ниже:

```
sourceMatrix.translate(dx, dy);
return sourceMatrix;
```

**Метод rotate()**

Метод `rotate()` конвертирует входной коэффициент поворота в радианы (если он представлен в градусах или единицах градиента), а затем вызывает метод `rotate()` матричного объекта:

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
sourceMatrix.rotate(angle);
return sourceMatrix;
```

## Вызов метода MatrixTransformer.transform() в приложении

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Приложение содержит интерфейс, с помощью которого пользователь вводит параметры преобразования. Затем приложение передает эти значения вместе со свойствами экранного объекта `matrix` и `transform` методу `Matrix.transform()`, как показано ниже:

```
tempMatrix = MatrixTransformer.transform(tempMatrix,
    xScaleSlider.value,
    yScaleSlider.value,
    dxSlider.value,
    dySlider.value,
    rotationSlider.value,
    skewSlider.value,
    skewSide );
```

После этого приложение применяет возвращаемое значение к свойству `matrix` свойства `transform` экранного объекта, осуществляя преобразование:

```
img.content.transform.matrix = tempMatrix;
```

# Глава 12. Использование API рисования

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Наряду с возможностью импортирования изображений и графических объектов у пользователя существует возможность начать работу с чистого холста. Набор функциональных возможностей API рисования позволяет рисовать в ActionScript линии и фигуры, создавая, таким образом, желаемые изображения. Возможность создавать собственную графику предоставляет пользователям широкое поле деятельности. С помощью технических приемов, описанных в данной главе, можно создать программу для рисования, анимированные интерактивные графические объекты, программные элементы пользовательского интерфейса и многое другое.

### Дополнительные разделы справки

flash.display.Graphics

## Основы использования API-интерфейса рисования

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

API рисования — это функция, встроенная в ActionScript, позволяющая создавать векторную графику (линии, кривые, фигуры, заливки и градиенты) и отображать ее на экране с помощью кода ActionScript. Данная функция обеспечивается классом flash.display.Graphics. С помощью кода ActionScript можно рисовать на любом экземпляре Shape, Sprite или MovieClip, используя свойство `graphics`, определенное в каждом из этих классов. (Свойство `graphics` каждого из классов на самом деле является экземпляром класса Graphics.)

Если пользователь никогда прежде не сталкивался с рисованием с помощью кода, то он может начать знакомство с этой возможностью с методов класса Graphics, упрощающих рисование обычных фигур, таких как круги, эллипсы, прямоугольники или прямоугольники со скругленными краями. Эти фигуры можно рисовать в виде контуров или фигур с заливкой. При необходимости использования расширенных функций можно воспользоваться и другими методами класса Graphics для рисования линий и квадратичных кривых Безье, используемых вместе с тригонометрическими функциями класса Math для создания любой необходимой фигуры.

Среды выполнения Flash (такие как Flash Player 10, Adobe AIR 1.5 и более поздние их версии) добавляют дополнительный API-интерфейс рисования, позволяющий программно рисовать целые фигуры с помощью одной команды. После ознакомления с классом Graphics и задачами, описанными в разделе «Основы использования API рисования», следует перейти к разделу ««Расширенное использование API рисования» на странице 253» для получения дополнительных сведений об этих функциях API.

#### Важные понятия и термины

В следующем списке содержатся важные термины, которые встречаются при использовании API-интерфейса рисования:

**Опорная точка** Одна из двух конечных точек квадратичной кривой Безье.

**Контрольная точка** Точка, определяющая направление и степень кривизны квадратичной кривой Безье. Кривая линия никогда не достигает контрольной точки, но рисуется так, что к ней направлены ее изгибы.

**Координатное пространство** Диаграмма координат, содержащаяся в экранном объекте, на которой располагаются его дочерние элементы.

**Заливка** Сплошное заполнение внутренней части фигуры цветом по контуру, либо всей фигуры, если контур отсутствует.

**Градиент** Цвет, состоящий из постепенного перехода от одного цвета к другому или нескольких другим цветам (в отличие от сплошного цвета).

**Точка** Единичный пункт в координатном пространстве. В двумерной координатной системе, используемой в ActionScript, точка определяется положением относительно осей x и y (координатами точки).

**Квадратичная кривая Безье** Тип кривой, определенный специальной математической формулой. Форма кривой этого типа вычисляется на основе позиций опорных точек (конечных точек кривой) и контрольной точки, определяющей степень кривизны и направление кривой.

**Масштаб** Размер объекта по отношению к исходному размеру. Масштабирование — изменение размера объекта за счет его сжатия или расширения.

**Обводка** Контур фигуры, состоящий из линии, заполненной цветом или линии без заполнения.

**Перемещение** Изменение координат точек при переходе из одного координатного пространства в другое.

**Ось X** Горизонтальная ось в двумерной системе координат, используемой в ActionScript.

**Ось Y** Вертикальная ось в двумерной системе координат, используемой в ActionScript.

# Класс Graphics

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Каждый из объектов Shape, Sprite и MovieClip наделен свойством `graphics`, являющимся экземпляром класса Graphics. Класс Graphics включает свойства и методы, предназначенные для создания линий, заливок и фигур. При необходимости использования объекта отображения в качестве холста для содержимого рисунка можно использовать экземпляр Shape. Экземпляр Shape лучше остальных объектов отображения подходит для рисования, поскольку он не имеет служебных данных, связанных с дополнительными функциональными возможностями в классах Sprite и MovieClip. Если требуется, чтобы объект отображения, на котором можно создавать графическое содержимое, также содержал и другие объекты отображения, можно использовать экземпляр Sprite. Дополнительные сведения о правильном выборе экранного объекта для выполнения определенных задач см. в разделе ««Выбор подкласса DisplayObject» на странице 185».

# Создание линий и кривых

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Все рисунки, создаваемые с помощью экземпляра Graphics, основываются на стандартном рисовании с помощью линий и кривых. В результате этого все рисунки ActionScript создаются с помощью одних и тех же шагов:

- определение стиля линий и заливок;
- определение начального положения рисуемого объекта;

- рисование линий, кривых и фигур (при желании — перемещение точки рисунка);
- при необходимости — заполнение фигур заливкой.

## Определение стилей линий и заливок

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для рисования с помощью свойства `graphics` экземпляра Shape, Sprite или MovieClip прежде всего необходимо определить применяемый стиль (цвет и размер линий, цвет заливки). При рисовании с помощью ActionScript можно использовать или не использовать штрих или заливку, аналогично тому, как это происходит в Adobe® Flash® Professional и других приложениях, предназначенных для рисования. Внешний вид обводки определяется с помощью метода `lineStyle()` или `lineGradientStyle()`. Для создания сплошной линии используется метод `lineStyle()`. При вызове этого метода пользователи наиболее часто изменяют первые три параметра: толщину, цвет и альфа-канал линии. Например, данная строка кода указывает фигуре с именем `myShape` на необходимость создания линии толщиной 2 пиксела, красного цвета (0x990000), со степенью непрозрачности 75%:

```
myShape.graphics.lineStyle(2, 0x990000, .75);
```

По умолчанию значение альфа-канала равно 1,0 (100%), поэтому при необходимости создания полностью непрозрачной линии данный параметр может быть проигнорирован. Метод `lineStyle()` также может иметь два дополнительных параметра для хинтинга пикселов и режима масштабирования; дополнительные сведения об использовании этих параметров см. в описании метода `Graphics.lineStyle()` в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

Для создания линии градиента используется метод `lineGradientStyle()`. Описание этого метода см. в разделе «Создание линий градиента и заливок» на странице 244.

При необходимости создания фигуры с заливкой перед началом рисования вызовите методы `beginFill()`, `beginGradientFill()`, `beginBitmapFill()` или `beginShaderFill()`. Основной из этих методов, метод `beginFill()`, имеет два параметра: цвет заливки и (необязательно) значение альфа-канала для цвета заливки. Например, для создания фигуры со сплошной заливкой зеленого цвета необходимо использовать следующий код (предположим, что создается объект с именем `myShape`):

```
myShape.graphics.beginFill(0x00FF00);
```

Вызов любого метода всегда завершает выполнение предыдущей заливки перед началом выполнения новой заливки. Вызов любого метода, определяющего стиль штриха, заменяет предыдущий стиль штриха, но не изменяет предыдущую заливку и наоборот.

По окончании выбора стиля линий и свойств заливок необходимо указать начальную точку рисунка. Экземпляр Graphics имеет точку рисунка, похожую на точку, которая остается от прикосновения пера к бумаге. Следующее действие рисования начинается с той позиции, где находится точка рисунка. Изначально объект Graphics начинается с точки рисунка, находящейся в точке 0, 0 пространства координат объекта, в котором выполняется рисование. Для начала рисования в другой точке перед вызовом одного из методов рисования можно вызвать метод `moveTo()`. Это действие аналогично отрыву пера от бумаги и перемещению его на новую позицию.

Используя точку рисунка можно рисовать с помощью серий вызовов методов рисования: `lineTo()` (для рисования прямых линий) и `curveTo()` (для рисования кривых линий).

*Во время рисования можно вызвать метод `moveTo()` в любое время для перемещения точки рисунка на новую позицию без рисования.*

При рисовании, если указан цвет заливки, можно замкнуть заливку, вызвав метод `endFill()`. Если замкнутая фигура не создана (другими словами, если при вызове метода `endFill()` точка рисунка находится не в начальной точке фигуры), то при вызове метода `endFill()` среда выполнения Flash автоматически замыкает фигуру путем проведения прямой линии из текущей точки рисунка в точку, указанную во время последнего вызова метода `moveTo()`. Если процесс заливки запущен, а метод `endFill()` не вызван, вызов метода `beginFill()` (или одного из других методов заливки) завершит текущую заливку и начнет новую.

## Создание прямых линий

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При вызове метода `lineTo()` объект Graphics рисует прямую линию из текущей точки рисунка к координатам, указанным при вызове метода в качестве двух параметров, используя при этом указанный стиль линии. Например, в этой строке кода точка рисунка помещается в точку 100, 100, после чего к точке 200, 200 проводится линия:

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.lineTo(200, 200);
```

Следующий пример иллюстриурет создание красных и зеленых треугольников высотой 100 пикселов:

```
var triangleHeight:uint = 100;
var triangle:Shape = new Shape();

// red triangle, starting at point 0, 0
triangle.graphics.beginFill(0xFF0000);
triangle.graphics.moveTo(triangleHeight / 2, 0);
triangle.graphics.lineTo(triangleHeight, triangleHeight);
triangle.graphics.lineTo(0, triangleHeight);
triangle.graphics.lineTo(triangleHeight / 2, 0);

// green triangle, starting at point 200, 0
triangle.graphics.beginFill(0x00FF00);
triangle.graphics.moveTo(200 + triangleHeight / 2, 0);
triangle.graphics.lineTo(200 + triangleHeight, triangleHeight);
triangle.graphics.lineTo(200, triangleHeight);
triangle.graphics.lineTo(200 + triangleHeight / 2, 0);

this.addChild(triangle);
```

## Создание кривых

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

С помощью метода `curveTo()` создается квадратичная кривая Безье. При этом создается дуга, соединяющая две точки (называемые опорными точками) и одновременно сгибающаяся по направлению к третьей точке (называемой контрольной точкой). Объект Graphics использует текущую позицию рисования в качестве первой опорной точки. При вызове метода `curveTo()` указываются четыре параметра: координаты оси x и y контрольной точки и координаты оси x и y второй опорной точки. Например, с помощью следующего кода создается кривая, начинающаяся в точке 100, 100 и заканчивающаяся в точке 200, 200. Поскольку контрольная точка находится в точке 175, 125, создается кривая, идущая вправо и затем вниз:

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.curveTo(175, 125, 200, 200);
```

В следующем примере описан процесс создания красных и зеленых круглых объектов высотой и шириной 100 пикселов. Следует отметить, что из-за специфики квадратичного уравнения Безье эти круги не являются идеальными:

```
var size:uint = 100;
var roundObject:Shape = new Shape();

// red circular shape
roundObject.graphics.beginFill(0xFF0000);
roundObject.graphics.moveTo(size / 2, 0);
roundObject.graphics.curveTo(size, 0, size, size / 2);
roundObject.graphics.curveTo(size, size, size / 2, size);
roundObject.graphics.curveTo(0, size, 0, size / 2);
roundObject.graphics.curveTo(0, 0, size / 2, 0);

// green circular shape
roundObject.graphics.beginFill(0x00FF00);
roundObject.graphics.moveTo(200 + size / 2, 0);
roundObject.graphics.curveTo(200 + size, 0, 200 + size, size / 2);
roundObject.graphics.curveTo(200 + size, size, 200 + size / 2, size);
roundObject.graphics.curveTo(200, size, 200, size / 2);
roundObject.graphics.curveTo(200, 0, 200 + size / 2, 0);

this.addChild(roundObject);
```

# Создание фигур с помощью встроенных методов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для удобства рисования в ActionScript 3.0 существуют методы для создания таких часто используемых фигур, как круги, эллипсы, прямоугольники и треугольники с закругленными углами. К этим методам относятся следующие методы класса Graphics: `drawCircle()`, `drawEllipse()`, `drawRect()` и `drawRoundRect()`. Эти методы можно использовать вместо методов `lineTo()` и `curveTo()`. Однако, следует отметить, что перед вызовом этих методов требуется указывать стили линий и заливок.

Следующий пример иллюстрирует создание красных, зеленых и синих квадратов высотой и шириной 100 пикселов. Этот код использует метод `drawRect()` и дополнительно указывает, что цвет заливки имеет альфа-значение, равное 50% (0,5):

```
var squareSize:uint = 100;
var square:Shape = new Shape();
square.graphics.beginFill(0xFF0000, 0.5);
square.graphics.drawRect(0, 0, squareSize, squareSize);
square.graphics.beginFill(0x00FF00, 0.5);
square.graphics.drawRect(200, 0, squareSize, squareSize);
square.graphics.beginFill(0x0000FF, 0.5);
square.graphics.drawRect(400, 0, squareSize, squareSize);
square.graphics.endFill();
this.addChild(square);
```

В объекте Sprite или MovieClip содержимое рисунка, созданное с помощью свойства `graphics`, всегда отображается позади всех дочерних объектов отображения, содержащихся в объекте. Кроме того, содержимое свойства `graphics` не является отдельным объектом отображения и отсутствует в списке дочерних объектов объекта Sprite или MovieClip. Например, следующий объект Sprite создает с помощью свойства `graphics` круг, а в списке дочерних объектов отображения содержит объект TextField:

```
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0xFFCC00);
mySprite.graphics.drawCircle(30, 30, 30);
var label:TextField = new TextField();
label.width = 200;
label.text = "They call me mellow yellow...";
label.x = 20;
label.y = 20;
mySprite.addChild(label);
this.addChild(mySprite);
```

Следует отметить, что объект TextField отображается в верхней части круга, созданного с помощью графического объекта.

# Создание линий градиента и заливок

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

С помощью графического объекта также можно создавать штрихи и заливки с градиентами вместо сплошных цветов. Штрих градиента создается с помощью метода `lineGradientStyle()`, а заливка градиента — с помощью метода `beginGradientFill()`.

Оба метода имеют одни и те же параметры. Первые четыре параметра являются обязательными: тип, цвета, альфа-каналы и коэффициенты. Оставшиеся четыре параметра являются необязательными, но могут использоваться при расширенной настройке.

- Первый параметр определяет тип создаваемого градиента. Допустимыми значениями являются `GradientType.LINEAR` или `GradientType.RADIAL`.
- Второй параметр определяет массив цветовых значений, предназначенных для использования. В линейном градиенте цвета будут организованы слева направо. В радиальном градиенте они будут организованы изнутри наружу. Порядок цветов массива задает порядок, в котором цвета будут представлены в градиенте.
- Третий параметр определяет значения альфа-прозрачности соответствующих цветов предыдущего параметра.
- Четвертый параметр определяет коэффициенты или степень насыщенности каждого цвета внутри градиента. Диапазон допустимых значений — от 0 до 255. Эти значения обозначают не длину или ширину, а положение внутри градиента; 0 обозначает начало градиента, 255 — конец градиента. Массив коэффициентов должен возрастать последовательно и иметь число элементов, указанное во втором и третьем параметрах для массивов альфа-каналов и цветов.

Несмотря на факультативность пятого параметра — матрицы преобразования — он часто используется, поскольку обеспечивает простой и эффективный способ управления внешним видом градиента. Этот параметр поддерживает экземпляр Matrix. Наиболее простым способом создания объекта Matrix для градиента является использование метода `createGradientBox()` класса Matrix.

## Определение объекта Matrix для использования с градиентом

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для определения градиентов и использования их в фигурах используйте методы `beginGradientFill()` и `lineGradientStyle()` класса flash.display.Graphics. При определении градиента матрица предоставляется в качестве одного из параметров этих методов.

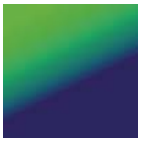
Наиболее простым способом определения матрицы является использование метода `createGradientBox()` класса Matrix, который создает матрицу, используемую для определения градиента. Масштаб, поворот и положение градиента указываются с помощью параметров, передаваемых в метод `createGradientBox()`. Метод `createGradientBox()` поддерживает следующие параметры.

- Ширина окна градиента: ширина (в пикселах), на которую будет распространяться градиент.
- Высота окна градиента: высота (в пикселах), на которую будет распространяться градиент.
- Поворот градиента: поворот (в радианах), применяемый к градиенту.
- Перемещение по горизонтали: на какое расстояние (в пикселах) по горизонтали будет перемещен градиент.
- Перемещение по вертикали: на какое расстояние (в пикселах) по вертикали будет перемещен градиент.

Например, рассмотрим градиент со следующими характеристиками:

- `GradientType.LINEAR`
- Два цвета (зеленый и синий) с массивом `ratios[0, 255]`
- `SpreadMethod.PAD`
- `InterpolationMethod.LINEAR_RGB`

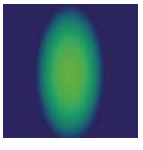
В следующих примерах представлены градиенты, отличающиеся только по параметру `rotation` метода `createGradientBox()`:

| | |
|---|---|
| `width = 100;`<br>`height = 100;`<br>`rotation = 0;`<br>`tx = 0;`<br>`ty = 0;` | |
| `width = 100;`<br>`height = 100;`<br>`rotation = Math.PI/4; // 45°`<br>`tx = 0;`<br>`ty = 0;` |  |
| `width = 100;`<br>`height = 100;`<br>`rotation = Math.PI/2; // 90°`<br>`tx = 0;`<br>`ty = 0;` | |

В следующих примерах показано изменение линейных градиентов (от зеленого к синему), отличающихся только по параметрам `rotation`, `tx` и `ty` метода `createGradientBox()`:

| | |
|---|---|
| `width = 50;`<br>`height = 100;`<br>`rotation = 0;`<br>`tx = 0;`<br>`ty = 0;` | |
| `width = 50;`<br>`height = 100;`<br>`rotation = 0`<br>`tx = 50;`<br>`ty = 0;` | |
| `width = 100;`<br>`height = 50;`<br>`rotation = Math.PI/2; // 90°`<br>`tx = 0;`<br>`ty = 0;` | |
| `width = 100;`<br>`height = 50;`<br>`rotation = Math.PI/2; // 90°`<br>`tx = 0;`<br>`ty = 50;` | |

Параметры `width`, `height`, `tx` и `ty` метода `createGradientBox()` также оказывают влияние на размер и положение заливки градиента *radial*, как это показано в следующем примере:

| | |
|---|---|
| `width = 50;`<br>`height = 100;`<br>`rotation = 0;`<br>`tx = 25;`<br>`ty = 0;` |  |

Радиальный градиент из последнего примера создан посредством следующего кода:

```
import flash.display.Shape;
import flash.display.GradientType;
import flash.geom.Matrix;

var type:String = GradientType.RADIAL;
var colors:Array = [0x00FF00, 0x000088];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var spreadMethod:String = SpreadMethod.PAD;
var interp:String = InterpolationMethod.LINEAR_RGB;
var focalPtRatio:Number = 0;

var matrix:Matrix = new Matrix();
var boxWidth:Number = 50;
var boxHeight:Number = 100;
var boxRotation:Number = Math.PI/2; // 90°
var tx:Number = 25;
var ty:Number = 0;
matrix.createGradientBox(boxWidth, boxHeight, boxRotation, tx, ty);

var square:Shape = new Shape;
square.graphics.beginGradientFill(type,
                           colors,
                           alphas,
                           ratios,
                           matrix,
                           spreadMethod,
                           interp,
                           focalPtRatio);
square.graphics.drawRect(0, 0, 100, 100);
addChild(square);
```

Обратите внимание, что ширина и высота заливки градиента определяются соответствующими параметрами матрицы градиента, а не объектом Graphics. При рисовании с помощью объекта Graphics рисунок создается в координатах матрицы градиента. Даже при использовании одного из методов фигур объекта Graphics, например `drawRect()`, градиент не растягивается до размера созданной фигуры; размер градиента необходимо указать в самой матрице градиента.

В примере представлено визуальное различие между размером матрицы градиента и размером самого рисунка:

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(100, 40, 0, 0, 0);
myShape.graphics.beginGradientFill(GradientType.LINEAR, [0xFF0000, 0x00FF00, 0x0000FF], [1,
1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 50, 40);
myShape.graphics.drawRect(0, 50, 100, 40);
myShape.graphics.drawRect(0, 100, 150, 40);
myShape.graphics.endFill();
this.addChild(myShape);
```

С помощью этого кода создаются три градиента с одинаковым стилем заливки, цвет которой в равной степени содержит красный, зеленый и синей цвета. Градиенты создаются с помощью метода `drawRect()` с шириной 50, 100 и 150 пикселов соответственно. Ширина создаваемой матрицы градиента, указанной в методе `beginGradientFill()`, равна 100 пикселам. Это означает, что первый градиент охватит только половину спектра градиента, второй — весь спектр, третий — весь спектр, а также дополнительные 50 пикселов синего цвета, распространяющегося вправо.

Метод `lineGradientStyle()` аналогичен методу `beginGradientFill()`, за исключением того, что помимо определения градиента перед началом рисования необходимо указать толщину штриха с помощью метода `lineStyle()`. Следующий код создает прямоугольник с красным, зеленым и синим штрихом градиента:

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(200, 40, 0, 0, 0);
myShape.graphics.lineStyle(5, 0);
myShape.graphics.lineGradientStyle(GradientType.LINEAR, [0xFF0000, 0x00FF00, 0x0000FF], [1,
1, 1], [0, 128, 255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 200, 40);
this.addChild(myShape);
```

Дополнительные сведения о классе Matrix см. в разделе ««Использование объектов Matrix» на странице 233».

# Использование класса Math с методами рисования

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

С помощью объекта Graphics можно создавать круги и квадраты, а также и более сложные фигуры, в особенности когда методы рисования используются в сочетании со свойствами и методами класса Math. Класс Math содержит обычные математические константы, такие как `Math.PI` (приблизительно 3,14159265...), константа, используемая для обозначения соотношения длины окружности круга и его диаметра. Этот класс также содержит методы для тригонометрических функций, например `Math.sin()`, `Math.cos()`, `Math.tan()` и другие. Рисование фигур с помощью этих методов и констант создает более динамичные визуальные эффекты, особенно при повторении или рекурсии.

Для многих методов класса Math измерения круглых фигур предполагаются в радианах, а не в градусах. Преобразование между этими двумя единицами измерения является стандартным использованием класса Math:

```
var degrees = 121;
var radians = degrees * Math.PI / 180;
trace(radians) // 2.111848394913139
```

В следующем примере, иллюстрирующем различие методов `Math.sin()` и `Math.cos()` для одинакового значения, показано создание синусоидальной и косинусоидальной волн.

```
var sinWavePosition = 100;
var cosWavePosition = 200;
var sinWaveColor:uint = 0xFF0000;
var cosWaveColor:uint = 0x00FF00;
var waveMultiplier:Number = 10;
var waveStretcher:Number = 5;

var i:uint;
for(i = 1; i < stage.stageWidth; i++)
{
    var sinPosY:Number = Math.sin(i / waveStretcher) * waveMultiplier;
    var cosPosY:Number = Math.cos(i / waveStretcher) * waveMultiplier;

    graphics.beginFill(sinWaveColor);
    graphics.drawRect(i, sinWavePosition + sinPosY, 2, 2);
    graphics.beginFill(cosWaveColor);
    graphics.drawRect(i, cosWavePosition + cosPosY, 2, 2);
}
```

# Анимирование с помощью API рисования

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Одним из преимуществ создания содержимого с помощью API рисования является возможность неоднократного размещения содержимого. Создаваемые рисунки могут изменяться путем ввода и изменения переменных, используемых для рисования. Анимация может создаваться посредством изменения переменных и обновления рисунка либо через несколько кадров, либо с помощью таймера.

Например, следующий код изменяет отображение в каждом передаваемом кадре (посредством прослушивания события `Event.ENTER_FRAME`) с помощью увеличения текущего счета и направляет графический объект на очистку и перерисовку в обновленной позиции.

```
stage.frameRate = 31;

var currentDegrees:Number = 0;
var radius:Number = 40;
var satelliteRadius:Number = 6;

var container:Sprite = new Sprite();
container.x = stage.stageWidth / 2;
container.y = stage.stageHeight / 2;
addChild(container);
var satellite:Shape = new Shape();
container.addChild(satellite);

addEventListener(Event.ENTER_FRAME, doEveryFrame);

function doEveryFrame(event:Event):void
{
    currentDegrees += 4;
    var radians:Number = getRadians(currentDegrees);
    var posX:Number = Math.sin(radians) * radius;
    var posY:Number = Math.cos(radians) * radius;
    satellite.graphics.clear();
    satellite.graphics.beginFill(0);
    satellite.graphics.drawCircle(posX, posY, satelliteRadius);
}
function getRadians(degrees:Number):Number
{
return degrees * Math.PI / 180;
}
```

Для получения существенно отличающихся результатов можно поэкспериментировать, изменяя исходные переменные в начале кода: `currentDegrees`, `radius` и `satelliteRadius`. Например, попробуйте уменьшить переменную radius и/или увеличить переменную totalSatellites. Это только один из примеров, демонстрирующий, как API рисования упрощает процесс создания сложных визуальных объектов отображения.

# Пример API рисования: наглядный алгоритмический генератор

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Пример наглядного алгоритмического генератора динамически рисует в рабочей области несколько так называемых «спутников», т.е. кругов, двигающихся по круговой орбите. К числу используемых функций относятся следующие:

- использование API рисования для создания основной фигуры с динамическими свойствами;
- подключение пользователя к свойствам, используемым при рисовании;
- перемещение анимации путем очищения рабочей области каждого кадра и последующей перерисовки.

В примере из предыдущего подраздела показана анимация «спутника», выполненная с помощью события `Event.ENTER_FRAME`. Этот пример расширяет предыдущий пример, отображая панель управления с набором ползунков, которые немедленно обновляют отображение нескольких спутников. В данном примере представлена формализация кода во внешние классы и объединение процесса создания спутника в цикл с сохранением ссылки на каждый спутник в массиве `satellites`.

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы с образцами приложения находятся в папке Samples/AlgorithmicVisualGenerator. Папка содержит следующие файлы.

| File | Описание |
|---|---|
| AlgorithmicVisualGenerator.fla | Основной файл приложения в Flash Professional (FLA). |
| com/example/programmingas3/algorithmic/AlgorithmicVisualGenerator.as | Класс, предоставляющий основные функциональные возможности приложения, включая создание спутников в рабочей области и ответ на события с панели управления для обновления переменных, оказывающих влияние на процесс создания спутников. |
| com/example/programmingas3/algorithmic/ControlPanel.as | Класс, управляющий использованием нескольких ползунков и отправляющий возникающие события. |
| com/example/programmingas3/algorithmic/Satellite.as | Класс, представляющий объект отображения, вращающийся по орбите вокруг центральной точки и содержащий свойства, относящиеся к его текущему состоянию рисования. |

## Установка прослушивателей

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Сначала приложение создает три прослушивателя. Первый прослушивает отправленное с панели управления событие о необходимости обновления спутников. Второй прослушивает изменения размера рабочей области SWF-файла. Третий прослушивает каждый проходящий кадр SWF-файла, используя для перерисовки функцию `doEveryFrame()`.

## Создание спутников

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

По окончании установки прослушивателей, вызывается функция `build()`. Эта функция сначала вызывает функцию `clear()`, очищающую массив `satellites` и удаляющую все предыдущие рисунки из рабочей области. Это необходимо, т.к. функция `build()` вызывается повторно всякий раз, когда с панели управления отсылается событие о необходимости выполнения этого действия, например, при изменении настроек цветов. В этом случае спутники необходимо удалить и создать заново.

После этого функция создает спутники путем настройки требуемых исходных свойств, таких как переменная `position`, начинающаяся с произвольной позиции на орбите, и переменная `color`, которая в этом примере не изменяется влоть до момента создания спутника.

По окончании создания к каждому спутнику в массиве `satellites` добавляется ссылка. Вызов функции `doEveryFrame()` обновляет все спутники в данном массиве.

## Обновление позиции спутника

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Функция `doEveryFrame()` является основной для процесса анимации приложения. Она вызывается для каждого кадра с коэффициентом, равным коэффициенту кадров SWF-файла. Так как переменные рисунка меняются незначительно, у пользователя возникает ощущение просмотра анимации.

Эта функция сначала удаляет все предыдущие рисунки и перерисовывает фон. Затем она объединяет в цикл все контейнеры спутников, увеличивает для каждого спутника значение свойства `position`, а также обновляет свойства `radius` и `orbitRadius`, которые можно изменить с помощью панели управления. Наконец, с помощью вызова метода `draw()` класса Satellite выполняется обновление спутника и его перемещение на новую позицию.

Следует отметить, что счетчик `i` увеличивает значения только до переменной `visibleSatellites`. Это происходит по той причине, что когда пользователь с панели управления ограничивает число отображаемых спутников, оставшиеся в цикле спутники не должны повторно создаваться, а вместо этого должны быть скрыты. Это происходит в цикле, идущем сразу же после цикла, отвечающего за рисование.

По завершении выполнения функции doEveryFrame() происходит обновление числа `visibleSatellites` на всем экране.

## Ответная реакция на действия пользователя

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Пользователь взаимодействует с системой через панель управления, управляемую классом ControlPanel. Данный класс устанавливает прослушиватель вместе с индивидуальным для каждого ползунка минимальным, максимальным значением, а также значением по умолчанию. При перемещении пользователем этих ползунков вызывается функция `changeSetting()`. Эта функция обновляет свойства панели управления. Если для изменения требуется повторное создание отображения, отправляется событие, которое впоследствии обрабатывается в основном файле приложения. Как только настройки панели управления изменены, функция `doEveryFrame()` создает каждый спутник с применением обновленных переменных.

## Дальнейшие настройки

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Этот пример является только базовой схемой, представляющей способ создания визуальных объектов с помощью API рисования. В целом, в этом примере участвует несколько строк кода для создания довольно сложной интерактивной среды. Тем не менее, рамки этого примера можно расширить с минимальными изменениями. Вот некоторые из способов.

- С помощью функции `doEveryFrame()` можно увеличить значение цвета спутника.
- С помощью функции `doEveryFrame()` можно уменьшить или увеличить радиус спутника.
- Не обязательно, чтобы радиус спутника имел форму окружности; например, для перемещения в соответствии с синусоидальной волной может использоваться класс Math.
- Спутники могут использовать функцию определения столкновений с другими спутниками.

API рисования можно использовать в качестве варианта для создания визуальных эффектов в среде разработок Flash, а также для создания базовых фигур во время выполнения. Его также можно использовать для создания многочисленных визуальных эффектов, которые невозможно создать вручную. Благодаря использованию API рисования и математики пользователь ActionScript обладает всем для реализации большого количества неожиданных и оригинальных идей.

# Расширенное использование API рисования

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Среды выполнения Flash Player 10, Adobe AIR 1.5 и более поздние версии сред выполнения Flash поддерживают расширенный набор функций рисования. Новые функции API-интерфейса рисования расширяют методы рисования из предыдущих версий и позволяют создавать наборы данных для рисования фигур, изменять их во время выполнения и создавать трехмерные эффекты. Новые функции рисования API объединяют существующие методы в альтернативные команды. Для предоставления наборов данных для методов рисования эти команды используют векторные массивы и классы перечисления. Векторные массивы позволяют ускорить визуализацию более сложных фигур, и разработчики могут изменять значения массива с помощью кода для динамической визуализации фигур во время выполнения.

Функции рисования, введенные в проигрывателе Flash Player 10, описываются в следующих разделах: «Создание контуров» на странице 254, «Определение правил поворотов» на странице 255, «Использование классов графических данных» на странице 257 и «Об использовании drawTriangles()» на странице 260.

В ActionScript с использованием расширенного API рисования обычно выполняются следующие задачи:

- использование объектов Vector для хранения данных для методов рисования;
- определение контуров для программного рисования фигур за одну операцию;
- создание правил поворотов для определения способа заливки накладывающихся фигур;
- запрос векторных графических данных какого-либо объекта отображения: например, для сериализации или сохранения графических данных, для создания листа спрайтов во время выполнения или для рисования копии этой векторной графики;
- использование треугольников и методов рисования для создания трехмерных эффектов.

**Важные понятия и термины**

Ниже приводится список важных терминов, встречаемых в этом разделе.

- Vector — массив значений, принадлежащих к одному и тому же типу данных. Объект Vector может сохранять массив значений, используемых методами рисования для построения линий и фигур с помощью одной команды. Дополнительные сведения об объектах Vector см. в разделе ««Индексные массивы» на странице 28».
- Контур состоит из одного или нескольких сегментов прямой или кривой. Начало и конец каждого сегмента отмечаются координатами, фиксирующими данную кривую на определенном месте. Контур может быть замкнутым (например, круг) или открытым с отдельными конечными точками (например, волнообразная линия).
- Поворот — направление контура, интерпретируемое визуализатором. Он может быть положительным (по часовой стрелке) или отрицательным (против часовой стрелки).

- GraphicsStroke: класс для настройки стиля линий. Несмотря на то, что понятие «штрих» не является компонентом расширенных функций API-интерфейса рисования, данный класс вполне может использоваться для определения стиля линии и ее свойства заливки в новом API-интерфейсе рисования. Стиль линии можно настраивать динамически с помощью класса GraphicsStroke.
- Объект Fill — объекты, созданные с помощью новых классов отображения, например, flash.display.GraphicsBitmapFill и flash.display.GraphicsGradientFill, которые передаются команде рисования `Graphics.drawGraphicsData()`. Объекты Fill и новые команды рисования представляют собой более объектно-ориентированный подход программирования к дублированию `Graphics.beginBitmapFill()` и `Graphics.beginGradientFill()`.

## Создание контуров

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

В разделе о рисовании линий и кривых (см. «Создание линий и кривых» на странице 240) представлены команды для рисования одной линии (`Graphics.lineTo()`) или кривой (`Graphics.curveTo()`), а затем для перемещения линии в другую точку (`Graphics.moveTo()`) с целью построения фигуры. Методы `Graphics.drawPath()` и `Graphics.drawTriangles()` принимают в качестве параметра набор объектов, представляющих собой вышеописанные команды рисования. Этими методами можно указать последовательность команд `Graphics.lineTo()`, `Graphics.curveTo()` или `Graphics.moveTo()` для выполнения в среде Flash одной инструкцией.

Класс перечисления GraphicsPathCommand задает набор констант, соответствующих командам рисования. Последовательность этих констант (заключенную в экземпляр Vector) можно передать в качестве параметра метода `Graphics.drawPath()`. Затем с помощью одной команды можно выполнить визуализацию всей фигуры или сразу нескольких фигур. Также можно изменять значения, передаваемые этим методам, для изменения существующей фигуры.

В дополнение к командам рисования в виде Vector, методу `drawPath()` требуется набор координат, соответствующих координатам для каждой из команд рисования. Следует создать экземпляр Vector, содержащий координаты (экземпляры Number) и передать его методу `drawPath()` в качестве второго аргумента (`data`).

***Примечание.*** *Значения в объекте Vector не являются объектами Point. Вектор — это последовательность чисел, где каждая группа из двух чисел представляет пару координат x и y.*

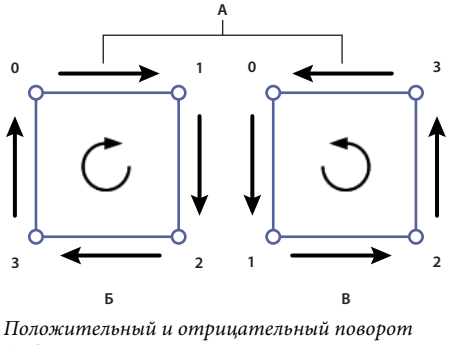
Метод `Graphics.drawPath()` сопоставляет каждую команду с соответствующими значениями точек (набор из двух или четырех чисел) для создания контура в объекте Graphics:

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

    public class DrawPathExample extends Sprite
    {
        public function DrawPathExample(){

            var squareCommands:Vector.<int> = new Vector.<int>(5, true);
            squareCommands[0] = GraphicsPathCommand.MOVE_TO;
            squareCommands[1] = GraphicsPathCommand.LINE_TO;
            squareCommands[2] = GraphicsPathCommand.LINE_TO;
            squareCommands[3] = GraphicsPathCommand.LINE_TO;
            squareCommands[4] = GraphicsPathCommand.LINE_TO;

            var squareCoord:Vector.<Number> = new Vector.<Number>(10, true);
            squareCoord[0] = 20; //x
            squareCoord[1] = 10; //y
            squareCoord[2] = 50;
            squareCoord[3] = 10;
            squareCoord[4] = 50;
            squareCoord[5] = 40;
            squareCoord[6] = 20;
            squareCoord[7] = 40;
            squareCoord[8] = 20;
            squareCoord[9] = 10;

            graphics.beginFill(0x442266);//set the color
            graphics.drawPath(squareCommands, squareCoord);
        }
    }
}
```

## Определение правил поворотов

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

В усовершенствованном API-интерфейсе рисования представлена концепция «закручивания» пути: направления пути. Поворот контура может быть положительным (по часовой стрелке) или отрицательным (против часовой стрелки). Поворот определяет порядок, в котором средство визуализации интерпретирует координаты, предоставленные вектором для параметра data.




*Положительный и отрицательный поворот*
***А.*** *Стрелки указывают направление рисования* ***Б.*** *Положительный поворот (по часовой стрелке)* ***В.*** *Отрицательный поворот (против часовой стрелки)*

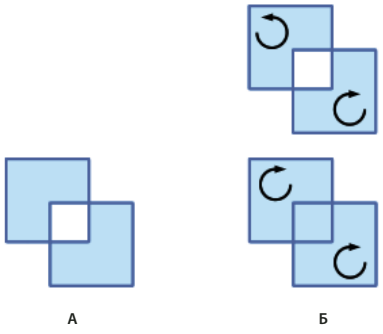
Кроме того, метод `Graphics.drawPath()` имеет дополнительный третий параметр winding.

```
drawPath(commands:Vector.<int>, data:Vector.<Number>, winding:String = "evenOdd"):void
```

В данном контексте третий параметр является строкой или константой, указывающей поворот или правило заливки для пересекающихся контуров. (Константы определяются в классе GraphicsPathWinding как `GraphicsPathWinding.EVEN_ODD` или `GraphicsPathWinding.NON_ZERO`.) Правило поворота играет важную роль при пересечении контуров.

Правило «четный-нечетный» является стандартным правилом поворота, используемым всеми предыдущими API рисования. Правило «четный-нечетный» также является правилом по умолчанию для метода `Graphics.drawPath()`. Согласно правилу поворота «четный-нечетный» пересекающиеся контуры чередуют открытые и закрытые заливки. При пересечении двух квадратов с одинаковой заливкой область пересечения остается без заливки. Как правило, одна из рядом стоящих областей заполнена заливкой, а другая — нет.

С другой стороны, правило «ненулевого значения» учитывает поворот (направление рисования) при определении необходимости заполнения заливкой области, ограниченной пересекающимися контурами. При пересечении контуров с противоположным поворотом, определенная область остается незаполненной как при использовании правила «четный-нечетный». Для контуров с одинаковым поворотом, незаполняемая область заполняется заливкой.

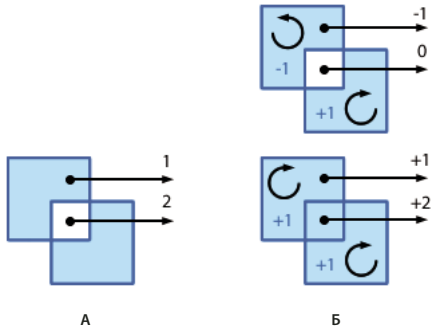



*Правила поворота для пересекающихся областей*
***А.** Правило поворота «четный-нечетный»* ***Б.** Правило поворота «ненулевого значения»*

## Имена правил поворота

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Имена ссылаются на более конкретное правило, определяющее способ управления заливками. Контурам с положительным поворотом присваивается значение +1; а контурам с отрицательным поворотом — значение -1. Требуется выбрать точку, находящуюся в замкнутой области фигуры, и нарисовать бесконечную линию с началом в этой точке. Заливка определяется в соответствии с числом пересечений линии с контуром, а также с учетом объединенных значений этих контуров. При использовании правила поворота «четный-нечетный» учитывается число пересечений линии с контуром. При нечетном числе область заполняется заливкой. При четном числе область остается без заливки. При использовании правила «ненулевого значения» учитываются значения, присвоенные контурам. Если объединенные значения не равны нулю, область заполняется заливкой. Если объединенное значение равно нулю, область не заполняется заливкой.




*Счет и заливки для правила поворота*
***А.*** *Правило поворота «четный-нечетный»* ***Б.*** *Правило поворота «ненулевого значения»*

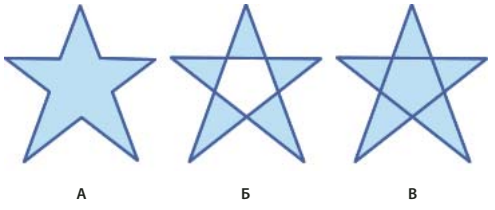
### Применение правил поворота

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Несмотря на сложность правил заливки, в некоторых ситуациях они необходимы. Например, требуется нарисовать звезду. Согласно стандартному правилу «четный-нечетный», для построения фигуры требуется десять разных линий. Согласно правилу поворота «ненулевого значения» вместо десяти линий используется только пять. Следующий код ActionScript рисует звезду с использованием пяти линий и правила поворота «ненулевого значения».

```
graphics.beginFill(0x60A0FF);
graphics.drawPath( Vector.<int>([1,2,2,2,2]),  Vector.<Number>([66,10, 23,127, 122,50, 10,49,
109,127]),   GraphicsPathWinding.NON_ZERO);
```

На рисунке представлена полученная звезда.




*Звезда, нарисованная с использованием разных правил поворота*
***А.*** *«Четный-нечетный»: 10 линий* ***Б.*** *«Четный-нечетный»: 5 линий* ***В.*** *«Ненулевое значение»: 5 линий*

При анимации изображений или использовании их в качестве текстур для трехмерных объектов и наложении правила поворота играют еще более значимую роль.

## Использование классов графических данных

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Расширенный API рисования включает набор классов в пакете flash.display, реализующих интерфейс IGraphicsData. Эти классы играют роль объектов-значений (контейнеров данных), представляющих собой методы API рисования.

Интерфейс IGraphicsData реализуют следующие классы:

- GraphicsBitmapFill

- GraphicsEndFill
- GraphicsGradientFill
- GraphicsPath
- GraphicsShaderFill
- GraphicsSolidFill
- GraphicsStroke
- GraphicsTrianglePath

Эти классы позволяют сохранять готовые рисунки в объекте Vector типа IGraphicsData (Vector.<IGraphicsData>). Графические данные затем можно повторно использовать в качестве исходных данных для других фигур или сохранить на будущее.

Обратите внимание, что имеется несколько классов заливки, представляющих собой разные стили, но класс штриха только один. В ActionScript предусмотрен только один класс штриха с интерфейсом IGraphicsData, поскольку для определения стиля класс штриха использует классы заливки. Поэтому каждый штрих на самом деле задается комбинацией класса штриха и класса заливки. Помимо этого, API-интерфейс для этих классов графических данных повторяет методы, представленные в классах flash.display.Graphics:

| **Метод класса Graphics** | **Соответствующий класс** |
|---|---|
| beginBitmapFill() | GraphicsBitmapFill |
| beginFill() | GraphicsSolidFill |
| beginGradientFill() | GraphicsGradientFill |
| beginShaderFill() | GraphicsShaderFill |
| lineBitmapStyle() | GraphicsStroke + GraphicsBitmapFill |
| lineGradientStyle() | GraphicsStroke + GraphicsGradientFill |
| lineShaderStyle() | GraphicsStroke + GraphicsShaderFill |
| lineStyle() | GraphicsStroke + GraphicsSolidFill |
| moveTo()<br>lineTo()<br>curveTo()<br>drawPath() | GraphicsPath |
| drawTriangles() | GraphicsTrianglePath |

Кроме того, класс GraphicsPath имеет собственные служебные методы `GraphicsPath.moveTo()`, `GraphicsPath.lineTo()`, `GraphicsPath.curveTo()`, `GraphicsPath.wideLineTo()` и `GraphicsPath.wideMoveTo()`, упрощающие определение этих команд для экземпляра GraphicsPath. Эти служебные методы упрощают непосредственное определение или обновление команд и значений данных.

## Рисование на основе векторных графических данных

Создав набор экземпляров IGraphicsData, можно начинать визуализацию графики с помощью метода `drawGraphicsData()` класса Graphics. Метод `drawGraphicsData()` выполняет набор инструкций рисования из векторного массива экземпляров IGraphicsData в последовательном порядке:

```
// stroke object
var stroke:GraphicsStroke = new GraphicsStroke(3);
stroke.joints = JointStyle.MITER;
stroke.fill = new GraphicsSolidFill(0x102020);// solid stroke

// fill object
var fill:GraphicsGradientFill = new GraphicsGradientFill();
fill.colors = [0x0000FF, 0xEEFFEE];
fill.matrix = new Matrix();
fill.matrix.createGradientBox(70, 70, Math.PI/2);
// path object
var path:GraphicsPath = new GraphicsPath(new Vector.<int>(), new Vector.<Number>());
path.commands.push(GraphicsPathCommand.MOVE_TO, GraphicsPathCommand.LINE_TO,
GraphicsPathCommand.LINE_TO);
path.data.push(125,0, 50,100, 175,0);

// combine objects for complete drawing
var drawing:Vector.<IGraphicsData> = new Vector.<IGraphicsData>();
drawing.push(stroke, fill, path);

// draw the drawing
graphics.drawGraphicsData(drawing);
```

Изменяя одно значение в контуре, используемом рисунком в примере, можно нарисовать фигуру несколько раз для получения более сложного изображения.

```
// draw the drawing multiple times
// change one value to modify each variation
graphics.drawGraphicsData(drawing);
path.data[2] += 200;
graphics.drawGraphicsData(drawing);
path.data[2] -= 150;
graphics.drawGraphicsData(drawing);
path.data[2] += 100;
graphics.drawGraphicsData(drawing);
path.data[2] -= 50;graphicsS.drawGraphicsData(drawing);
```

Хотя объекты IGraphicsData могут определять стили заливки и обводки, использование этих стилей не является обязательным. Другими словами, методы класса Graphics можно использовать для определения стилей, а объекты IGraphicsData — для рисования сохраненной коллекции контуров и наоборот.

***Примечание.*** *Метод `Graphics.clear()` используется для удаления предыдущего рисунка перед началом выполнения нового, если не предусмотрено дополнение к исходному рисунку, как показано в примере выше. При изменении одной части контура или коллекции объектов IGraphicsData, требуется нарисовать заново весь рисунок, чтобы увидеть изменения.*

При использовании классов графических данных заливка визуализируется всегда, когда рисуется три или более точек, так как на этом этапе фигура по определению имеет закрытый контур. Несмотря на то, что заливка закрывается, обводка остается открытой, и такое поведение отличается от поведения при использовании нескольких команд `Graphics.lineTo()` и `Graphics.moveTo()`.

### Чтение векторных графических данных

**Flash Player 11.6 и более поздних версий, Adobe AIR 3.6 и более поздних версий**

Помимо рисования объекта отображения на основе содержимого векторного массива, класс Graphics позволяет получать представление векторной графики любого объекта отображения при помощи метода `readGraphicsData()` (в среде Flash Player 11.6 или Adobe AIR 3.6 и более поздних версий). Это удобно для получения «мгновенного слепка» графического объекта, что позволяет сохранить, скопировать объект, создать лист спрайтов во время выполнения и многое другое.

Вызов метода `readGraphicsData()` возвращает экземпляр Vector, содержащий объекты типа IGraphicsData. Это те самые объекты, что использовались для рисования векторной графики при помощи метода `drawGraphicsData()`.

Имеется несколько ограничений на применение метода `readGraphicsData()` для чтения данных векторной графики. Дополнительные сведения см. в разделе readGraphicsData() справочника по языку ActionScript.

## Об использовании drawTriangles()

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Еще один усовершенствованный метод, `Graphics.drawTriangles()`, введенный в Flash Player 10 и Adobe AIR 1.5, похож на метод `Graphics.drawPath()`. Метод `Graphics.drawTriangles()` также использует объект Vector.<Number> для указания местоположения точек для создания контура.

Однако в действительности метод `Graphics.drawTriangles()` предназначен для упрощения создания трехмерных эффектов с помощью ActionScript. Сведения об использовании метода `Graphics.drawTriangles()` для создания трехмерных эффектов см. в разделе ««Использование треугольников для создания трехмерных эффектов» на странице 392».

# Глава 13. Работа с растровыми изображениями

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Помимо векторной графики ActionScript 3.0 позволяет создавать растровые изображения и управлять данными пикселов внешних растровых изображений, загружаемых в SWF. Благодаря возможности доступа и изменения отдельных значений пикселов можно создавать собственные эффекты наподобие фильтров и использовать встроенные функции шумов для создания текстур и случайных шумов.

- Ренаун Эриксон (Renaun Erickson). Визуализация активов игр в ActionScript с использованием методов блитирования
- Программирование растровых изображений. Колин Мук (Colin Moock) Основы ActionScript 3. Глава 26. (O'Reilly Media, 2007)
- Майк Джонс (Mike Jones). Работа со спрайтами в модуле Pushbutton
- Flash & Math: упрощение работы с пиксельными частицами
- Flixel

## Основы работы с растровыми изображениями

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При работе с цифровой графикой вы обычно сталкиваетесь с двумя основными ее типами: растровыми и векторными изображениями. Растровые изображения, также известные как растровая графика, состоят из мельчайших квадратиков (пикселов), организованных в виде особой прямоугольной сетки. Векторные изображения состоят из математически генерируемых геометрических фигур: линий, кривых и многоугольников.

Растровые изображения характеризуются шириной и высотой в пикселах и количеством бит на пиксел, указывающим, какое количество цветов может содержаться в одном пикселе. Если растровое изображение использует цветовую модель RGB, каждый пиксел содержит три байта: красный, зеленый и синий. Каждый из этих байтов имеет значение от 0 до 255. При смешении байтов в пикселе они, как на палитре художника, образуют определенный цвет. Например, чтобы пиксел был ярко-оранжевым, нужно присвоить ему следующие значения: красный = 255, зеленый = 102, синий = 0.

Качество растрового изображения определяется разрешением и глубиной цвета. *Разрешение* означает количество пикселов в изображении. Чем больше пикселов, тем выше разрешение и тем четче изображение. *Глубина цвета* определяет объем информации, содержащийся в каждом пикселе. Например, изображение с глубиной цвета 16 бит на пиксел не сможет передать то количество цветов, которое передает изображение с глубиной цвета 48 бит. Поэтому изображение с глубиной цвета 48 бит будет более мягко передавать оттенки, чем изображение с глубиной цвета 16 бит.

Так как качество растровых изображений зависит от разрешения, они не очень хорошо масштабируются. Это особенно заметно, если попытаться увеличить картинку в размере. При увеличении размера изображения обычно теряется качество и четкость.

### Форматы растровых изображений

Файлы растровых изображений бывают разных форматов. Все они используют разные алгоритмы сжатия для уменьшения размера файла и оптимизации качества изображения в зависимости от конечной цели. Форматы растрового изображения, поддерживаемые приложениями Adobe — BMP, GIF, JPG, PNG и TIFF.

#### BMP

Формат BMP (точечный рисунок) является графическим форматом по умолчанию в операционной системе Microsoft Windows. Он не использует алгоритмов сжатия, и поэтому файлы этого формата обычно имеют большой размер.

#### GIF

Формат обмена графическими данными GIF (англ. Graphics Interchange Format) был разработан в 1987 году компанией CompuServe для передачи 256-цветных изображений (с глубиной цвета 8 бит). Файлы в этом формате получаются небольшого размера, поэтому он идеален для публикации изображений на веб-страницах. Из-за ограниченной цветовой палитры формата GIF он не подходит для фотографий, где обычно требуется хорошая передача полутонов и высокая четкость. Формат GIF поддерживает однобитовую прозрачность, что позволяет делать цвет чистым (прозрачным). Благодаря этому фон веб-страницы может проступать через прозрачные области изображения.

#### JPEG

Формат JPEG (часто записывается как JPG), разработанный Объединенной экспертной группой в области фотографии (англ. Joint Photographic Experts Group), использует алгоритм сжатия с потерей качества, благодаря чему при глубине цвета 24 бита удается сохранить размер файла небольшим. Сжатие с потерей качества означает, что каждый раз при сохранении изображения теряется качество и данные, но в результате получается небольшой размер файла. Формат JPEG идеален для фотографий, потому что он поддерживает передачу миллионов цветов. Возможность выбирать степень сжатия изображения позволяет находить баланс между качеством изображения и размером файла.

#### PNG

Формат переносимой сетевой графики PNG (англ. Portable Network Graphics) был разработан как формат с открытым исходным кодом, альтернативный запатентованному GIF. Формат PNG поддерживает глубину цвета до 64 бит, что позволяет передавать до 16 миллионов цветов. Так как PNG является относительно новым форматом, некоторые ранние обозреватели его не поддерживают. В отличие от JPG, PNG использует сжатие без потери качества, т.е. при сохранении изображения данные не теряются. PNG-файлы также поддерживают альфа-прозрачность, обеспечивающую до 256 уровней прозрачности.

#### TIFF

Формат изображения с тегами TIFF (Tagged Image File Format) использовался для работы на разных платформах до появления PNG. Недостатком формата TIFF было наличие его многочисленных вариантов и отсутствие единой программы чтения для обработки всех версий. Кроме того, в настоящее время этот формат не поддерживается ни одним обозревателем. Формат TIFF может использовать сжатие с потерей и без потери качества, а также позволяет обрабатывать пространства цветов устройства (такие как CMYK).

### Прозрачные и непрозрачные растровые изображения

В растровых изображениях в форматах GIF и PNG каждый пиксел может иметь дополнительный байт (альфа-канал). Этот дополнительный байт отвечает за прозрачность пиксела.

Изображения в формате GIF поддерживают однобитовую прозрачность, что позволяет делать один из 256 цветов палитры прозрачным. PNG-изображения, в свою очередь, поддерживают до 256 уровней прозрачности. Эта функция особенно полезна, когда требуется, чтобы изображения или текст сливались с фоном.

В ActionScript 3.0 дополнительный пиксел, отвечающий за прозрачность, представлен классом BitmapData. Как и в модели прозрачности PNG, ActionScript обеспечивает до 256 уровней прозрачности.

**Важные понятия и термины**

Ниже приводится список важных терминов, встречающихся в этой главе при описании растровой графики:

**Альфа** Уровень прозрачности (или, точнее, непрозрачности) цвета в изображении. Величину альфа часто называют значением *альфа-канала*.

**Цвет ARGB** Цветовая схема, где каждый пиксел является сочетанием значений красного, зеленого и синего цветов, а его прозрачность задается значением альфа.

**Цветовой канал** Цвета принято обозначать сочетанием нескольких основных цветов (в компьютерной графике это обычно красный, зеленый и синий). Каждый основной цвет считается цветовым каналом, а количество того или иного цвета в каждом цветовом канале, смешанном с другими каналами, определяет итоговый оттенок.

**Глубина цвета** Также называется *глубиной в битах*. Объем компьютерной памяти, выделяемой на каждый пиксел, что в свою очередь определяет максимально возможное число цветов в изображении.

**Пиксел** Наименьшая единица информации в растровом изображении, цветная точка.

**Разрешение** Размер изображения в пикселах, определяющий степень его детализации. Разрешение часто характеризируют шириной и высотой изображения в пикселах.

**Цвет RGB** Цветовая схема, в которой цвет каждого пиксела представлен сочетанием красного, зеленого и синего.

# Классы Bitmap и BitmapData

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Основными классами ActionScript 3.0 для работы с растровыми изображениями являются класс Bitmap, который используется для отображения растровых изображений на экране, и класс BitmapData, который используется для доступа к данным несжатых изображений и манипуляций с ними.

## Дополнительные разделы справки

flash.display.Bitmap

flash.display.BitmapData

## Объяснение класса Bitmap

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Являясь подклассом класса DisplayObject, Bitmap представляет собой основной класс ActionScript 3.0, который используется для вывода растровых изображений. Эти изображения, возможно, были загружены через класс flash.display.Loader или созданы динамически с помощью конструктора `Bitmap()`. При загрузке изображения из внешнего источника объект Bitmap может использовать только форматы изображения GIF, JPEG или PNG. После создания экземпляр Bitmap считается оболочкой для объекта BitmapData, который необходимо визуализировать в рабочей области. Так как экземпляр Bitmap является экранным объектом, для управления им можно использовать любые характеристики и функции экранных объектов. Дополнительную информацию о работе с экранными объектами см. в разделе «Программирование отображаемого содержимого» на странице 162

## Привязка и сглаживание пикселов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В дополнение к обычной функциональности экранных объектов класс Bitmap обладает некоторыми специфическими особенностями, относящимися к растровым изображениям.

Свойство `pixelSnapping` класса Bitmap определяет, привязан ли объект Bitmap к ближайшему пикселу. Это свойство принимает одну из трех констант, определенных в классе PixelSnapping: `ALWAYS`, `AUTO` или `NEVER`.

Ниже показан синтаксис применения привязки к пикселу:

```
myBitmap.pixelSnapping = PixelSnapping.ALWAYS;
```

Масштабирование растровых изображений часто приводит к потере четкости и искажениям. Для уменьшения искажений пользуйтесь свойством `smoothing` класса BitmapData. Если этому логическому свойству присвоено значение `true`, то при масштабировании пикселы изображения сглаживаются. Это помогает сохранить изображение более четким и естественным.

## Объяснение класса BitmapData

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс BitmapData, входящий в пакет flash.display, можно сравнить с фотоснимком пикселов загруженного или динамически созданного растрового изображения. Этот снимок представлен массивом данных о пикселах внутри объекта. Класс BitmapData также содержит ряд встроенных методов, которые могут быть полезны для создания и управления данными пикселов.

Для создания экземпляра объекта BitmapData используйте следующий код:

```
var myBitmap:BitmapData = new BitmapData(width:Number, height:Number, transparent:Boolean,
fillColor:uinit);
```

Параметры `width` и `height` определяют размеры растрового изображения. Начиная с версий AIR 3 и Flash Player 11 ограничения по размеру для объекта BitmapData были удалены. Максимальный размер растрового изображения зависит от операционной системы.

В AIR 1.5 и Flash Player 10 максимальный размер объекта BitmapData составляет 8,191 пикселя в ширину или высоту, а общее количество пикселей не может превышать 16,777,215. (Т. е. если ширина объекта BitmapData составляет 8,191 пикселя, его высота не может быть больше 2 048 пикселей.) В проигрывателе Flash Player 9 и более ранних версий, а также в AIR 1.1 и более ранних версий, это ограничение составляет 2 880 пикселей в высоту и 2 880 пикселей в ширину.

Параметр `transparent` указывает, содержит растровое изображение альфа-канал (`true`) или нет (`false`). Параметр `fillColor` — это 32-битная величина, задающая цвет фона и степень прозрачности (если ей присвоено значение `true`). В примере ниже показано, как создать объект BitmapData с оранжевым фоном 50-процентной прозрачности:

```
var myBitmap:BitmapData = new BitmapData(150, 150, true, 0x80FF3300);
```

Для визуализации на экране вновь созданного объекта BitmapData назначьте его экземпляру Bitmap или сделайте экземпляр Bitmap его оболочкой. Для этого нужно либо передать объект BitmapData как параметр конструктора объекта Bitmap, либо назначить его свойству `bitmapData` существующего экземпляра Bitmap. Также необходимо добавить экземпляр растрового изображения к списку отображения, вызвав метод `addChild()` или `addChildAt()` контейнера экранного объекта, содержащего экземпляр Bitmap. Дополнительную информацию о работе со списком отображения см. в разделе «Добавление экранных объектов в список отображения» на странице 171.

В примере ниже показано, как создавать объект BitmapData с красной заливкой и отображать его в экземпляре Bitmap:

```
var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false, 0xFF0000);
var myImage:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myImage);
```

# Управление пикселами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс BitmapData содержит набор методов, позволяющих управлять значениями данных пикселов.

## Управление отдельными пикселами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При изменении вида растрового изображения на уровне пикселов необходимо в первую очередь получить значения цветов пикселов в той области, с которой вы хотите работать. Для получения этих значений используйте метод `getPixel()`.

Метод `getPixel()` возвращает значение RGB по набору координат пиксела x, y, которые передаются как параметр. Если какие-либо пикселы, с которыми вы работаете, содержат прозрачность (альфа-канал), используйте метод `getPixel32()`. Этот метод также возвращает значение RGB, но в отличие от метода `getPixel()` значение, возвращаемое `getPixel32()`, содержит дополнительные данные, представляющие значение альфа-канала (прозрачности) данного пиксела.

Если же вы хотите просто изменить цвет или прозрачность пиксела растрового изображения, воспользуйтесь методом `setPixel()` или `setPixel32()`. Для задания цвета пиксела просто передайте координаты x, y и значение цвета одному из этих методов.

В примере ниже показано, как с помощью метода `setPixel()` нарисовать крестик на зеленом фоне. Затем метод `getPixel()` возвращает значение цвета пиксела с координатами 50, 50 и отрисовывает его.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 100, false, 0x009900);

for (var i:uint = 0; i < 100; i++)
{
    var red:uint = 0xFF0000;
    myBitmapData.setPixel(50, i, red);
    myBitmapData.setPixel(i, 50, red);
}

var myBitmapImage:Bitmap = new Bitmap(myBitmapData);
addChild(myBitmapImage);

var pixelValue:uint = myBitmapData.getPixel(50, 50);
trace(pixelValue.toString(16));
```

Если необходимо считать значение группы пикселов, а не отдельного пиксела, следует использовать метод `getPixels()`. Этот метод создает массив байтов из прямоугольной области данных пикселов, передаваемых как параметр. Каждый из элементов массива байтов (иными словами, значений пикселов) является целым числом без знака — 32-битным значением пиксела без множителя.

И наоборот, для изменения (или задания) значения группе пикселов используйте метод `setPixels()`. Этот метод подразумевает наличие двух параметров (`rect` и `inputByteArray`), при сочетании которых получается прямоугольная область (`rect`) данных пикселов (`inputByteArray`).

По мере чтения (и записи) данных из `inputByteArray` для каждого пиксела массива вызывается метод `ByteArray.readUnsignedInt()`. Если по какой-либо причине совокупность значений пикселов в `inputByteArray` не формирует прямоугольник, метод прекращает обработку, когда обнаруживает это.

Не забывайте, что и для получения, и для установки данных пикселов массив байтов должен иметь значения пикселов альфа, красного, зеленого и синего цветов (ARGB) с глубиной цвета 32 бита.

В следующем примере методы `getPixels()` и `setPixels()` используются для копирования группы пикселов из одного объекта BitmapData в другой.

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.utils.ByteArray;
import flash.geom.Rectangle;

var bitmapDataObject1:BitmapData = new BitmapData(100, 100, false, 0x006666FF);
var bitmapDataObject2:BitmapData = new BitmapData(100, 100, false, 0x00FF0000);

var rect:Rectangle = new Rectangle(0, 0, 100, 100);
var bytes:ByteArray = bitmapDataObject1.getPixels(rect);

bytes.position = 0;
bitmapDataObject2.setPixels(rect, bytes);

var bitmapImage1:Bitmap = new Bitmap(bitmapDataObject1);
addChild(bitmapImage1);
var bitmapImage2:Bitmap = new Bitmap(bitmapDataObject2);
addChild(bitmapImage2);
bitmapImage2.x = 110;
```

## Обнаружение наложений на уровне пикселов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `BitmapData.hitTest()` обнаруживает наложения данных растровых изображений на другой объект или точку на уровне пикселов.

Метод `BitmapData.hitTest()` принимает пять параметров.

- `firstPoint` (Point): этот параметр обозначает положение пиксела в левом верхнем углу первого объекта BitmapData, на котором выполняется проверка наложения.
- `firstAlphaThreshold` (uint): этот параметр задает максимальное значение альфа-канала, которое соответствует непрозрачному состоянию в данной операции проверки наложения.
- `secondObject` (Object): этот параметр обозначает вторую область, участвующую в наложении. Объект `secondObject` может быть объектом Rectangle, Point, Bitmap или BitmapData. Этот объект представляет собой область, в которой выполняется обнаружение наложения.
- `secondBitmapDataPoint` (Point): этот необязательный параметр используется для определения положения пиксела во втором объекте BitmapData. Используйте этот параметр, только если значением `secondObject` является объект BitmapData. Значение по умолчанию `null`.
- `secondAlphaThreshold` (uint): этот необязательный параметр обозначает максимальное значение альфа-канала, которое соответствует непрозрачному состоянию второго объекта BitmapData. Значение по умолчанию 1. Используйте этот параметр, только когда значением `secondObject` является BitmapData и оба объекта BitmapData прозрачные.

При обнаружении наложения непрозрачных изображений помните, что ActionScript рассматривает изображение как полностью непрозрачный прямоугольник (или ограничивающий прямоугольник). В то же время при проверке наложения прозрачных изображений на уровне пикселов прозрачными должны быть оба изображения. Кроме того, ActionScript использует параметры порога альфа-канала для определения того, в какой точке пиксел становится из прозрачного непрозрачным.

В примере ниже показано, как создать три растровых изображения и проверить их на наложение пикселов с помощью двух точек наложения (одна возвращает значение false, другая — true).

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.geom.Point;

var bmd1:BitmapData = new BitmapData(100, 100, false, 0x000000FF);
var bmd2:BitmapData = new BitmapData(20, 20, false, 0x00FF3300);

var bm1:Bitmap = new Bitmap(bmd1);
this.addChild(bm1);

// Create a red square.
var redSquare1:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare1);
redSquare1.x = 0;

// Create a second red square.
var redSquare2:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare2);
redSquare2.x = 150;
redSquare2.y = 150;

// Define the point at the top-left corner of the bitmap.
var pt1:Point = new Point(0, 0);
// Define the point at the center of redSquare1.
var pt2:Point = new Point(20, 20);
// Define the point at the center of redSquare2.
var pt3:Point = new Point(160, 160);

trace(bmd1.hitTest(pt1, 0xFF, pt2)); // true
trace(bmd1.hitTest(pt1, 0xFF, pt3)); // false
```

# Копирование данных растровых изображений

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Скопировать данные из одного растрового изображения в другое можно с помощью нескольких методов: `clone()`, `copyPixels()`, `copyChannel()`, `draw()` и `drawWithQuality()` (метод `drawWithQuality` доступен в средах Flash Player 11.3 и AIR 3.3 и более поздних версий).

Как можно догадаться по его имени, метод `clone()` позволяет клонировать, или дублировать, данные растрового изображения одного объекта BitmapData в другой. При вызове этого метода он возвращает новый объект BitmapData, являющийся точной копией исходного экземпляра.

В примере ниже показано, как клонировать оранжевый квадрат (родитель) и разместить клон рядом с ним.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myParentSquareBitmap:BitmapData = new BitmapData(100, 100, false, 0x00ff3300);
var myClonedChild:BitmapData = myParentSquareBitmap.clone();

var myParentSquareContainer:Bitmap = new Bitmap(myParentSquareBitmap);
this.addChild(myParentSquareContainer);

var myClonedChildContainer:Bitmap = new Bitmap(myClonedChild);
this.addChild(myClonedChildContainer);
myClonedChildContainer.x = 110;
```

Метод `copyPixels()` предлагает простой и быстрый способ копирования пикселов одного объекта BitmapData в другой. Метод делает прямоугольный снимок (что определяется параметром `sourceRect`) исходного изображения и копирует его в другую прямоугольную область того же размера. Расположение вновь «вставленного» прямоугольника определяется параметром `destPoint`.

Метод `copyChannel()` дублирует значение канала (альфа, красный, зеленый или синий) исходного объекта BitmapData и копирует его в канал целевого объекта BitmapData. При вызове этого метода остальные каналы целевого объекта BitmapData никак не затрагиваются.

Методы `draw()` и `drawWithQuality()` отрисовывают, или визуализируют, графическое содержимое исходного спрайта, ролика или другого экранного объекта в новом растровом изображении. С помощью параметров `matrix`, `colorTransform`, `blendMode` и `clipRect` (для целевого объекта) можно указывать способ визуализации. Этот метод использует для генерации данных векторный визуализатор проигрывателя Flash Player и среды AIR.

При вызове метода `draw()` или `drawWithQuality()` исходный объект (спрайт, ролик или другой экранный объект) передается в качестве первого параметра, как показано ниже.

```
myBitmap.draw(movieClip);
```

Если какие-то свойства исходного объекта (цвет, матрица и т.п.) изменялись после того, как он был загружен, они не будут скопированы в новый объект. Если вам нужно скопировать изменения в новый объект, скопируйте значения свойства `transform` исходного объекта в свойство `transform` объекта Bitmap, который использует новый объект BitmapData.

# Сжатие растровых данных

**Flash Player 11.3 и более поздних версий, AIR 3.3 и более поздних версий**

Метод `flash.display.BitmapData.encode()` позволяет встроенными средствами сжимать растровые данные в один из следующих форматов сжатия изображений.

- **PNG** — используется сжатие в формате PNG, дополнительно доступно быстрое сжатие, при котором основное внимание уделяется скорости сжатия, а не размеру файлов. Чтобы использовать сжатие в формате PNG, передайте новый объект `flash.display.PNGEncoderOptions` в качестве второго параметра метода `BitmapData.encode()`.
- **JPEG** — используется сжатие в формате JPEG с дополнительным указанием качества изображения. Чтобы использовать сжатие в формате JPEG, передайте новый объект `flash.display.JPEGEncoderOptions` в качестве второго параметра метода `BitmapData.encode()`.

- **JPEGXR** — используется сжатие в формате JPEG с расширенным диапазоном (XR); при этом можно дополнительно указать канал цвета, параметры потерь и энтропии. Чтобы использовать сжатие в формате JPEGXR, передайте новый объект `flash.display.JPEGXREncoderOptions` в качестве второго параметра метода `BitmapData.encode()`.

Данную функцию можно использовать для обработки изображений в процессе отправки данных на сервер или загрузки с него.

В следующем примере фрагмент обеспечивает сжатие объекта BitmapData с помощью `JPEGEncoderOptions`:

```
// Compress a BitmapData object as a JPEG file.
var bitmapData:BitmapData = new BitmapData(640,480,false,0x00FF00);
var byteArray:ByteArray = new ByteArray();
bitmapData.encode(new Rectangle(0,0,640,480), new flash.display.JPEGEncoderOptions(),
byteArray);
```

# Создание текстур с помощью функций шумов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для изменения вида растрового изображения можно применить к нему шумовой эффект с помощью метода `noise()` или `perlinNoise()`. Эффект шума можно сравнить с помехами, которые появляются на экране ненастроенного телевизора.

Для применения эффекта шума к растровому изображению используйте метод `noise()`. Этот метод применяет случайное значение цвета к пикселам внутри заданной области изображения.

Метод принимает пять перечисленных ниже параметров.

- `randomSeed` (int): величина случайного начального числа, определяющая узор. Несмотря на свое название, если параметр передает одно и то же число, результат тоже получается один и тот же. Для получения случайного результата используйте метод `Math.random()`. Он передаст для этого параметра случайное число.
- `low` (uint): этот параметр задает минимальное значение, генерируемое для каждого пиксела (от 0 до 255). Значение по умолчанию равно 0. Установка меньшего значения приводит к более темному рисунку шума, а большего — к более светлому.
- `high` (uint): этот параметр задает максимальное значение, генерируемое для каждого пиксела (от 0 до 255). Значение по умолчанию равно 255. Установка меньшего значения приводит к более темному рисунку шума, а большего — к более светлому.
- `channelOptions` (uint): этот параметр указывает, к какому каналу изображения применять эффект шума. Это число может сочетать любые из четырех значений каналов ARGB. Значение по умолчанию равно 7.
- `grayScale` (Boolean): при значении `true` этот параметр применяет значение `randomSeed` к пикселам растрового изображения и убирает цвет из изображения. Этот параметр не влияет на альфа-канал. Значение по умолчанию равно `false`.

В примере ниже показано, как создавать растровое изображение и применить к нему синий шум.

```
package
{
    import flash.display.Sprite;
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.BitmapDataChannel;

    public class BitmapNoise1 extends Sprite
    {
        public function BitmapNoise1()
        {
            var myBitmap:BitmapData = new BitmapData(250, 250,false, 0xff000000);
            myBitmap.noise(500, 0, 255, BitmapDataChannel.BLUE,false);
            var image:Bitmap = new Bitmap(myBitmap);
            addChild(image);
        }
    }
}
```

Если вы хотите создать более естественную текстуру, используйте метод `perlinNoise()` . Метод `perlinNoise()` позволяет создавать реалистичные текстуры, идеально подходящие для имитации облаков, дыма, воды, огня и даже взрывов.

Так как метод `perlinNoise()` генерируется алгоритмом, ему требуется меньше памяти, чем текстурам на основе растровых изображений. Тем не менее, он может снижать быстродействие процессора и замедлять обработку содержимого, в результате чего экран будет перерисовываться медленнее, особенно на старых компьютерах. Это происходит из-за вычислений с плавающей точкой, которые нужны для обработки алгоритмов шума Перлина.

Этот метод принимает девять параметров (первые шесть обязательны).

- `baseX` (Number): определяет значение x (размер) создаваемого узора.
- `baseY` (Number): определяет значение y (размер) создаваемого узора.
- `numOctaves` (uint): количество октав или индивидуальных функций шума, которые необходимо объединить с целью создания шума. При большем количестве октав создаются более детализированные изображения, но и времени на обработку требуется больше.
- `randomSeed` (int): значение случайного начального числа действует так же, как и в случае функции `noise()`. Для получения случайного результата используйте метод `Math.random()`. Он передаст для этого параметра случайное число.
- `stitch` (Boolean): при значении `true` этот метод будет стараться сгладить края перехода изображения для создания ровных текстур, которые можно использовать для мозаичной заливки растровым изображением.
- `fractalNoise` (Boolean): этот параметр отвечает за края градиентов, генерируемых методом. При значении `true` он создает фрактальный шум, смягчающий границы области эффекта. При значении `false` он создает турбулентность. Изображение с турбулентностью имеет видимые прерывания градиента, благодаря чему оно больше подходит для более резких визуальных эффектов, например для создания языков пламени или морских волн.
- `channelOptions` (uint): параметр `channelOptions` действует так же, как и в случае метода `noise()`. Он указывает, к какому цветовому каналу (растрового изображения) применяется узор шума. Это число может сочетать любые из четырех значений каналов ARGB. Значение по умолчанию равно 7.

- `grayScale` (Boolean): параметр `grayScale` действует так же, как и в случае метода `noise()`. При значении `true` этот параметр применяет значение `randomSeed` к пикселам растрового изображения и убирает цвет из изображения. Значение по умолчанию равно `false`.
- `offsets` (Array): массив точек, соответствующих смещениям в направлениях x и y для каждой октавы. Изменяя значения смещения, можно плавно прокручивать слои изображения. Каждая точка в массиве смещения применяется к функции шума конкретной октавы. Значение по умолчанию — `null`.

В следующем примере создается объект BitmapData размером 150 x 150 пикселов, для создания эффекта зеленых и синих облаков на котором вызывается метод `perlinNoise()`.

```
package
{
    import flash.display.Sprite;
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.BitmapDataChannel;

    public class BitmapNoise2 extends Sprite
    {
        public function BitmapNoise2()
        {
            var myBitmapDataObject:BitmapData =
                new BitmapData(150, 150, false, 0x00FF0000);

            var seed:Number = Math.floor(Math.random() * 100);
            var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE
            myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false, null);

            var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
            addChild(myBitmap);
        }
    }
}
```

# Прокрутка изображений

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Представьте, что вы создали электронную карту города, и каждый раз, когда пользователь сдвигает карту, нужно обновлять картинку (даже если карта сдвигается всего на несколько пикселов).

Один из способов реализации этой функциональности — перерисовывать изображение каждый раз, когда карта сдвигается. Другой путь — создать единое крупное изображение и применить метод `scroll()`.

Метод `scroll()` копирует растровое изображение на экране и вставляет его в новый экран со смещением, определяемым параметрами (`x`, `y`). Если часть изображения лежит за границами области видимости, создается эффект смещения. В сочетании с функцией таймера (или события `enterFrame`) этот метод позволяет анимировать прокрутку изображения.

В примере ниже показано, как превратить изображение с шумом Перлина (из предыдущего примера) в более крупное, три четверти которого остаются вне области видимости. Затем применяются метод `scroll()` и прослушиватель событий `enterFrame`, который смещает изображение на один пиксел по диагонали вниз. Этот метод вызывается каждый раз при вводе кадра, поэтому области за пределами зоны видимости визуализируются в рабочей области по мере смещения вниз.

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(1000, 1000, false, 0x00FF0000);
var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE;
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false, null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
myBitmap.x = -750;
myBitmap.y = -750;
addChild(myBitmap);

addEventListener(Event.ENTER_FRAME, scrollBitmap);

function scrollBitmap(event:Event):void
{
    myBitmapDataObject.scroll(1, 1);
}
```

# Выполнение множественного отображения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

*Множественные отображения* (*mipmaps*) — это растровые изображения, сгруппированные и связанные с текстурой для повышения качества визуализации и быстродействия. Каждое растровое изображение карты MIP представляет собой версию основного растрового изображения, но со сниженным уровнем детализации.

Например, можно использовать карту MIP, содержащую основное изображение размером 64 × 64 пикселов при самом высоком качестве. Изображения карты MIP с более низким качеством будут иметь размер 32 × 32, 16 × 16, 8 × 8, 4 × 4, 2 × 2 и 1 × 1 пиксел.

*Потоковая передача текстуры* — это возможность загрузить, в первую очередь, растровое изображение с самым низким качеством, а затем постепенно выводить на экран изображения более высокого качества по мере их загрузки. Поскольку растровые изображения более низкого качества имеют небольшой размер, они загружаются быстрее, чем основное изображение. Поэтому пользователи могут просмотреть изображение в приложении до загрузки основной его версии с высоким качеством.

Проигрыватель Flash Player версии 9.115.0 и более поздних и среда AIR поддерживают технологию *множественного отображения* через создание оптимизированных версий каждого растрового изображения различных масштабов (начиная с 50 %).

Проигрыватель Flash Player 11.3 и среда AIR 3.3 поддерживают потоковую передачу текстур с помощью параметра `streamingLevels` методов `Context3D.createCubeTexture()` и `Context3D.createTexture()`.

Благодаря сжатию текстур можно хранить изображения текстур в сжатом формате непосредственно в графическом процессоре, тем самым экономя память графического процессора и пропускную способность памяти. Как правило, сжатие текстур выполняется автономно, и сжатые текстуры загружаются в графический процессор в сжатом формате. Однако Flash Player 11.4 и AIR 3.4 поддерживают сжатие текстур во время выполнения, что бывает полезно в некоторых случаях, например при рендеринге динамических текстур из векторного содержимого. Для использования сжатия текстур во время выполнения необходимо выполнить следующее.

- Создайте объект текстуры. Для этого вызовите метод `Context3D.createTexture()`, передав `flash.display3D.Context3DTextureFormat.COMPRESSED` или `flash.display3D.Context3DTextureFormat.COMPRESSED_ALPHA` в третьем параметре.
- С использованием экземпляра `flash.display3D.textures.Texture`, возвращенного `createTexture()`, вызовите `flash.display3D.textures.Texture.uploadFromBitmapData()` или `flash.display3D.textures.Texture.uploadFromByteArray()`. Эти методы загружают и сжимают текстуру за одно действие.

Множественные отображения создаются для следующих типов растровых изображений:

- изображение (файлы JPEG, GIF и PNG), отображаемое классом Loader ActionScript 3.0
- изображение, находящееся в библиотеке документа Flash Professional
- объект BitmapData
- изображение, отображаемое при помощи функции `loadMovie()` ActionScript 2.0

Множественное отображение нельзя применить к объектам, к которым уже применен фильтр, или к роликам, к которым применено растровое кэширование. Тем не менее, множественное отображение можно применять к измененным растровым изображениям внутри экранного объекта с фильтром, даже если растровое изображение лежит внутри маскируемой области.

Создание множественных отображений происходит автоматически, но вы можете следовать некоторым рекомендациям, чтобы гарантировать оптимизацию изображений.

- Для воспроизведения видео установите для свойства `smoothing` объекта Video значение `true` (см. класс Video).
- В случае растровых изображений необязательно задавать свойству `smoothing` значение `true`, но со сглаживанием улучшение качества более заметно.
- Используйте для двухмерных растровых изображений размеры, кратные 4 или 8 (например, размер 640 x 128, который можно уменьшить следующим образом: 320 x 64 > 160 x 32 > 80 x 16 > 40 x 8 > 20 x 4 > 10 x 2 > 5 x 1).

  Для трехмерных текстур используйте карты MIP, в которых каждое изображение имеет разрешение, кратное 2 (то есть 2^n). Например, основное изображение имеет разрешение 1024 x 1024 пикселов. В этом случае изображения более низкого качества в карте MIP должны иметь размер 512 x 512, 256 x 256, 128 x 128 и так далее до 1 x 1 пиксела, чтобы в итоге карта MIP содержала 11 изображений.

  Обратите внимание, что множественное отображение не создается для содержимого растрового изображения с избыточной шириной или высотой.

# Пример растрового изображения: анимация вращающейся луны

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В примере с анимацией вращающейся луны показаны техники работы с объектами Bitmap и данными растровых изображений (объектами BitmapData). В этом примере мы создадим анимированное изображение вращающейся сферической луны, взяв в качестве исходного изображения плоскую фотографию лунной поверхности. Техники, демонстрируемые в примере, перечислены ниже.

- Загрузка внешнего изображения и доступ к данным без сжатия.
- Создание анимации путем многократного копирования пикселов из разных частей исходного изображения.
- Создание растрового изображения путем установки значений пикселов

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложений анимированной вращающейся луны можно найти в папке Samples/SpinningMoon. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| SpinningMoon.mxml<br>или<br>SpinningMoon.fla | Основной файл приложения в Flex (MXML) или Flash (FLA). |
| com/example/programmingas3/moon/MoonSphere.as | Класс, отвечающий за загрузку, отображение и анимацию луны. |
| moonMap.png | Файл фотографии лунной поверхности, который загружается и используется для создания анимации вращающейся луны. |

## Загрузка внешнего изображения в растровом формате

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Первой ключевой задачей в данном примере является загрузка внешнего файла изображения с фотографией лунной поверхности. Операция загрузки обрабатывается двумя методами класса MoonSphere: конструктором `MoonSphere()`, который инициирует процесс загрузки, и методом `imageLoadComplete()`, который вызывается после полной загрузки внешнего изображения.

Загрузка внешнего изображения похожа на загрузку внешнего SWF-файла: и в том, и в другом случае для загрузки используется экземпляр класса flash.display.Loader. Реальный код метода `MoonSphere()`, который начинает загрузку, выглядит следующим образом:

```
var imageLoader:Loader = new Loader();
imageLoader.contentLoaderInfo.addEventListener(Event.COMPLETE, imageLoadComplete);
imageLoader.load(new URLRequest("moonMap.png"));
```

В первой строке объявляется экземпляр Loader с именем `imageLoader`. Третья строка фактически начинает процесс загрузки, вызывая метод `load()` объекта Loader, передающий экземпляр URLRequest с URL-адресом загружаемой картинки. Во второй строке задается прослушиватель событий, который будет вызван после полной загрузки изображения. Обратите внимание, что метод `addEventListener()` вызывается не в самом экземпляре Loader, а, наоборот, в свойстве `contentLoaderInfo` объекта Loader. Сам по себе экземпляр Loader

не отправляет события, связанные с загружаемым содержимым. Однако его свойство `contentLoaderInfo` содержит ссылку на объект LoaderInfo, связанный с загружаемым в объект Loader содержимым (в данном случае внешним изображением). Объект LoaderInfo располагает несколькими событиями, относящимися к процессу и завершению загрузки внешнего содержимого, включая событие `complete` (`Event.COMPLETE`), которое вызывает метод `imageLoadComplete()`, когда изображение полностью загрузится.

Несомненно, начало загрузки внешнего изображение является важной частью процесса, но не менее важно знать, что делать после окончания загрузки. Как видно из приведенного выше кода, после загрузки изображения вызывается функция `imageLoadComplete()`. Эта функция выполняет ряд операций над загруженными данными изображения. Они описаны далее в соответствующих разделах. Тем не менее, для использования данных ей необходимо иметь к ним доступ. Когда для загрузки внешнего изображения используется объект Loader, загружаемое изображение становится экземпляром Bitmap, присоединенным в качестве дочернего экранного объекта Loader. В этом случае метод прослушивателя событий будет подключаться к экземпляру Loader как часть объекта события, который передается методу в качестве параметра. Метод `imageLoadComplete()` начинается со следующих строк:

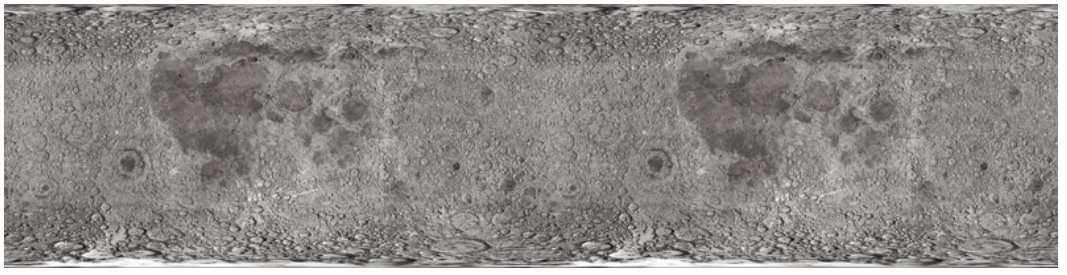
```
private function imageLoadComplete(event:Event):void
{
    textureMap = event.target.content.bitmapData;
    ...
}
```

Обратите внимание, что параметр объекта события называется `event` и является экземпляром класса Event. Каждый экземпляр класса Event имеет свойство `target`, относящееся к объекту, который вызывает событие (в данном случае это экземпляр LoaderInfo, в котором вызывается метод `addEventListener()`, как описано выше). Объект LoaderInfo, в свою очередь, имеет свойство `content`, которое (после окончания процесса загрузки) содержит экземпляр Bitmap с загруженным растровым изображением. Если вам нужно вывести изображение непосредственно на экран, можно присоединить этот экземпляр Bitmap (`event.target.content`) к контейнеру экранного объекта. (Также можно присоединить объект Loader к контейнеру экранного объекта.) Тем не менее, в данном примере загруженное содержимое используется в качестве данных изображения без сжатия, а не для вывода на экран. Следовательно, в первой строке метода `imageLoadComplete()` содержится свойство `bitmapData` загружаемого экземпляра Bitmap (`event.target.content.bitmapData`), сохраняемое в переменной экземпляра под названием `textureMap`, которая используется для создания анимации вращения луны. Описание см. далее.

## Создание анимации путем копирования пикселов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Суть анимации в воспроизведении движения или трансформации, чего можно добиться, изменяя изображение во времени. В данном примере наша цель — воспроизвести сферическую луну, которая вращается вокруг вертикальной оси. Тем не менее, в целях анимации в данном примере можно пожертвовать точностью сферического искажения. Рассмотрим изображение, которое загружается и используется в качестве источника данных изображения луны.



Как видите, это не одна и не несколько сфер, а лишь прямоугольная фотография лунной поверхности. Так как фотография сделана точно на экваторе луны, части изображения в верхней и нижней части растянуты и деформированы. Для устранения этого искажения и придания «сферичности» мы воспользуемся фильтром замещения текстуры, как будет показано позднее. Однако, поскольку исходная фотография имеет прямоугольную форму, для создания иллюзии вращения сферы нужно написать код, который будет сдвигать поверхность луны в горизонтальном направлении.

Обратите внимание, что изображение состоит из двух расположенных рядом копий фотографии лунной поверхности. Эта фотография является источником данных изображения, которые многократно копируются для создания иллюзии вращения. Расположение двух копий фотографии друг рядом с другом удобнее для создания эффекта непрерывного скольжения. Рассмотрим процесс анимации шаг за шагом.

В этом процессе участвуют два отдельных объекта ActionScript. Во-первых, у нас есть загруженное исходное изображение, представленное в коде экземпляром BitmapData с именем `textureMap`. Как описано выше, `textureMap` заполняется данными, как только загрузится внешнее изображение. Для этого используется следующий код:

```
textureMap = event.target.content.bitmapData;
```

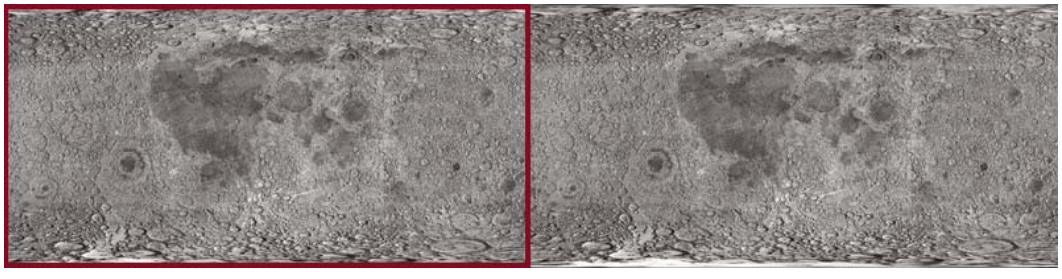
Содержимое `textureMap` представляет собой прямоугольное изображение луны. Кроме того, для создания анимации вращения в коде используется экземпляр Bitmap с именем `sphere`, который является экранным объектом, выводящим изображение луны на экран. Как и `textureMap`, объект `sphere` создается и заполняется исходными данными изображения в методе `imageLoadComplete()` с помощью следующего кода:

```
sphere = new Bitmap();
sphere.bitmapData = new BitmapData(textureMap.width / 2, textureMap.height);
sphere.bitmapData.copyPixels(textureMap,
                        new Rectangle(0, 0, sphere.width, sphere.height),
                        new Point(0, 0));
```

Как видно из кода, создается экземпляр `sphere`. Его свойство `bitmapData` (данные изображения без сжатия, отображаемые экземпляром `sphere`) создается с тем же значением высоты и вполовину меньшим значением ширины `textureMap`. Иными словами, содержимое `sphere` будет иметь тот же размер, что и фотография луны (так как изображение `textureMap` содержит две расположенные рядом фотографии луны). Затем свойство `bitmapData` заполняется данными изображения с помощью метода `copyPixels()`. Параметры метода `copyPixels()` могут указывать на несколько моментов.

- Первый параметр указывает, что данные изображения копируются из `textureMap`.
- Второй параметр (новый экземпляр Rectangle) указывает, из какой части `textureMap` будет сделан снимок. В данном случае снимком будет прямоугольник в левом верхнем углу `textureMap`, заданном первыми двумя параметрами `Rectangle(): 0, 0`, а ширина и высота снимка будут соответствовать свойствам `width` и `height` экземпляра `sphere`.
- Третий параметр (новый экземпляр Point со значениями x и y, равными `0`) определяет целевое расположение данных пиксела: в данном случае это верхний левый угол (0, 0) `sphere.bitmapData`.

Можно наглядно видеть, как код копирует пикселы из объекта `textureMap`, показанного на рисунке ниже, и вставляет их в `sphere`. Иными словами, содержимое BitmapData объекта `sphere` является частью объекта `textureMap`, которая выделена на рисунке.



Не забывайте, что это лишь начальное состояние объекта `sphere` — первое изображение, которое копируется в объект `sphere`.

Когда исходное изображение загружено, а объект `sphere` создан, остается последняя задача — создать анимацию. Она решается методом `imageLoadComplete()`. Анимацией управляет экземпляр Timer с именем `rotationTimer`, который создается и запускается с помощью следующего кода:

```
var rotationTimer:Timer = new Timer(15);
rotationTimer.addEventListener(TimerEvent.TIMER, rotateMoon);
rotationTimer.start();
```

Сначала создается экземпляр Timer с именем `rotationTimer`. Параметр, передаваемый конструктору `Timer()`, указывает, что экземпляр `rotationTimer` должен вызывать событие `timer` каждые 15 миллисекунд. Затем вызывается метод `addEventListener()` и указывает, что при выполнении события `timer` (`TimerEvent.TIMER`) вызывается метод `rotateMoon()`. Наконец, таймер запускается через вызов метода `start()`.

Из-за способа определения `rotationTimer` проигрыватель Flash Player вызывает метод `rotateMoon()` в классе MoonSphere примерно каждые 15 миллисекунд, и именно таким образом осуществляется анимация луны. Исходный код метода `rotateMoon()` показан ниже:

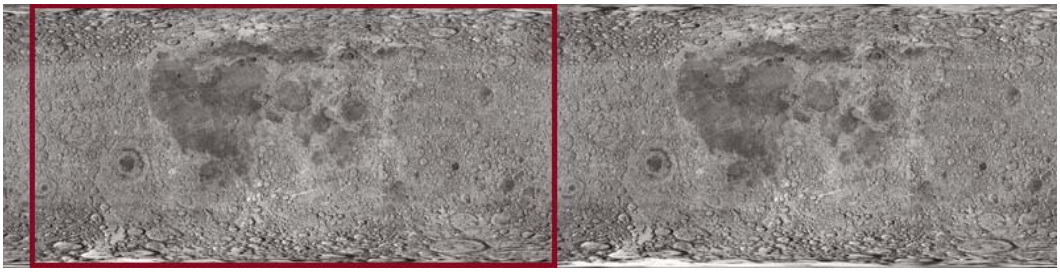
```
private function rotateMoon(event:TimerEvent):void
{
    sourceX += 1;
    if (sourceX > textureMap.width / 2)
    {
        sourceX = 0;
    }

    sphere.Data.copyPixels(textureMap,
                                    new Rectangle(sourceX, 0, sphere.width, sphere.height),
                                    new Point(0, 0));

    event.updateAfterEvent();
}
```

Код выполняет три задачи.

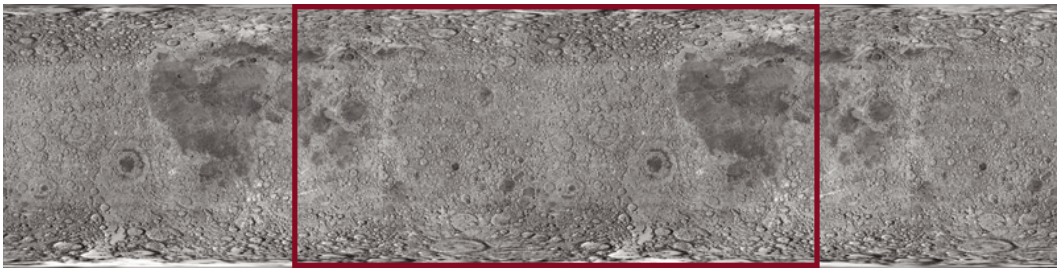
**1** Значение переменной `sourceX` (изначально равно 0) увеличивается на 1.

```
sourceX += 1;
```

Позже вы увидите, что `sourceX` используется для указания расположения объекта `textureMap`, из которого пикселы копируются в объект `sphere`, поэтому с помощью данного кода мы смещаем прямоугольник в объекте `textureMap` на один пиксел вправо. Вернемся к визуальному представлению. Через несколько циклов анимации исходный прямоугольник сместится вправо на несколько пикселов, как показано ниже.

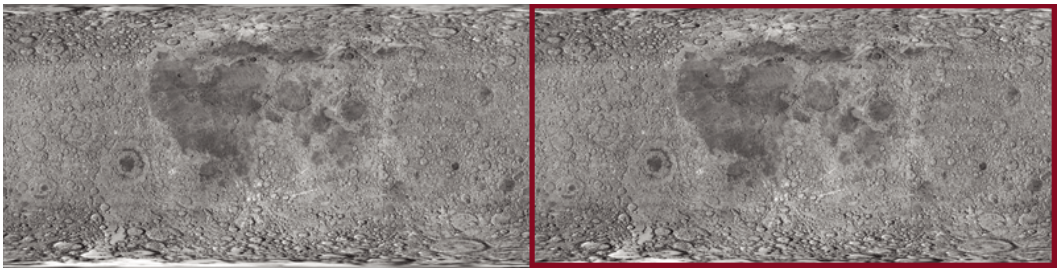


Еще через несколько циклов он сместится еще дальше.



Это постепенное, равномерное смещение области, из которой копируются пикселы — ключ к эффекту анимации. Медленно и равномерно смещая исходную область вправо, мы добиваемся эффекта вращения объекта `sphere`, который отображается на экране, влево. По этой причине требуется две копии исходного изображения поверхности луны (`textureMap`). Так как прямоугольник непрерывно движется вправо, большую часть времени оказывается выделенной область, расположенная не на одной копии фотографии, а захватывающая и ту, и другую копии.

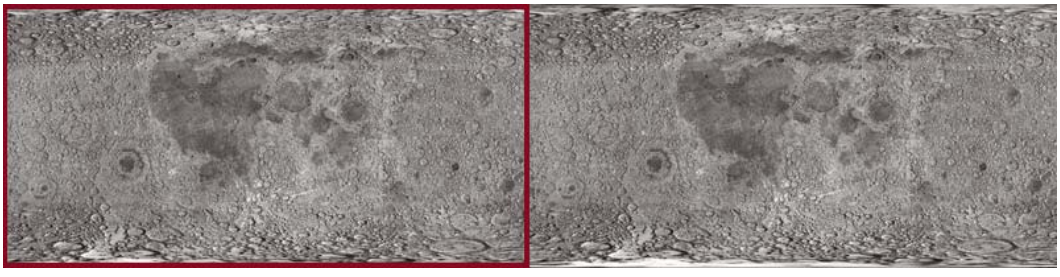
**2** С движением прямоугольника исходной области вправо связана одна сложность. Рано или поздно прямоугольник достигнет края изображения `textureMap`, и пикселы лунной поверхности, которые нужны для копирования в объект `sphere`, закончатся.



Эта проблема решается с помощью следующих строк кода:

```
if (sourceX >= textureMap.width / 2)
{
    sourceX = 0;
}
```

Код следит, когда `sourceX` (левый край прямоугольника) достигнет середины объекта `textureMap`. Когда это происходит, значение `sourceX` снова сбрасывается до 0, прямоугольник смещается к левому краю `textureMap`, и цикл начинается заново:



**3** После вычисления нужного значения `sourceX` остается последний этап — создание анимации для копирования новых пикселов исходного прямоугольника в объект `sphere`. Используемый для этого код похож на тот, с помощью которого заполняется объект `sphere` (приводился ранее). Единственное отличие — в данном случае при вызове конструктора `new Rectangle()` левый край прямоугольника помещается в точку `sourceX`:

```
sphere.bitmapData.copyPixels(textureMap,
                             new Rectangle(sourceX, 0, sphere.width, sphere.height),
                             new Point(0, 0));
```

Помните, что этот код вызывается многократно с интервалом в 15 миллисекунд. Так как исходная область прямоугольника постоянно смещается, а пикселы копируются в объект `sphere`, создается ощущение, что объект `sphere`, изображающий луну, постоянно движется. Другими словами, кажется, что луна непрерывно вращается.

## Создание сферической формы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Разумеется, луна имеет сферическую, а не прямоугольную форму. Следовательно, нужно взять анимированную прямоугольную фотографию лунной поверхности и превратить ее в шар. Для этого нужно выполнить два действия: скрыть все содержимое фотографии луны, лежащее за границами круга, с помощью маски и придать луне трехмерный вид с помощью фильтра замещения текстуры.

Во-первых, воспользуемся маской круглой формы для скрытия содержимого объекта MoonSphere, кроме сферы, созданной с помощью фильтра. Приведенный ниже код создает маску как экземпляр Shape и применяет ее к экземпляру MoonSphere.

```
moonMask = new Shape();
moonMask.graphics.beginFill(0);
moonMask.graphics.drawCircle(0, 0, radius);
this.addChild(moonMask);
this.mask = moonMask;
```

Обратите внимание, что из-за того, что MoonSphere является экранным объектом (основанным на классе Sprite), маску можно применить напрямую к экземпляру MoonSphere с помощью унаследованного свойства `mask`.

Для создания реалистичного эффекта вращающейся сферы недостаточно просто скрыть какие-то области фотографии под круглой маской. Из-за способа фотосъемки изображение непропорционально: верхняя и нижняя части искажены гораздо сильнее, чем экваториальная область. Чтобы придать луне объемный вид, понадобится фильтр замещения текстуры.

Фильтр замещения текстуры относится к тому типу фильтров, который используется для искажения изображения. В данном случае мы «исказим» фотографию луны, чтобы придать ей реалистичность. Для этого оставим середину как есть, а верхнюю и нижнюю часть сожмем по горизонтали. Так как фильтр применяется к квадратной области фотографии, сжатие верхней и нижней частей и сохранение середины в прежнем виде превратит квадрат в круг. Побочным эффектом анимации искаженного изображения является иллюзия, что пикселы в середине движутся быстрее, чем в верхней и нижней части. Это создает ощущение, что круг является объемным объектом (сферой).

Ниже показан код для создания фильтра замещения текстуры `displaceFilter`:

```
var displaceFilter:DisplacementMapFilter;
displaceFilter = new DisplacementMapFilter(fisheyeLens,
                                new Point(radius, 0),
                                BitmapDataChannel.RED,
                                BitmapDataChannel.GREEN,
                                radius, 0);
```

Первый параметр — `fisheyeLens` — известен как текстура изображения. В данном случае это объект BitmapData, создаваемый программными средствами. Создание этого изображения описано в разделе ««Создание растрового изображения путем установки значений пикселов» на странице 282». Остальные параметры указывают, в какой области фильтрованного изображения нужно применить фильтр, какие цветовые каналы использовать для управления эффектом смещения и насколько сильным будет это смещение. Созданный фильтр замещения текстуры применяется к объекту `sphere`, по-прежнему внутри метода `imageLoadComplete()`:

```
sphere.filters = [displaceFilter];
```

Итоговое изображение (с применением маски и фильтра замещения текстуры) показано ниже.

С каждым циклом анимации вращающейся луны содержимое сферы BitmapData заменяется новым снимком исходных данных изображения. Тем не менее, каждый раз применять фильтр заново необязательно, так как фильтр применяется к экземпляру Bitmap (экранному объекту), а не к данным растрового изображения (информации пикселов без сжатия). Помните, что экземпляр Bitmap — это не сами данные растрового изображения, а экранный объект, который выводит данные растрового изображения на экран. Можно провести аналогию: экземпляр Bitmap — это проектор, с помощью которого мы выводим слайды на экран, а объект BitmapData — это собственно слайд, который можно вывести на экран с помощью проектора. Фильтр можно применить напрямую к объекту BitmapData (пользуясь той же аналогией, подрисовать что-нибудь прямо на слайде, чтобы изменить картинку). Фильтр также можно применить к любому экранному объекту, включая экземпляр Bitmap — это можно сравнить с размещением фильтра перед линзой проектора для искажения выводимого на экран изображения (при этом исходный слайд не меняется). Доступ к данным без сжатия осуществляется через свойство bitmapData экземпляра Bitmap, поэтому фильтр можно применить и напрямую к несжатым данным растрового изображения. Тем не менее, лучше применять фильтр к экранному объекту Bitmap, а не к данным растрового изображения.

Подробную информацию об использовании фильтра замещения текстуры в языке ActionScript см. в разделе «Фильтрация экранных объектов» на странице 289.

## Создание растрового изображения путем установки значений пикселов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Важным свойством фильтра замещения текстуры является то, что он состоит из двух изображений. Одно из них — исходное — это изображение, которое изменяется в результате наложения фильтра. В данном примере исходным изображением является экземпляр Bitmap с именем `sphere`. Второе изображение, используемое фильтром — это изображение текстуры. Текстура не отображается на экране. Вместо этого цвет каждого из ее пикселов используется в качестве входа функции замещения, т.е. цвет пиксела с некоторыми координатами x, y в изображении текстуры определяет, какое смещение (физическое изменение положения) применяется к пикселу с такими же координатами x, y в исходном изображении.

Следовательно, чтобы использовать фильтр замещения текстуры для создания сферического эффекта, потребуется подходящее изображение текстуры: картинка с серым фоном и кругом, залитым одноцветным (красным) горизонтальным градиентом с переходом от темного к светлому, как показано ниже.

Так как в примере используется только одно изображение текстуры, оно создается только один раз — в методе `imageLoadComplete()` (иными словами, после окончания загрузки внешнего изображения). Изображение текстуры с именем `fisheyeLens` создается в результате вызова метода `createFisheyeMap()` класса MoonSphere:

```
var fisheyeLens:BitmapData = createFisheyeMap(radius);
```

Внутри метода `createFisheyeMap()` изображение текстуры фактически отрисовывается по пикселам с помощью метода `setPixel()` класса BitmapData. Ниже приведен полный код метода `createFisheyeMap()` и описание того, как он работает:

```
private function createFisheyeMap(radius:int):BitmapData
{
    var diameter:int = 2 * radius;

    var result:BitmapData = new BitmapData(diameter,
                                           diameter,
                                           false,
                                           0x808080);

    // Loop through the pixels in the image one by one
    for (var i:int = 0; i < diameter; i++)
    {
        for (var j:int = 0; j < diameter; j++)
        {
            // Calculate the x and y distances of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctX:Number = (i - radius) / radius;
            var pctY:Number = (j - radius) / radius;

            // Calculate the linear distance of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);

            // If the current pixel is inside the circle,
```

```
            // set its color.
            if (pctDistance < 1)
            {
                // Calculate the appropriate color depending on the
                // distance of this pixel from the center of the circle.
                var red:int;
                var green:int;
                var blue:int;
                var rgb:uint;
                red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
                green = 0;
                blue = 0;
                rgb = (red << 16 | green << 8 | blue);
                // Set the pixel to the calculated color.
                result.setPixel(i, j, rgb);
            }
        }
    }
    return result;
}
```

Во-первых, при вызове метода ему присваивается параметр `radius`, указывающий радиус создаваемого круга. Затем создается объект BitmapData, на котором будет рисоваться круг. Этот объект, названный `result`, в результате передается назад в качестве возвращаемого методом значения. Как показано в приведенном ниже фрагменте кода, экземпляр `result` создается с шириной и высотой, равными диаметру круга, является непрозрачным (значение третьего параметра — `false`) и имеет заливку цвета `0x808080` (серого):

```
var result:BitmapData = new BitmapData(diameter,
                                       diameter,
                                       false,
                                       0x808080);
```

Затем для повторного прохождения каждого пиксела используются два цикла. Внешний цикл проходит через каждый столбец изображения слева направо (переменной `i` обозначается горизонтальное положение пиксела, который в данный момент обрабатывается), а внутренний цикл проходит через каждый пиксел текущего столбца сверху вниз (переменной `j` обозначается вертикальное положение текущего пиксела). Ниже показан код для циклов (содержимое внутреннего цикла не показано):

```
for (var i:int = 0; i < diameter; i++)
{
    for (var j:int = 0; j < diameter; j++)
    {
        ...
    }
}
```

По мере прохождения циклов через пикселы вычисляется значение каждого пиксела (значение цвета этого пиксела в изображении текстуры). Этот процесс состоит из четырех шагов.

1. Код рассчитывает расстояние от текущего пиксела до центра круга по оси x (`i - radius`). Это значение нужно разделить на радиус, чтобы получить не абсолютное расстояние, а величину в процентах от радиуса (`(i - radius) / radius`). Это процентное значение сохраняется в переменной `pctX`, а аналогичное значение по оси y после вычисления сохраняется в переменной `pctY`, как показано в приведенном ниже коде.

   ```
   var pctX:Number = (i - radius) / radius;
   var pctY:Number = (j - radius) / radius;
   ```

2 С помощью стандартной тригонометрической формулы и теоремы Пифагора можно рассчитать линейное расстояние от центра круга до текущей точки, зная значения `pctX` и `pctY`. Это значение сохраняется в переменной `pctDistance`:

```
var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);
```

3 Затем код проверяет, не превышает ли относительное расстояние единицу (т. е. 100 % радиуса или, иными словами, не лежит ли пиксел за границами круга). Если пиксел оказывается внутри круга, ему присваивается вычисленное значение цвета (здесь это не показано, но описано в шаге 4). Если же он лежит вне круга, то с пикселом ничего не происходит, и его цвет остается серым:

```
if (pctDistance < 1)
{
    ...
}
```

4 Для пикселов, лежащих внутри круга, вычисляется значение цвета. Итоговый цвет — это один из оттенков красного, от черного (0 % красного) в левой части круга до ярко-красного (100 % красного) в правой. Цветовое значение вычисляется с использованием трех компонентов (красного, зеленого и синего):

```
red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
green = 0;
blue = 0;
```

Обратите внимание, что какое-либо значение есть только у красного компонента (переменная `red`). Зеленый и синий компоненты (переменные `green` и `blue`) показаны здесь для справки, но на самом деле их можно опустить. Так как целью этого метода является создание круга с красной градиентной заливкой, значения зеленого и синего компонентов не требуются.

Когда три отдельные величины цветов определены, они сводятся в единое целое значение цвета с помощью стандартного алгоритма битового сдвига:

```
rgb = (red << 16 | green << 8 | blue);
```

Наконец, когда значение цвета вычислено, оно назначается текущему пикселу с помощью метода `setPixel()` объекта BitmapData `result`:

```
result.setPixel(i, j, rgb);
```

# Асинхронное раскодирование растровых изображений

**Flash Player 11 и более поздних версий, Adobe AIR 2.6 и более поздних версий**

При работе с растровыми изображениями для повышения производительности приложения можно выполнять асинхронное раскодирование и загрузку растровых изображений. Во многих случаях асинхронное раскодирование растрового изображения может занимать столько же времени, как и синхронное. Однако раскодировка растрового изображения осуществляется в отдельном потоке, прежде чем связанный объект `Loader` отправляет событие `COMPLETE`. Таким образом, асинхронное раскодирование изображений большого размера можно выполнять после их загрузки.

Класс `ImageDecodingPolicy` в пакете `flash.system` позволяет указать схему загрузки растрового изображения. По умолчанию установлена синхронная схема загрузки.

| Политика раскодировки растровых изображений | Схема загрузки растровых изображений | Описание |
|---|---|---|
| `ImageDecodingPolicy.ON_DEMAND` | Синхронность | Загруженные изображения раскодируются при обращении к их данным.<br><br>Данную политику рекомендуется применять для изображений небольшого размера. Кроме того, политику можно использовать, когда приложению не требуется выполнять сложные эффекты и переходы. |
| `ImageDecodingPolicy.ON_LOAD` | Асинхронность | Загруженные изображения раскодируются при загрузке до отправки события `COMPLETE`.<br><br>Идеально подходит для изображений большого размера (больше 10 мегапикселов). При разработке приложений для мобильных устройств на базе AIR с переходами страниц использование данной политики загрузки позволит повысить производительность приложения. |

***Примечание.*** *Если загружается растровое изображение и выбрана политика `ON_LOAD`, изображение будет раскодировано асинхронно до отправки события `COMPLETE`.*

В примере кода ниже показано использование класса `ImageDecodingPolicy`.

```
var loaderContext:LoaderContext = new LoaderContext();
loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_LOAD
var loader:Loader = new Loader();
loader.load(new URLRequest("http://www.adobe.com/myimage.png"), loaderContext);
```

Схему раскодирования `ON_DEMAND` можно также использовать с методами `Loader.load()` и`Loader.loadBytes()`. Однако все остальные методы, получающие объект `LoaderContext` в качестве аргумента, будут игнорировать любые переданные значения `ImageDecodingPolicy` .

Следующий пример показывает отличие между схемами синхронного и асинхронного раскодирования растровых изображений.

```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.system.ImageDecodingPolicy;
    import flash.system.LoaderContext;

    public class AsyncTest extends Sprite
    {
        private var loaderContext:LoaderContext;
        private var loader:Loader;
        private var urlRequest:URLRequest;
        public function AsyncTest()
        {
            //Load the image synchronously
            loaderContext = new LoaderContext();
            //Default behavior.
            loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_DEMAND;
            loader = new Loader();
            loadImageSync();

            //Load the image asynchronously
            loaderContext = new LoaderContext();
            loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_LOAD;
            loader = new Loader();
            loadImageASync();
        }

        private function loadImageASync():void{
            trace("Loading image asynchronously...");
            urlRequest = new URLRequest("http://www.adobe.com/myimage.png");
            urlRequest.useCache = false;
            loader.load(urlRequest, loaderContext);
            loader.contentLoaderInfo.addEventListener
                (Event.COMPLETE, onAsyncLoadComplete);
```

```
        }

        private function onAsyncLoadComplete(event:Event):void{
            trace("Async. Image Load Complete");
        }

        private function loadImageSync():void{
            trace("Loading image synchronously...");
            urlRequest = new URLRequest("http://www.adobe.com/myimage.png");
            urlRequest.useCache = false;
            loader.load(urlRequest, loaderContext);
            loader.contentLoaderInfo.addEventListener
                (Event.COMPLETE, onSyncLoadComplete);
        }

        private function onSyncLoadComplete(event:Event):void{
            trace("Sync. Image Load Complete");
        }
    }
}
```

Эффект от использования различных политик декодирования продемонстрирован на странице Тибо Имбер (Thibaud Imbert). Асинхронное декодирование растровой графики в средах выполнения Adobe Flash

# Глава 14. Фильтрация экранных объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

С исторической точки зрения применение эффектов фильтра к растровым изображениям было прерогативой специализированного программного обеспечения для обработки изображений (такого как Adobe Photoshop® и Adobe Fireworks®). ActionScript 3.0 включает в себя пакет flash.filters, содержащий набор классов фильтров для создания эффектов на растровых изображениях. Эти эффекты позволяют разработчику программным образом применять фильтры к растровым изображениям и экранным объектам, тем самым во многих случаях добиваясь таких же эффектов, которые можно получить при помощи приложений для работы с графикой.

## Основы фильтрации экранных объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Одним из способов придания внешней изысканности приложению является использование в нем простых графических эффектов. Вы можете добавить тень позади фотографии, чтобы создать иллюзию трехмерности, или свечение вокруг кнопки, чтобы показать, что она активна. ActionScript 3.0 включает десять фильтров, которые можно применить к любому экранному объекту или экземпляру BitmapData. Диапазон встроенных фильтров простирается от базовых, таких как фильтры тени или свечения, до сложных, таких как фильтр схемы искривления и фильтр свертки.

***Примечание.*** *В дополнение к встроенным фильтрам вы можете программировать пользовательские фильтры и эффекты, используя технологию Pixel Bender. См. раздел «Работа с шейдерами Pixel Bender» на странице 324.*

**Важные понятия и термины**

Ниже приводится список важных терминов, встречающихся при создании фильтров.

**Скос** Кромка, создаваемая подсвечивающими пикселами с двух сторон и затемняющими пикселами с двух противоположных сторон. Этот эффект создает впечатление трехмерности границы. Обычно он используется для изображения отжатых или нажатых кнопок и подобного рода графики.

**Свертка** Искажение пикселов в изображении путем сочетания значения каждого пиксела со значениями некоторых или всех окружающих пикселов с применением разных соотношений.

**Смещение** Сдвиг или перемещение пикселов в изображении на новую позицию.

**Matrix** Сетка с числами, использованная для выполнения определенных математических расчетов путем применения чисел в сетке к различным значениям с последующим сочетанием результатов.

### Дополнительные разделы справки

пакет flash.filters

flash.display.DisplayObject.filters

flash.display.BitmapData.applyFilter()

# Создание и применение фильтров

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Фильтры позволяют применять ряд эффектов к растровым объектам и экранным объектам (от тени до скосов и размытия). Каждый фильтр определен как класс. Поэтому применение фильтров предполагает создание экземпляров объектов фильтров, что неотличимо от создания любого другого объекта. После создания экземпляра объекта фильтра его можно легко применить к экранному объекту, воспользовавшись свойством `filters` объекта. В случае объекта BitmapData это делается с помощью метода `applyFilter()`.

## Создание фильтра

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы создать объект фильтра, просто вызовите метод конструктора для выбранного класса фильтра. Например, чтобы создать объект DropShadowFilter, воспользуйтесь следующим кодом:

```
import flash.filters.DropShadowFilter;
var myFilter:DropShadowFilter = new DropShadowFilter();
```

Хотя здесь это не показано, конструктор `DropShadowFilter()` (как и конструкторы всех классов фильтра) принимает ряд необязательных параметров, которые можно использовать для настройки внешнего вида эффекта фильтра.

## Применение фильтра

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

После создания объекта фильтра его можно применить к экранному объекту или объекту BitmapData. Метод применения фильтра зависит от объекта, к которому он применяется.

**Применение фильтра к экранному объекту**

Когда эффекты фильтров применяются к экранному объекту, это происходит с помощью свойства `filters`. Свойство `filters` экранного объекта является экземпляром Array, элементы которого являются объектами фильтров, примененными к экранному объекту. Чтобы применить к экранному объекту один фильтр, создайте экземпляр фильтра, добавьте его к экземпляру Array и присвойте этот объект Array свойству `filters` экранного объекта:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.filters.DropShadowFilter;

// Create a bitmapData object and render it to screen
var myBitmapData:BitmapData = new BitmapData(100,100,false,0xFFFF3300);
var myDisplayObject:Bitmap = new Bitmap(myBitmapData);
addChild(myDisplayObject);

// Create a DropShadowFilter instance.
var dropShadow:DropShadowFilter = new DropShadowFilter();

// Create the filters array, adding the filter to the array by passing it as
// a parameter to the Array() constructor.
var filtersArray:Array = new Array(dropShadow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

Если нужно присвоить объекту несколько фильтров, просто добавьте все фильтры к экземпляру Array перед присвоением его свойству `filters`. К объекту Array можно добавить несколько объектов, передав их конструктору в качестве параметров. Например, этот код применяет фильтр фаски и фильтр свечения к ранее созданному экранному объекту:

```
import flash.filters.BevelFilter;
import flash.filters.GlowFilter;

// Create the filters and add them to an array.
var bevel:BevelFilter = new BevelFilter();
var glow:GlowFilter = new GlowFilter();
var filtersArray:Array = new Array(bevel, glow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

Для создания массива, содержащего фильтры, можно использовать конструктор `new Array()` (как показано в предыдущих примерах), либо воспользоваться синтаксисом литерала Array, поместив фильтры в квадратные скобки (`[]`). Например, следующая строка кода:

```
var filters:Array = new Array(dropShadow, blur);
```

выполняет то же действие, что и строка кода

```
var filters:Array = [dropShadow, blur];
```

Если к экранным объектам применяется несколько фильтров, они вступают в силу последовательно и имеют кумулятивный эффект. Например, если массив фильтров располагает двумя элементами (фильтром фаски, который добавляется первым, и фильтром тени, который добавляется вторым), фильтр тени применяется как к фильтру фаски, так и к экранному объекту. Это происходит из-за того, что фильтр тени находится в массиве фильтров на второй позиции. Если требуется применять фильтры без кумулятивного эффекта, необходимо применить каждый фильтр к новой копии экранного объекта.

Если один или несколько фильтров только присваиваются экранному объекту, можно создать экземпляр фильтра и присвоить его объекту в одной инструкции. Например, следующая строка кода применяет фильтр размытия к экранному объекту с именем `myDisplayObject`:

```
myDisplayObject.filters = [new BlurFilter()];
```

Предыдущий код создает экземпляр Array с помощью синтаксиса литерала Array (квадратные скобки), создает экземпляр BlurFilter в качестве элемента объекта Array и присваивает этот объект Array свойству `filters` экранного объекта с именем `myDisplayObject`.

**Удаление фильтров из экранного объекта**

Удаление всех фильтров из экранного объекта настолько же просто, как и присвоение нулевого значения свойству `filters`:

```
myDisplayObject.filters = null;
```

Если к объекту применены несколько фильтров и требуется удалить только один из фильтров, следует выполнить ряд действий, чтобы изменить массив свойства `filters`. Дополнительные сведения см. в разделе «Возможные проблемы при работе с фильтрами» на странице 292.

**Применение фильтра к объекту BitmapData**

Применение фильтра к объекту BitmapData требует использования метода `applyFilter()` для объекта BitmapData:

```
var rect:Rectangle = new Rectangle();
var origin:Point = new Point();
myBitmapData.applyFilter(sourceBitmapData, rect, origin, new BlurFilter());
```

Метод `applyFilter()` применяет фильтр к исходному объекту BitmapData, в результате чего появляется новое фильтрованное изображение. Этот метод не изменяет исходный оригинал; вместо этого результат применения фильтра к исходному изображению сохраняется в экземпляре BitmapData, по отношению к которому применяется метод `applyFilter()`.

## Как работают фильтры

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Фильтрация экранных объектов выполняется путем кэширования копии исходного объекта в качестве прозрачного растрового изображения.

После применения фильтра к экранному объекту среда выполнения кэширует объект в качестве растрового изображения до тех пор, пока объект имеет действительный список фильтров. Исходное растровое изображение затем используется в качестве исходного изображения для всех примененных впоследствии эффектов фильтров.

Каждый экранный объект, как правило, имеет два растровых изображения: одно с исходным нефильтрованным экранным объектом, а другое для конечного изображения, полученного после фильтрации. Конечное изображение используется при визуализации. Если экранный объект не изменяется, конечное изображение не обновляется.

## Возможные проблемы при работе с фильтрами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При работе с фильтрами необходимо учитывать ряд потенциальных источников путаницы или проблем.

### Кэширование фильтров и растровых изображений

Чтобы применить фильтр к экранному объекту, необходимо включить кэширование растровых изображений для этого объекта. В случае применения фильтра к экранному объекту, свойство `cacheAsBitmap` которого имеет значение `false`, свойству объекта `cacheAsBitmap` автоматически устанавливается значение `true`. Если впоследствии вы удалите все фильтры из экранного объекта, свойству `cacheAsBitmap` будет назначено последнее из присвоенных значений.

### Изменение фильтров во время выполнения

Если к экранному объекту уже применен один или несколько фильтров, набор фильтров невозможно изменить путем добавления дополнительных фильтров или удаления фильтров из массива свойства `filters`. Вместо этого, чтобы добавить или изменить применяемый набор фильтров, необходимо внести изменения в отдельный массив, а затем присвоить данный массив свойству filters экранного объекта для фильтров, применяемых к объекту. Это проще всего сделать путем считывания массива свойства `filters` в переменную Array и внесения модификаций в этот временный массив. После этого массив повторно назначается свойству `filters` экранного объекта. В более сложных случаях может быть необходимо сохранить отдельный главный массив фильтров. В него вносятся изменения, а затем главный массив повторно назначается свойству `filters` экранного объекта после каждого изменения.

### Добавление дополнительного фильтра

Процесс добавления дополнительного фильтра к экранному объекту, к которому уже применен один или несколько фильтров, показан в следующем коде. Сначала к экранному объекту с именем `myDisplayObject` применяется фильтр свечения; затем при нажатии на экранный объект вызывается функция `addFilters()`. В этой функции к `myDisplayObject` применяются два дополнительных фильтра:

```
import flash.events.MouseEvent;
import flash.filters.*;

myDisplayObject.filters = [new GlowFilter()];

function addFilters(event:MouseEvent):void
{
    // Make a copy of the filters array.
    var filtersCopy:Array = myDisplayObject.filters;

    // Make desired changes to the filters (in this case, adding filters).
    filtersCopy.push(new BlurFilter());
    filtersCopy.push(new DropShadowFilter());

    // Apply the changes by reassigning the array to the filters property.
    myDisplayObject.filters = filtersCopy;
}

myDisplayObject.addEventListener(MouseEvent.CLICK, addFilters);
```

### Удаление фильтра из набора фильтров

Если к экранному объекту применено несколько фильтров и требуется удалить один из этих фильтров, продолжая применять остальные фильтры к объекту, фильтры копируются во временный массив, ненужный фильтр удаляется из массива, а затем выполняется повторное назначение временного массива свойству `filters` экранного объекта. В разделе ««Получение значений и удаление элементов массива» на странице 33» указан ряд способов удаления одного или нескольких элементов из любого массива.

Наиболее прямой способ состоит в удалении самого верхнего фильтра объекта (фильтра, примененного к объекту последним). Метод `pop()` класса Array используется для удаления фильтра из массива:

```
// Example of removing the top-most filter from a display object
// named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the last element from the Array (the top-most filter).
tempFilters.pop();

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

Точно так же, чтобы удалить самый нижний фильтр (первый фильтр, примененный к объекту), используется тот же код, подставляющий метод `shift()` класса Array вместо метода `pop()`.

Чтобы удалить фильтр из середины массива фильтров (предполагая, что массив имеет более двух фильтров), необходимо воспользоваться методом `splice()`. Вы должны знать указатель (позицию в массиве) удаляемого фильтра. Например, следующий код удаляет второй фильтр (фильтр с указателем 1) из экранного объекта:

```
// Example of removing a filter from the middle of a stack of filters
// applied to a display object named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the second filter from the array. It's the item at index 1
// because Array indexes start from 0.
// The first "1" indicates the index of the filter to remove; the
// second "1" indicates how many elements to remove.
tempFilters.splice(1, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

**Определение указателя фильтра**

Необходимо знать, какой фильтр следует удалить из массива, чтобы узнать указатель фильтра. Необходимо либо узнать (изучив способ разработки приложения), либо вычислить указатель удаляемого фильтра.

Наиболее оптимальный подход состоит в разработке собственного приложения, чтобы удаляемый фильтр всегда находился на одной и той же позиции в наборе фильтров. Например, если у вас имеется единичный экранный объект, к которому применен фильтр свертки и фильтр тени (именно в таком порядке), и требуется удалить фильтр тени, сохранив фильтр свертки, этот фильтр находится в известной позиции (самый верхний фильтр). Благодаря этому вы всегда можете заранее узнать, какой метод Array использовать (в данном случае `Array.pop()` для удаления фильтра тени).

Если фильтр, который требуется удалить, всегда принадлежит определенному типу, но не обязательно занимает одну и ту же позицию в наборе фильтров, вы можете проверить тип данных для каждого фильтра в массиве, чтобы определить, какой из них следует удалить. Например, следующий код определяет, какой фильтр в наборе является фильтром свечения, и удаляет этот фильтр из набора.

```
// Example of removing a glow filter from a set of filters, where the
//filter you want to remove is the only GlowFilter instance applied
// to the filtered object.

var tempFilters:Array = filteredObject.filters;

// Loop through the filters to find the index of the GlowFilter instance.
var glowIndex:int;
var numFilters:int = tempFilters.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (tempFilters[i] is GlowFilter)
    {
        glowIndex = i;
        break;
    }
}

// Remove the glow filter from the array.
tempFilters.splice(glowIndex, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

В более сложном случае, например, тогда, когда удаляемый фильтр выбирается во время выполнения, наилучший подход состоит в сохранении отдельной постоянной копии массива фильтра, который служит главным списком фильтров. Каждый раз при внесении изменений в набор фильтров измените главный список, а затем используйте этот массив фильтра как свойство `filters` экранного объекта.

Например, в следующем примере кода к экранному объекту применяется ряд фильтров свертки для создания различных визуальных эффектов. Позднее в приложении один из этих фильтров удаляется, в то время как другие остаются. В данном случае код сохраняет главную копию массива фильтров, а также ссылку на подлежащий удалению фильтр. Поиск и удаление определенного фильтра похожи на предыдущий подход, за исключением того, что вместо создания временной копии массива фильтров обрабатывается главная копия, которая затем применяется к экранному объекту.

```
// Example of removing a filter from a set of
// filters, where there may be more than one
// of that type of filter applied to the filtered
// object, and you only want to remove one.

// A master list of filters is stored in a separate,
// persistent Array variable.
var masterFilterList:Array;

// At some point, you store a reference to the filter you
// want to remove.
var filterToRemove:ConvolutionFilter;

// ... assume the filters have been added to masterFilterList,
// which is then assigned as the filteredObject.filters:
filteredObject.filters = masterFilterList;

// ... later, when it's time to remove the filter, this code gets called:

// Loop through the filters to find the index of masterFilterList.
var removeIndex:int = -1;
var numFilters:int = masterFilterList.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (masterFilterList[i] == filterToRemove)
    {
        removeIndex = i;
        break;
    }
}

if (removeIndex >= 0)
{
    // Remove the filter from the array.
    masterFilterList.splice(removeIndex, 1);

    // Apply the new set of filters to the display object.
    filteredObject.filters = masterFilterList;
}
```

В данном подходе (когда вы сравниваете сохраненную ссылку на фильтр с элементами в массиве фильтров, чтобы определить, какой фильтр следует удалить) *необходимо* сохранить отдельную копию массива фильтров. Код не действует, если вы сравниваете сохраненную ссылку на фильтр с элементами во временном массиве, скопированными из свойства `filters` экранного объекта. Это происходит потому, что когда вы присваиваете массив свойству `filters`, среда выполнения создает копию каждого объекта фильтра в массиве внутренним образом. Эти копии (в отличие от исходных объектов) применяются к экранному объекту. И когда вы считываете свойство `filters` во временный массив, временный массив содержит ссылки на скопированные объекты фильтра, а не на ссылки на исходные объекты фильтра. Вследствие этого, если в предыдущем примере вы пытались определить указатель `filterToRemove`, сравнивая его с фильтрами во временном массиве фильтров, совпадения найдено не было.

**Фильтры и трансформации объекта**

Ни одна из фильтрованных областей (например, тень), находящихся за пределами ограничивающего прямоугольного окна экранного объекта, не рассматривается как часть поверхности в целях распознавания указываемой точки (определения наложения или пересечения экземпляра с другим экземпляром). Так как методы распознавания указываемой точки для класса DisplayObject основаны на векторах, распознавание невозможно выполнить по отношению к растровому результату. Например, если вы примените фильтр фаски к экземпляру кнопки, распознавание не будет доступно для скошенной части экземпляра.

Масштабирование, вращение и сдвиг не поддерживаются фильтрами; если сам фильтрованный объект масштабируется (если `scaleX` и `scaleY` не составляют 100 %), эффект фильтра не масштабируется вместе с экземпляром. Это означает, что исходная фигура экземпляра вращается, масштабируется или сдвигается, но фильтр не вращается, не масштабируется и не сдвигается вместе с экземпляром.

Экземпляр можно анимировать с помощью фильтра, чтобы создать реалистичные эффекты или вложенные экземпляры и воспользоваться классом BitmapData, чтобы анимировать фильтры для получения этого эффекта.

**Фильтры и объекты Bitmap**

Если к объекту BitmapData применяется любой фильтр, свойству `cacheAsBitmap` автоматически присваивается значение `true`. Таким образом, фильтр фактически применяется к копии объекта, а не к оригиналу.

Эта копия затем помещается на основной экран (над исходным объектом) как можно ближе к ближайшему пикселу. Если границы исходного растрового объекта изменяются, растровое изображение фильтрованной копии воссоздается на основе оригинала, а не растягивается и не искажается.

Если вы удалите все фильтры для экранного объекта, свойство `cacheAsBitmap` примет значение, которое было ему присвоено перед применением фильтра.

# Доступные фильтры показа

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

ActionScript 3.0 включает десять классов фильтров, которые можно применять к экранным объектам и объектам BitmapData:

- фильтр «Фаска» (класс BevelFilter);
- фильтр «Размытие» (класс BlurFilter);
- фильтр «Тень» (класс DropShadowFilter);
- фильтр «Свечение» (класс GlowFilter);
- фильтр «Градиентная фаска» (класс GradientBevelFilter);
- фильтр «Градиентное свечение» (класс GradientGlowFilter);
- фильтр «Матрица линейного преобразования» (класс ColorMatrixFilter);
- фильтр «Свертка» (класс ConvolutionFilter);
- фильтр «Карта смещения» (класс DisplacementMapFilter);
- фильтр «Шейдер» (класс ShaderFilter).

Первые шесть фильтров — это фильтры, которые можно использовать для создания одного специфического эффекта и определенной настройки доступного эффекта. Эти шесть фильтров можно применить с помощью ActionScript. Их также можно применить к объектам в Flash Professional с помощью панели «Фильтры». Следовательно, даже тогда, когда фильтры применяются с помощью ActionScript, если у вас есть Flash Professional, можно воспользоваться визуальным интерфейсом, чтобы быстро попробовать применить разные фильтры и настройки и выяснить, как создать необходимый эффект.

Последние четыре фильтра доступны только в ActionScript. Эти фильтры (фильтр матрицы линейного преобразования, фильтр свертки, фильтр карты смещения и фильтр шейдера) значительно повышают гибкость при выборе типов эффектов, для создания которых их можно использовать. Благодаря тому, что они не оптимизированы для одного эффекта, они обладают гибкостью и широкими возможностями. Например, путем выбора различных значений для своей матрицы фильтр свертки можно применять для создания таких эффектов, как размытие, выдавливание, увеличение резкости, поиск границ цветовых переходов, трансформация и т.д.

Каждый из этих фильтров (простой или сложный) можно настроить с помощью свойств. Обычно существует два варианта задания свойств фильтра. Все фильтры позволяют устанавливать свойства путем передачи значений параметров конструктору объекта фильтра. Кроме этого, независимо от того, задаются ли свойства фильтра путем передачи параметров, можно отрегулировать фильтры позднее, задав значения для свойств объекта фильтра. Большинство распечаток кода в примерах задают свойства напрямую, чтобы примеры было легче выполнять. Тем не менее, того же результата можно достичь с помощью меньшего числа строк, передав значения в качестве параметров в конструктор объекта фильтра. Дополнительные сведения о специфике каждого фильтра, его свойствах и параметрах его конструктора см. в описании пакета flash.filters в справочнике ActionScript® 3.0 для платформы ® Flash® Platform.

## Фильтр «Фаска»

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс BevelFilter позволяет добавлять к фильтруемому объекту трехмерную кромку с фаской. Этот фильтр заставляет твердые края объекта казаться вырубленными или скошенными.

Свойства класса BevelFilter позволяют настроить внешний вид фаски. Вы можете задать цвета подсветки и тени, размытие края фаски, углы фаски и их размещение. Можно даже добиться эффекта удаления фона.

В следующем примере загружается внешнее изображение и к нему применяется фильтр фаски.

```
import flash.display.*;
import flash.filters.BevelFilter;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.net.URLRequest;

// Load an image onto the Stage.
var imageLoader:Loader = new Loader();
var url:String = "http://www.helpexamples.com/flash/images/image3.jpg";
var urlReq:URLRequest = new URLRequest(url);
imageLoader.load(urlReq);
addChild(imageLoader);

// Create the bevel filter and set filter properties.
var bevel:BevelFilter = new BevelFilter();

bevel.distance = 5;
bevel.angle = 45;
bevel.highlightColor = 0xFFFF00;
bevel.highlightAlpha = 0.8;
bevel.shadowColor = 0x666666;
bevel.shadowAlpha = 0.8;
bevel.blurX = 5;
bevel.blurY = 5;
bevel.strength = 5;
bevel.quality = BitmapFilterQuality.HIGH;
bevel.type = BitmapFilterType.INNER;
bevel.knockout = false;

// Apply filter to the image.
imageLoader.filters = [bevel];
```

## Фильтр «Размытие»

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс BlurFilter размазывает, делает неясным экранный объект и его содержимое. Эффекты размытия нужны для того, чтобы создать впечатление, что объект находится не в фокусе, или симулировать быстрое движение, как в случае размытия при движении. Задав низкое значение для свойства `quality` фильтра размытия, можно симулировать эффект линзы, которая находится немного не в фокусе. Задав высокое значение для свойства `quality`, можно добиться мягкого эффекта размытия, который напоминает гауссово размытие.

В следующем примере с помощью метода `drawCircle()` создается круг, относящийся к классу Graphics, к которому применяется фильтр размытия:

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.BlurFilter;

// Draw a circle.
var redDotCutout:Sprite = new Sprite();
redDotCutout.graphics.lineStyle();
redDotCutout.graphics.beginFill(0xFF0000);
redDotCutout.graphics.drawCircle(145, 90, 25);
redDotCutout.graphics.endFill();

// Add the circle to the display list.
addChild(redDotCutout);

// Apply the blur filter to the rectangle.
var blur:BlurFilter = new BlurFilter();
blur.blurX = 10;
blur.blurY = 10;
blur.quality = BitmapFilterQuality.MEDIUM;
redDotCutout.filters = [blur];
```

## Фильтр «Тень»

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Тени создают впечатление, что над целевым объектом находится отдельный источник освещения. Позицию и интенсивность этого источника освещения можно изменять, чтобы добиваться ряда различных эффектов тени.

Класс DropShadowFilter использует алгоритм, аналогичный алгоритму фильтра размытия. Основное различие состоит в том, что фильтр тени имеет ряд дополнительных свойств, которые можно модифицировать в целях симуляции различных атрибутов источника света (альфа-канала, цвета, сдвига и яркости).

Фильтр тени позволяет также применять параметры заказной трансформации к стилю тени, включая внутреннюю или внешнюю тень и режим удаления фона (аппликации).

Следующий код создает спрайт квадратного окна и применяет к нему фильтр тени:

```
import flash.display.Sprite;
import flash.filters.DropShadowFilter;

// Draw a box.
var boxShadow:Sprite = new Sprite();
boxShadow.graphics.lineStyle(1);
boxShadow.graphics.beginFill(0xFF3300);
boxShadow.graphics.drawRect(0, 0, 100, 100);
boxShadow.graphics.endFill();
addChild(boxShadow);

// Apply the drop shadow filter to the box.
var shadow:DropShadowFilter = new DropShadowFilter();
shadow.distance = 10;
shadow.angle = 25;

// You can also set other properties, such as the shadow color,
// alpha, amount of blur, strength, quality, and options for
// inner shadows and knockout effects.

boxShadow.filters = [shadow];
```

## Фильтр «Свечение»

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс GlowFilter применяет к экранным объектам эффект освещения, благодаря которому кажется, что из-за объекта исходит свет, создавая мягкое сияние.

Подобно фильтру тени фильтр свечения включает свойства для изменения расстояния, угла и цвета источника освещения, приводящие к различным эффектам. GlowFilter располагает несколькими вариантами выбора стиля свечения, включая внутреннее и внешнее свечения и эффект удаления фона.

Следующий код создает крест с помощью класса Sprite и применяет к нему эффект свечения:

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.GlowFilter;

// Create a cross graphic.
var crossGraphic:Sprite = new Sprite();
crossGraphic.graphics.lineStyle();
crossGraphic.graphics.beginFill(0xCCCC00);
crossGraphic.graphics.drawRect(60, 90, 100, 20);
crossGraphic.graphics.drawRect(100, 50, 20, 100);
crossGraphic.graphics.endFill();
addChild(crossGraphic);

// Apply the glow filter to the cross shape.
var glow:GlowFilter = new GlowFilter();
glow.color = 0x009922;
glow.alpha = 1;
glow.blurX = 25;
glow.blurY = 25;
glow.quality = BitmapFilterQuality.MEDIUM;

crossGraphic.filters = [glow];
```

## Фильтр «Градиентная фаска»

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс GradientBevelFilter позволяет применить усиленный эффект фаски к экранным объектам или объектам BitmapData. Использование цвета градиента по отношению к фаске сильно повышает пространственную глубину фаски, придавая краям более реалистичный трехмерный вид.

Следующий код создает прямоугольный объект с помощью метода `drawRect()`, относящегося к классу Shape, и применяет к нему фильтр градиентной фаски.

```
import flash.display.Shape;
import flash.filters.BitmapFilterQuality;
import flash.filters.GradientBevelFilter;

// Draw a rectangle.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Apply a gradient bevel to the rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter();

gradientBevel.distance = 8;
gradientBevel.angle = 225; // opposite of 45 degrees
gradientBevel.colors = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
gradientBevel.alphas = [1, 0, 1];
gradientBevel.ratios = [0, 128, 255];
gradientBevel.blurX = 8;
gradientBevel.blurY = 8;
gradientBevel.quality = BitmapFilterQuality.HIGH;

// Other properties let you set the filter strength and set options
// for inner bevel and knockout effects.

box.filters = [gradientBevel];

// Add the graphic to the display list.
addChild(box);
```

## Фильтр «Градиентное свечение»

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс GradientGlowFilter позволяет применить усиленный эффект свечения к экранным объектам или объектам BitmapData. Этот эффект расширяет возможности управления свечением и в свою очередь образует более реалистичный эффект свечения. Кроме того, фильтр градиентного свечения позволяет применять градиентное свечение к внутреннему, внешнему или верхнему краю объекта.

В следующем примере в рабочей области отрисовывается круг, к которому применяется фильтр градиентного свечения. По мере перемещения мыши далее вправо и вниз доля свечения увеличивается в горизонтальном и вертикальном направлениях соответственно. Кроме того, при каждом щелчке по рабочей области сила размытия возрастает.

```
import flash.events.MouseEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.GradientGlowFilter;

// Create a new Shape instance.
var shape:Shape = new Shape();

// Draw the shape.
shape.graphics.beginFill(0xFF0000, 100);
shape.graphics.moveTo(0, 0);
shape.graphics.lineTo(100, 0);
shape.graphics.lineTo(100, 100);
shape.graphics.lineTo(0, 100);
shape.graphics.lineTo(0, 0);
shape.graphics.endFill();

// Position the shape on the Stage.
addChild(shape);
shape.x = 100;
shape.y = 100;

// Define a gradient glow.
var gradientGlow:GradientGlowFilter = new GradientGlowFilter();
gradientGlow.distance = 0;
gradientGlow.angle = 45;
gradientGlow.colors = [0x000000, 0xFF0000];
gradientGlow.alphas = [0, 1];
gradientGlow.ratios = [0, 255];
gradientGlow.blurX = 10;
gradientGlow.blurY = 10;
gradientGlow.strength = 2;
gradientGlow.quality = BitmapFilterQuality.HIGH;
gradientGlow.type = BitmapFilterType.OUTER;

// Define functions to listen for two events.
function onClick(event:MouseEvent):void
{
    gradientGlow.strength++;
    shape.filters = [gradientGlow];
}

function onMouseMove(event:MouseEvent):void
{
    gradientGlow.blurX = (stage.mouseX / stage.stageWidth) * 255;
    gradientGlow.blurY = (stage.mouseY / stage.stageHeight) * 255;
    shape.filters = [gradientGlow];
}
stage.addEventListener(MouseEvent.CLICK, onClick);
stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
```

## Пример: сочетание основных фильтров

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В следующем примере кода ряд базовых фильтров используется в сочетании с объектом Timer для создания повторяющихся действий, симулирующих анимацию светофора.

```
import flash.display.Shape;
import flash.events.TimerEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.DropShadowFilter;
import flash.filters.GlowFilter;
import flash.filters.GradientBevelFilter;
import flash.utils.Timer;

var count:Number = 1;
var distance:Number = 8;
var angleInDegrees:Number = 225; // opposite of 45 degrees
var colors:Array = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
var alphas:Array = [1, 0, 1];
var ratios:Array = [0, 128, 255];
var blurX:Number = 8;
var blurY:Number = 8;
var strength:Number = 1;
var quality:Number = BitmapFilterQuality.HIGH;
var type:String = BitmapFilterType.INNER;
var knockout:Boolean = false;

// Draw the rectangle background for the traffic light.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Draw the 3 circles for the three lights.
var stopLight:Shape = new Shape();
stopLight.graphics.lineStyle();
stopLight.graphics.beginFill(0xFF0000);
stopLight.graphics.drawCircle(145,90,25);
stopLight.graphics.endFill();

var cautionLight:Shape = new Shape();
cautionLight.graphics.lineStyle();
cautionLight.graphics.beginFill(0xFF9900);
cautionLight.graphics.drawCircle(145,150,25);
cautionLight.graphics.endFill();

var goLight:Shape = new Shape();
goLight.graphics.lineStyle();
goLight.graphics.beginFill(0x00CC00);
goLight.graphics.drawCircle(145,210,25);
goLight.graphics.endFill();

// Add the graphics to the display list.
addChild(box);
```

```
addChild(stopLight);
addChild(cautionLight);
addChild(goLight);

// Apply a gradient bevel to the traffic light rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter(distance, angleInDegrees,
colors, alphas, ratios, blurX, blurY, strength, quality, type, knockout);
box.filters = [gradientBevel];

// Create the inner shadow (for lights when off) and glow
// (for lights when on).
var innerShadow:DropShadowFilter = new DropShadowFilter(5, 45, 0, 0.5, 3, 3, 1, 1, true,
false);
var redGlow:GlowFilter = new GlowFilter(0xFF0000, 1, 30, 30, 1, 1, false, false);
var yellowGlow:GlowFilter = new GlowFilter(0xFF9900, 1, 30, 30, 1, 1, false, false);
var greenGlow:GlowFilter = new GlowFilter(0x00CC00, 1, 30, 30, 1, 1, false, false);

// Set the starting state of the lights (green on, red/yellow off).
stopLight.filters = [innerShadow];
cautionLight.filters = [innerShadow];
goLight.filters = [greenGlow];

// Swap the filters based on the count value.
function trafficControl(event:TimerEvent):void
{
    if (count == 4)
    {
        count = 1;
    }

    switch (count)
    {
        case 1:
            stopLight.filters = [innerShadow];
            cautionLight.filters = [yellowGlow];
            goLight.filters = [innerShadow];
            break;
        case 2:
            stopLight.filters = [redGlow];
            cautionLight.filters = [innerShadow];
            goLight.filters = [innerShadow];
            break;
        case 3:
            stopLight.filters = [innerShadow];
            cautionLight.filters = [innerShadow];
            goLight.filters = [greenGlow];
            break;
    }

    count++;
}

// Create a timer to swap the filters at a 3 second interval.
var timer:Timer = new Timer(3000, 9);
timer.addEventListener(TimerEvent.TIMER, trafficControl);
timer.start();
```

## Фильтр «Матрица линейного преобразования»

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс ColorMatrixFilter используется для манипуляций значениями цвета и альфа-канала фильтруемого объекта. Это позволяет создавать изменения насыщенности, вращение цветовых тонов (сдвиг палитры с одного диапазона цветов к другому), изменения яркости и альфа-канала и другие эффекты манипуляций с цветами с помощью значений из одного канала цвета, которые потенциально можно применить к другим каналам.

Фильтр обрабатывает пикселы в исходном изображении один за другим и разделяет каждый пиксел на красный, зеленый, синий и альфа-компоненты. Затем он умножает значения, представленные в матрице линейного преобразования по каждому из этих значений, и складывает результаты, чтобы определить итоговое значение цвета, которое будет показано на экране для этого пиксела. Свойство `matrix` фильтра — это массив из 20 чисел, используемых при расчете итогового цвета. Детали конкретного алгоритма, используемого для расчета значений цвета, см. в разделе описания свойства `matrix` класса ColorMatrixFilter в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

## Фильтр «Свертка»

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс ConvolutionFilter можно использовать для применения к объектам BitmapData и экранным объектам широкого спектра графических преобразований, таких как размытие, выделение краев, усиление резкости, тиснения и фаски.

Фильтр свертки обрабатывает пикселы исходного изображения один за другим и определяет финальный цвет пиксела с помощью значения пиксела и пикселов, которые его окружают. Матрица, заданная как массив числовых значений, указывает, до какой степени значение каждого из окружающих пикселов влияет на финальный результат.

Рассмотрим наиболее часто используемый тип матрицы 3 x 3. Эта матрица включает девять значений:

```
N   N   N
N   P   N
N   N   N
```

Когда фильтр свертки применяется к определенному пикселу, программа учитывает значение цвета самого пиксела (в данном примере это «P»), а также значения окружающих пикселов (в данном примере это «N»). Однако путем установки значений в матрице вы указываете приоритет, который имеют те или иные пикселы при воздействии на финальное изображение.

Например, следующая матрица, которая применена с использованием фильтра свертки, оставит изображение совершенно без изменений:

```
0   0   0
0   1   0
0   0   0
```

Причина неизменности изображения состоит в том, что значение исходного пиксела имеет относительную силу 1 при определении финального цвета пиксела, тогда как значения окружающих пикселов имеют относительную силу 0. Это означает, что их цвета не влияют на финальное изображение.

Точно так же эта матрица вызовет сдвиг пикселов изображения на один пиксел влево:

```
0   0   0
0   0   1
0   0   0
```

Учтите, что в данном случае сам пиксел не влияет на финальное значение пиксела, отображенного на этой позиции в финальном изображении. Для определения конечного результата пиксела применяется только значение пиксела справа.

В ActionScript матрица создается как сочетание экземпляра Array, содержащего значения, и двух свойств, указывающих количество строк и столбцов в матрице. В следующем примере загружается изображение, и по завершении загрузки к изображению применяется фильтр свертки с помощью матрицы в предыдущем примере:

```
// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

function applyFilter(event:MouseEvent):void
{
    // Create the convolution matrix.
    var matrix:Array = [0, 0, 0,
                                    0, 0, 1,
                                    0, 0, 0];

    var convolution:ConvolutionFilter = new ConvolutionFilter();
    convolution.matrixX = 3;
    convolution.matrixY = 3;
    convolution.matrix = matrix;
    convolution.divisor = 1;

    loader.filters = [convolution];
}

loader.addEventListener(MouseEvent.CLICK, applyFilter);
```

В этом коде есть нечто неочевидное, а именно применение значений, отличных от 1 или 0 в матрице. Например, одна и та же матрица с числом 8 вместо 1 на правой позиции выполняет то же самое действие (сдвигает пикселы влево). Кроме того, она оказывает влияние на цвета изображения, усиливая их яркость в 8 раз. Это происходит потому, что финальные значения цвета пикселов рассчитываются путем умножения матричных значений на значения исходного цвета пикселов, сложения этих значений и деления на значение свойства `divisor` фильтра. Учтите, что в коде примера свойству `divisor` присвоено значение 1. Общее правило состоит в том, что если вы хотите, чтобы яркость цветов оставалась примерно такой же, что и в исходном изображении, необходимо приравнять делитель к сумме значений матрицы. Поэтому если значения матрицы составляют 8, а делитель равен 1, финальное изображение будет примерно в 8 раз ярче исходного.

Хотя воздействие этой матрицы не очень заметно, для создания различных эффектов можно использовать другие значения матрицы. Ниже приводятся несколько стандартных наборов значений матрицы 3 x 3 для различных эффектов.

- Основное размытие (делитель 5):

```
0 1 0
1 1 1
0 1 0
```

- Усиление резкости (делитель 1):

```
 0, -1, 0
-1, 5, -1
 0, -1, 0
```

- Выделение краев (делитель 1):

```
 0, -1, 0
-1, 4, -1
 0, -1, 0
```

- Эффект выдавливания (делитель 1):

```
-2, -1, 0
-1, 1, 1
 0, 1, 2
```

Обратите внимание, что в большинстве этих эффектов делитель равен 1. Это происходит потому, что сложение отрицательных и положительных значений матрицы приводит к результату равному 1 (или 0 в случае выделения краев, но значение свойства `divisor` не может быть равно 0).

## Фильтр «Карта смещения»

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс DisplacementMapFilter использует пиксельные значения из объекта BitmapData (который называется изображением карты смещения) для применения эффекта смещения к новому объекту. Изображение карты смещения обычно отличается от фактического экранного объекта или экземпляра BitmapData, к которому применяется фильтр. Эффект смещения включает смещение пикселов в фильтрованном изображении. Иными словами, пикселы до некоторой степени сдвигаются с исходной позиции. Этот фильтр можно применять для создания сдвига, деформации или пятен.

Местоположение и величина смещения, примененного к данному пикселу, определяется значением цвета, относящимся к изображению карты смещения. При работе с фильтром помимо указания изображения карты необходимо указать следующие значения, чтобы управлять расчетом смещения из изображения карты:

- Точка на карте: местоположение фильтрованного изображения, к которому будет применяться левый верхний угол фильтра смещения. Это можно использовать только в том случае, если вы хотите применить фильтр к части изображения.
- Компонент X: какой канал цвета, относящийся к изображению карты, влияет на позицию пикселов по оси x.
- Компонент Y: какой канал цвета, относящийся к изображению карты, влияет на позицию пикселов по оси y.
- Масштаб X: значение множителя, которое указывает на величину смещения по оси x.
- Масштаб Y: значение множителя, которое указывает на величину смещения по оси y.
- Режим фильтра: определяет действие по отношению к пустому месту, которое образуется при сдвиге пикселов. Параметры, определенные как константы в классе DisplacementMapFilterMode, должны показывать исходные пикселы (режим фильтра `IGNORE`) для того, чтобы пикселы начали появляться с другой стороны изображения (режим фильтра `WRAP`, выбранный по умолчанию), использовать ближайший сдвинутый пиксел (режим фильтра `CLAMP`) либо заполнить пространство цветом (режим фильтра `COLOR`).

Рассмотрим простой пример, чтобы понять принцип действия фильтра «Карта смещения». В этом коде загружается изображение. По завершении загрузки оно размещается по центру рабочей области, и к нему применяется фильтр карты смещения, благодаря чему пикселы во всем изображении сдвигаются влево по горизонтали.

```
import flash.display.BitmapData;
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.geom.Point;
import flash.net.URLRequest;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image3.jpg");
loader.load(url);
this.addChild(loader);

var mapImage:BitmapData;
var displacementMap:DisplacementMapFilter;

// This function is called when the image finishes loading.
function setupStage(event:Event):void
{
    // Center the loaded image on the Stage.
    loader.x = (stage.stageWidth - loader.width) / 2;
    loader.y = (stage.stageHeight - loader.height) / 2;

    // Create the displacement map image.
    mapImage = new BitmapData(loader.width, loader.height, false, 0xFF0000);

    // Create the displacement filter.
    displacementMap = new DisplacementMapFilter();
    displacementMap.mapBitmap = mapImage;
    displacementMap.mapPoint = new Point(0, 0);
    displacementMap.componentX = BitmapDataChannel.RED;
    displacementMap.scaleX = 250;
    loader.filters = [displacementMap];
}

loader.contentLoaderInfo.addEventListener(Event.COMPLETE, setupStage);
```

Для определения смещения использованы следующие свойства.

- Растровое изображение карты: растровое изображение карты является новым экземпляром BitmapData, созданным этим кодом. Его размеры соответствуют размерам загруженного изображения (чтобы смещение применялось ко всему изображению в целом). В нем используется сплошная заливка красными пикселами.
- Точка на карте: это значение соответствует точке с координатами 0, 0 (при этом смещение применяется к изображению в целом).
- Компонент X: в качестве этого значения выбрана константа `BitmapDataChannel.RED`. Это означает, что значение красного цвета растрового изображения карты будет определять величину смещения пикселов (расстояние сдвига) вдоль оси x.
- Масштаб X: это значение равно 250. Суммарная величина смещения (если за основу изображения карты взято полностью красное изображение) приводит лишь к небольшому сдвигу изображения (примерно на полпиксела). Поэтому если это значение приравнено 1, изображение сдвинется только на 0,5 пикселов по горизонтали. Приравняв его 250, вы получите сдвиг изображения приблизительно на 125 пикселов.

Эти параметры приводят к тому, что пикселы фильтрованного изображения сдвинутся на 250 пикселов влево. Направление (налево или направо) и величина сдвига основаны на значении цвета пикселов в изображении карты. Пикселы фильтрованного изображения обрабатываются поочередно (по крайней мере, пикселы в области применения фильтра, что в данном случае означает все пикселы). По отношению к каждому пикселу выполняются следующие операции.

1. Программа находит соответствующий пиксел в изображении карты. Например, когда фильтр рассчитывает величину смещения для пиксела, расположенного в верхнем левом углу фильтрованного изображения, эта программа обращается к пикселу, находящемуся в верхнем левом углу изображения карты.
2. Программа определяет значение указанного канала цвета в пикселе карты. В данном случае каналом цвета компонента x является красный канал, поэтому фильтр ищет значение красного канала изображения карты для указанного пиксела. Так как изображение карты имеет сплошную красную заливку, красный канал пиксела имеет значение 0xFF или 255. Эта цифра применяется в качестве значения смещения.
3. Программа сравнивает значение смещения со «средним» значением (127, которое находится посередине между 0 и 255). Если значение смещения меньше среднего значения, пиксел сдвигается в направлении положительных чисел (вправо при смещении по оси x; вниз при смещении по оси y). С другой стороны, если значение смещения выше среднего значения (как в данном примере), пиксел смещается в направлении отрицательных чисел (влево при смещении по оси x; вверх при смещении по оси y). Говоря точнее, фильтр вычитает значение смещения из 127, и результат (положительный или отрицательный) является относительной величиной применяемого смещения.
4. И напоследок программа определяет фактическую величину смещения, выясняя, какой процент от полного смещения выражает значение относительного смещения. В данном случае полностью красный означает 100 % смещения. Затем это процентное соотношение умножается на значение масштаба x или y, чтобы получить количество пикселов применяемого смещения. В данном примере 100 %, умноженные на 250, определяют величину смещения (примерно 125 пикселов влево).

Поскольку компонент y и масштаб y не определены, используются значения по умолчанию (не вызывающие смещения). Поэтому изображение вообще не сдвигается по вертикали.

В этом примере использован параметр режима фильтра по умолчанию (`WRAP`). Поэтому по мере сдвига пикселов влево пустое пространство справа заполняется пикселами, сдвинутыми с левого края изображения. Вы можете поэкспериментировать с этим значением, чтобы увидеть различные эффекты его применения. Например, при добавлении следующей строки к той части кода, где задаются свойства смещения (перед строкой `loader.filters = [displacementMap]`), это приведет к тому, что изображение будет выглядеть смазанным в рабочей области:

```
displacementMap.mode = DisplacementMapFilterMode.CLAMP;
```

Рассмотрим более сложный пример. В следующем примере фильтр карты смещения применяется для создания эффекта увеличительного стекла, наведенного на изображение:

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.display.BitmapDataChannel;
import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Shape;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.filters.DisplacementMapFilterMode;
import flash.geom.Matrix;
import flash.geom.Point;
import flash.net.URLRequest;

// Create the gradient circles that will together form the
// displacement map image
var radius:uint = 50;

var type:String = GradientType.LINEAR;
var redColors:Array = [0xFF0000, 0x000000];
var blueColors:Array = [0x0000FF, 0x000000];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var xMatrix:Matrix = new Matrix();
xMatrix.createGradientBox(radius * 2, radius * 2);
var yMatrix:Matrix = new Matrix();
yMatrix.createGradientBox(radius * 2, radius * 2, Math.PI / 2);

var xCircle:Shape = new Shape();
xCircle.graphics.lineStyle(0, 0, 0);
xCircle.graphics.beginGradientFill(type, redColors, alphas, ratios, xMatrix);
xCircle.graphics.drawCircle(radius, radius, radius);

var yCircle:Shape = new Shape();
yCircle.graphics.lineStyle(0, 0, 0);
yCircle.graphics.beginGradientFill(type, blueColors, alphas, ratios, yMatrix);
yCircle.graphics.drawCircle(radius, radius, radius);

// Position the circles at the bottom of the screen, for reference.
this.addChild(xCircle);
xCircle.y = stage.stageHeight - xCircle.height;
this.addChild(yCircle);
yCircle.y = stage.stageHeight - yCircle.height;
yCircle.x = 200;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create the map image by combining the two gradient circles.
var map:BitmapData = new BitmapData(xCircle.width, xCircle.height, false, 0x7F7F7F);
map.draw(xCircle);
var yMap:BitmapData = new BitmapData(yCircle.width, yCircle.height, false, 0x7F7F7F);
yMap.draw(yCircle);
map.copyChannel(yMap, yMap.rect, new Point(0, 0), BitmapDataChannel.BLUE,
BitmapDataChannel.BLUE);
```

```
yMap.dispose();

// Display the map image on the Stage, for reference.
var mapBitmap:Bitmap = new Bitmap(map);
this.addChild(mapBitmap);
mapBitmap.x = 400;
mapBitmap.y = stage.stageHeight - mapBitmap.height;

// This function creates the displacement map filter at the mouse location.
function magnify():void
{
    // Position the filter.
    var filterX:Number = (loader.mouseX) - (map.width / 2);
    var filterY:Number = (loader.mouseY) - (map.height / 2);
    var pt:Point = new Point(filterX, filterY);
    var xyFilter:DisplacementMapFilter = new DisplacementMapFilter();
    xyFilter.mapBitmap = map;
    xyFilter.mapPoint = pt;
    // The red in the map image will control x displacement.
    xyFilter.componentX = BitmapDataChannel.RED;
    // The blue in the map image will control y displacement.
    xyFilter.componentY = BitmapDataChannel.BLUE;
    xyFilter.scaleX = 35;
    xyFilter.scaleY = 35;
    xyFilter.mode = DisplacementMapFilterMode.IGNORE;
    loader.filters = [xyFilter];
}

// This function is called when the mouse moves. If the mouse is
// over the loaded image, it applies the filter.
function moveMagnifier(event:MouseEvent):void
{
    if (loader.hitTestPoint(loader.mouseX, loader.mouseY))
    {
        magnify();
    }
}
loader.addEventListener(MouseEvent.MOUSE_MOVE, moveMagnifier);
```

Код сначала генерирует два круга с градиентами, которые сочетаются, формируя изображение карты смещения. Красный круг создает смещение по оси x (`xyFilter.componentX = BitmapDataChannel.RED`), а синий круг приводит к смещению по оси y (`xyFilter.componentY = BitmapDataChannel.BLUE`). Чтобы вам было легче понять, как будет выглядеть изображение карты смещения, код добавляет в нижней части экрана исходные круги помимо их сочетания, которое служит изображением карты.

Затем код загружает изображение и по мере перемещения мыши применяет фильтр смещения к той части изображения, на которую указывает курсор. Круги с градиентом, использованные в качестве изображения карты смещения, вызывают распространение смещенной области в сторону от курсора. Обратите внимание, что серые области на изображении карты смещения не вызывают никакого смещения. Серый цвет имеет значение `0x7F7F7F`. Синий и красный каналы этого оттенка серого в точности совпадают со средним оттенком этих цветовых каналов, и смещения в серой области этого изображения карты не происходит. Точно так же смещение не выполняется в центре круга. Хотя цвет центра круга не серый, синий и красный канал этого цвета идентичны соответствующим каналам серого цвета. Поскольку синий и красный цвета являются цветами, вызывающими смещение, то смещение здесь не выполняется.

## Фильтр «Шейдер»

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Класс ShaderFilter позволяет использовать эффект настраиваемого фильтра, определенного как шейдер Pixel Bender. Поскольку эффект фильтра записан как шейдер Pixel Bender, этот эффект является полностью настраиваемым. Отфильтрованное содержимое передается шейдеру в качестве входных данных изображения, а результат операции шейдера становится результатом фильтра.

***Примечание.*** *Фильтр Shader доступен для использования, начиная с Flash Player версии 10 и Adobe AIR версии 1.5.*

Чтобы применить фильтр шейдера к объекту, сначала необходимо создать экземпляр Shader, представляющий используемый шейдер Pixel Bender. Сведения о процедуре создания экземпляра Shader и о том, как указать значения вводимого изображения и параметров, см. в разделе «Работа с шейдерами Pixel Bender» на странице 324.

При использовании шейдера в качестве фильтра необходимо учитывать три важных момента.

- Определение шейдера должно предусматривать прием по крайней мере одного вводимого изображения.

- Фильтрованный объект (экранный объект или объект BitmapData, к которому применяется фильтр) передается шейдеру в качестве значения первого вводимого изображения. Из-за этого не нужно указывать значение первого вводимого изображения вручную.
- Если шейдер определяет более одного вводимого изображения, дополнительные вводимые данные необходимо указывать вручную (путем задания свойства `input` любого экземпляра ShaderInput, который принадлежит экземпляру Shader).

После получения объекта Shader для своего шейдера создается экземпляр ShaderFilter. Это конкретный объект фильтра, который используется так же, как и любой другой фильтр. Чтобы создать ShaderFilter, использующий объект Shader, вызовите конструктор `ShaderFilter()` и передайте объект Shader в качестве аргумента, как показано в данном примере:

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
```

Полный пример использования фильтра «Шейдер» см. в разделе «Использование шейдера в качестве графического фильтра» на странице 343.

# Пример фильтрации экранных объектов: Filter Workbench

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Filter Workbench имеет пользовательский интерфейс, позволяющий применять к изображениям и другому визуальному содержимому различные фильтры и просматривать полученный в итоге код, который можно использовать для получения того же эффекта в ActionScript. Помимо предоставления инструмента для экспериментов с фильтрами это приложение располагает следующими техниками.

- Создание экземпляров различных фильтров
- Применение нескольких фильтров к экранному объекту

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения Filter Workbench находятся в папке Samples/FilterWorkbench. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| com/example/programmingas3/filterWorkbench/FilterWorkbenchController.as | Класс, обеспечивающий основные функциональные возможности приложения, включая смену содержимого, к которому применяются фильтры, и их применение по отношению к содержимому. |
| com/example/programmingas3/filterWorkbench/IFilterFactory.as | Интерфейс, определяющий общие методы, которые реализуются каждым заводским классом фильтров. Этот интерфейс определяет распространенные функции, используемые классом FilterWorkbenchController при взаимодействии с отдельными заводскими классами фильтров. |
| в папке com/example/programmingas3/filterWorkbench/:<br>BevelFactory.as<br>BlurFactory.as<br>ColorMatrixFactory.as<br>ConvolutionFactory.as<br>DropShadowFactory.as<br>GlowFactory.as<br>GradientBevelFactory.as<br>GradientGlowFactory.as | Набор классов, в каждом из которых реализован интерфейс IFilterFactory. Каждый из этих классов предоставляет функциональные возможности для создания и задания значений, относящихся к одному типу фильтра. Панели свойств фильтра в приложении используют эти заводские классы для создания экземпляров своих конкретных фильтров, которые класс FilterWorkbenchController извлекает и применяет к содержимому изображения. |
| com/example/programmingas3/filterWorkbench/IFilterPanel.as | Интерфейс, определяющий распространенные методы, которые реализуются классами, определяющими панели пользовательского интерфейса, используемые для операций со значениями фильтра в приложении. |
| com/example/programmingas3/filterWorkbench/ColorStringFormatter.as | Служебный класс, включающий метод преобразования численного значения цвета в шестнадцатеричный строковый формат |
| com/example/programmingas3/filterWorkbench/GradientColor.as | Класс, который служит объектом значения, сочетая в одном объекте три значения (цвет, альфа и соотношение), которые ассоциируются с каждым из цветов в GradientBevelFilter и GradientGlowFilter |
| Пользовательский интерфейс (Flex) | |
| FilterWorkbench.mxml | Основной файл, определяющий пользовательский интерфейс приложения. |
| flexapp/FilterWorkbench.as | Класс, обеспечивающий функциональные возможности пользовательского интерфейса основного приложения. Этот класс применяется как класс, стоящий за кодом файла приложения MXML. |

| File | Описание |
|---|---|
| В папке flexapp/filterPanels:<br>BevelPanel.mxml<br>BlurPanel.mxml<br>ColorMatrixPanel.mxml<br>ConvolutionPanel.mxml<br>DropShadowPanel.mxml<br>GlowPanel.mxml<br>GradientBevelPanel.mxml<br>GradientGlowPanel.mxml | Набор компонентов MXML, обеспечивающих функциональные возможности для каждой панели, используемой при выборе параметров для единичного фильтра. |
| flexapp/ImageContainer.as | Экранный объект, выступающий в роли контейнера для загруженного изображения на экране |
| flexapp/controls/BGColorCellRenderer.as | Заказное средство визуализации ячеек, используемое для изменения цвета фона ячейки в компоненте DataGrid |
| flexapp/controls/QualityComboBox.as | Заказной элемент управления, определяющий комбинированное окно, которое можно использовать в качестве параметра качества в ряде панелей фильтров. |
| flexapp/controls/TypeComboBox.as | Заказной элемент управления, определяющий комбинированное окно, которое может применяться в качестве параметра типа в ряде панелей фильтров. |
| Пользовательский интерфейс (Flash) | |
| FilterWorkbench.fla | Основной файл, определяющий пользовательский интерфейс приложения. |
| flashapp/FilterWorkbench.as | Класс, обеспечивающий функциональные возможности пользовательского интерфейса основного приложения. Этот класс применяется как класс документа для FLA-файла приложения. |
| В папке flashapp/filterPanels:<br>BevelPanel.as<br>BlurPanel.as<br>ColorMatrixPanel.as<br>ConvolutionPanel.as<br>DropShadowPanel.as<br>GlowPanel.as<br>GradientBevelPanel.as<br>GradientGlowPanel.as | Набор классов, обеспечивающих функциональные возможности для каждой панели, используемой при выборе параметров для единичного фильтра.<br>Для каждого класса существует также связанный символ MovieClip в библиотеке FLA-файла основного приложения, чье имя совпадает с именем класса (например, символ «BlurPanel» связан с классом, определенном в файле BlurPanel.as). В этих символах содержатся данные об именах и расположении компонентов, входящих в состав пользовательского интерфейса. |
| flashapp/ImageContainer.as | Экранный объект, выступающий в роли контейнера для загруженного изображения на экране |

| File | Описание |
|---|---|
| flashapp/BGColorCellRenderer.as | Заказное средство визуализации ячеек, используемое для изменения цвета фона ячейки в компоненте DataGrid |
| flashapp/ButtonCellRenderer.as | Заказное средство визуализации ячеек, используемое для включения компонента кнопки в ячейку компонента DataGrid |
| Содержимое фильтрованного изображения | |
| com/example/programmingas3/filterWorkbench/ImageType.as | Этот класс служит объектом значения, содержащим тип и URL одного файла изображения, для которого приложение может загружать и применять фильтры. Этот класс включает также набор констант, представляющий фактически доступные файлы изображений. |
| images/sampleAnimation.swf,<br>images/sampleImage1.jpg,<br>images/sampleImage2.jpg | Изображения и другое визуальное содержимое, к которому в приложении применяются фильтры. |

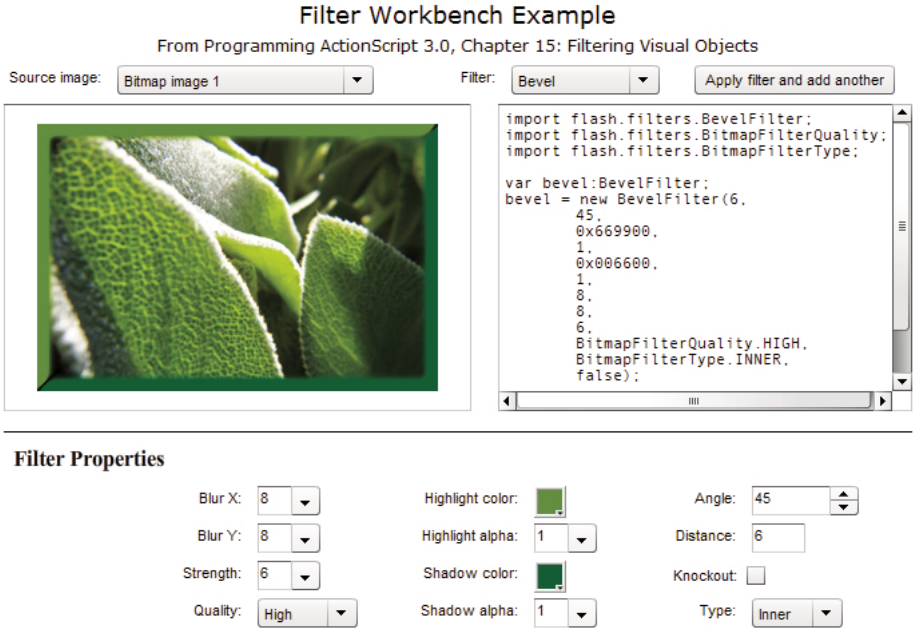
## Эксперименты с фильтрами ActionScript

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Приложение Filter Workbench разработано для того, чтобы помочь вам экспериментировать с различными эффектами фильтров и генерировать соответствующий код ActionScript для этого эффекта. Это приложение позволяет выбрать один из трех различных файлов, включающих визуальное содержимое (в том числе растровые изображения и анимацию, созданную в инструменте Flash) и применить восемь различных фильтров ActionScript к выбранному изображению. Фильтры могут применяться по отдельности либо в сочетании с другими фильтрами. Приложение включает следующие фильтры:

- фаска (flash.filters.BevelFilter),
- размытие (flash.filters.BlurFilter),
- матрица линейного преобразования (flash.filters.ColorMatrixFilter),
- свертка (flash.filters.ConvolutionFilter),
- тень (flash.filters.DropShadowFilter),
- свечение (flash.filters.GlowFilter),
- градиентная фаска (flash.filters.GradientBevelFilter),
- градиентное свечение (flash.filters.GradientGlowFilter).

После выбора пользователем изображения и фильтра, применяемого к изображению, приложение показывает панель с элементами управления для выбора конкретных свойств выделенного фильтра. Например, на следующем изображении показано приложение с выбранным фильтром «Фаска»:




Когда пользователь регулирует свойства фильтра, предварительный просмотр обновляется в реальном времени. Пользователь может также применить несколько фильтров. Для этого необходимо настроить один из них, нажать кнопку «Применить», настроить другой фильтр, нажать кнопку «Применить» и так далее.

Существует ряд функций и ограничений на панелях фильтров приложения.

- Фильтр матрицы линейного преобразования включает набор элементов управления для прямых манипуляций с распространенными свойствами изображения, включая яркость, контраст, насыщенность и цветовой тон. Кроме того, можно указать заказные значения матрицы линейного преобразования.
- Фильтр свертки, который доступен только с использованием ActionScript, включает набор часто применяемых значений матрицы свертки, который можно дополнить указанными заказными значениями. Однако если класс ConvolutionFilter принимает матрицу любого размера, приложение Filter Workbench использует фиксированную матрицу 3 x 3 (наиболее часто используемый размер фильтра).
- Фильтр карты смещения и фильтр шейдера, которые имеются только в ActionScript, недоступны в приложении Filter Workbench.

## Создание экземпляров фильтра

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Приложение Filter Workbench включает набор классов (по одному на каждый из доступных фильтров), которые применяются отдельными панелями для создания фильтров. Когда пользователь выбирает фильтр, код ActionScript, связанный с панелью фильтров, создает экземпляр соответствующего заводского класса фильтра. Эти классы называются *заводскими*, поскольку их цель состоит в создании экземпляров других объектов. Это во многом напоминает ситуацию на настоящем заводе, создающем отдельные продукты.

Каждый раз, когда пользователь изменяет значение свойства на панели, код панели вызывает соответствующий метод в заводском классе. Каждый заводской класс включает конкретные методы, которые применяются панелью для создания соответствующего экземпляра фильтра. Например, если пользователь выбирает фильтр «Размытие», приложение создает экземпляр BlurFactory. Класс BlurFactory включает метод `modifyFilter()`, принимающий три параметра: `blurX`, `blurY` и `quality`, которые совместно применяются для создания необходимого экземпляра BlurFilter:

```
private var _filter:BlurFilter;

public function modifyFilter(blurX:Number = 4, blurY:Number = 4, quality:int = 1):void
{
    _filter = new BlurFilter(blurX, blurY, quality);
    dispatchEvent(new Event(Event.CHANGE));
}
```

С другой стороны, если пользователь выбирает фильтр «Свертка», этот фильтр обладает гораздо большей гибкостью ) и, следовательно, может управлять большим набором свойств. В классе ConvolutionFactory при выборе пользователем другого значения на панели фильтров вызывается следующий код:

```
private var _filter:ConvolutionFilter;

public function modifyFilter(matrixX:Number = 0,
                                            matrixY:Number = 0,
                                            matrix:Array = null,
                                            divisor:Number = 1.0,
                                            bias:Number = 0.0,
                                            preserveAlpha:Boolean = true,
                                            clamp:Boolean = true,
                                            color:uint = 0,
                                            alpha:Number = 0.0):void
{
    _filter = new ConvolutionFilter(matrixX, matrixY, matrix, divisor, bias, preserveAlpha,
clamp, color, alpha);
    dispatchEvent(new Event(Event.CHANGE));
}
```

Учтите, что в каждой из этих ситуаций при изменении значений фильтра заводской объект отправляет событие `Event.CHANGE`, уведомляющее прослушиватели об изменении значений фильтра. Класс FilterWorkbenchController, который выполняет задачу фактического применения фильтров к фильтрованному содержимому, прослушивает это событие, чтобы определить момент, когда следует извлечь новую копию фильтра и повторно применить его к фильтрованному содержимому.

Классу FilterWorkbenchController не нужно располагать конкретными сведениями каждом заводском классе фильтров. Ему просто необходима информация о том, что фильтр изменился, и возможность доступа к копии данного фильтра. Для этого в приложении имеется интерфейс IFilterFactory, определяющий поведение, которое должен обеспечить заводской класс фильтра, чтобы эту работу смог проделать экземпляр FilterWorkbenchController приложения. IFilterFactory определяет метод `getFilter()`, используемый в классе FilterWorkbenchController:

```
function getFilter():BitmapFilter;
```

Учтите, что в определении метода интерфейса `getFilter()` указано, что он возвращает экземпляр BitmapFilter вместо конкретного типа фильтра. Класс BitmapFilter не определяет конкретный тип фильтра. Вместо этого BitmapFilter выступает в роли базового класса, на основе которого создаются все классы фильтров. Каждый заводской класс фильтра определяет конкретную реализацию метода `getFilter()`, в котором он возвращает ссылку на созданный им объект фильтра. Вот, например, укороченная версия исходного кода класса ConvolutionFactory:

```
public class ConvolutionFactory extends EventDispatcher implements IFilterFactory
{
    // ------- Private vars -------
    private var _filter:ConvolutionFilter;
    ...
    // ------- IFilterFactory implementation -------
    public function getFilter():BitmapFilter
    {
        return _filter;
    }
    ...
}
```

В реализации метода `getFilter()` для класса ConvolutionFactory он возвращает экземпляр ConvolutionFilter, хотя любой объект, вызывающий метод `getFilter()`, не обязательно располагает информацией об этом. Согласно определению метода `getFilter()`, которому следует ConvolutionFactory, он должен вернуть любой экземпляр BitmapFilter, который может относиться к любому из классов фильтра ActionScript.

## Применение фильтров к экранным объектам

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Как пояснялось ранее, приложение Filter Workbench использует экземпляр класса FilterWorkbenchController (далее он будет называться «экземпляром контроллера»), который выполняет задачу применения фильтров к выделенному визуальному объекту. Перед тем, как экземпляр контроллера получит возможность применения фильтра, ему необходимо определить, к какому изображению или визуальному содержимому его следует применить. Когда пользователь выбирает изображение, приложение вызывает метод `setFilterTarget()` в классе FilterWorkbenchController, передавая одну из констант, определенных в классе ImageType:

```
public function setFilterTarget(targetType:ImageType):void
{
    ...
    _loader = new Loader();
    ...
    _loader.contentLoaderInfo.addEventListener(Event.COMPLETE, targetLoadComplete);
    ...
}
```

С помощью этой информации экземпляр контроллера загружает намеченный файл, сохраняя его в переменной экземпляра с именем `_currentTarget` после завершения загрузки:

```
private var _currentTarget:DisplayObject;

private function targetLoadComplete(event:Event):void
{
    ...
    _currentTarget = _loader.content;
    ...
}
```

Когда пользователь выбирает фильтр, это приложение вызывает метод `setFilter()` экземпляра контроллера, передавая контроллеру ссылку на соответствующий заводской объект фильтра, который сохраняется в переменной экземпляра с именем `_filterFactory`.

```
private var _filterFactory:IFilterFactory;

public function setFilter(factory:IFilterFactory):void
{
    ...

    _filterFactory = factory;
    _filterFactory.addEventListener(Event.CHANGE, filterChange);
}
```

Учтите, что, как описывалось ранее, экземпляр контроллера не располагает сведениями о конкретном типе данных, относящемся к заводскому экземпляру фильтра. Он имеет только информацию о том, что в объекте реализован экземпляр IFilterFactory. Это означает, что у него есть метод `getFilter()` и он отправляет событие `change` (`Event.CHANGE`) при изменении фильтра.

Когда пользователь изменяет свойства фильтра на панели фильтров, экземпляр контроллера выясняет, что этот фильтр был изменен с помощью заводского события `change` фильтра, вызывающего метод `filterChange()` экземпляра контроллера. Этот метод в свою очередь вызывает метод `applyTemporaryFilter()`:

```
private function filterChange(event:Event):void
{
    applyTemporaryFilter();
}

private function applyTemporaryFilter():void
{
    var currentFilter:BitmapFilter = _filterFactory.getFilter();

    // Add the current filter to the set temporarily
    _currentFilters.push(currentFilter);

    // Refresh the filter set of the filter target
    _currentTarget.filters = _currentFilters;

    // Remove the current filter from the set
    // (This doesn't remove it from the filter target, since
    // the target uses a copy of the filters array internally.)
    _currentFilters.pop();
}
```

Задача применения фильтра к экранному объекту выполняется в методе `applyTemporaryFilter()`. Во-первых, контроллер извлекает ссылку на объект фильтра путем вызова заводского метода `getFilter()` фильтра.

```
var currentFilter:BitmapFilter = _filterFactory.getFilter();
```

Экземпляр контроллера имеет переменную экземпляра Array с именем `_currentFilters`, в которой он сохраняет все фильтры, примененные к экранному объекту. Следующее действие состоит в добавлении недавно обновленного фильтра к этому массиву:

```
_currentFilters.push(currentFilter);
```

Затем код присваивает массив фильтров свойству `filters` экранного объекта, которое выполняет фактическое применение фильтров к изображению:

```
_currentTarget.filters = _currentFilters;
```

Наконец, поскольку последний из добавленных фильтров является «рабочим», он не должен постоянно применяться к экранному объекту. Поэтому он удаляется из массива `_currentFilters`:

```
_currentFilters.pop();
```

Удаление этого фильтра из массива не влияет на фильтрованный экранный объект, поскольку экранный объект создает копию этого массива фильтров, когда он присваивается свойству `filters` и использует этот внутренний массив вместо оригинального. По этой причине любые изменения, внесенные в массив фильтров, не влияют на экранный объект до тех пор, пока массив не будет снова присвоен свойству `filters` экранного объекта.

# Глава 15. Работа с шейдерами Pixel Bender

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Набор инструментов Adobe Pixel Bender Toolkit позволяет разработчикам создавать шейдеры для применения графических эффектов и выполнения других видов обработки изображения и данных. Байт-код Pixel Bender может выполняться в ActionScript с целью применения эффекта к графическим данными или визуальному содержимому. Использование шейдеров Pixel Bender в ActionScript позволяет создавать собственные зрительные эффекты и выполнять обработку данных, используя не только встроенные возможности ActionScript.

***Примечание.*** *Начиная с Flash Player версии 10 и Adobe AIR версии 1.5 доступна поддержка шейдеров Pixel Bender. Наложения, фильтры и заливка Pixel Bender не поддерживаются при визуализации с использованием графического процессора.*

### Дополнительные разделы справки

Центр технологий Adobe Pixel Bender

Руководство разработчика по Pixel Bender (на английском языке)

Справочник по Pixel Bender

flash.display.Shader

flash.filters.ShaderFilter

Основы использования Pixel Bender для Flash (на английском языке)

Основы использования Pixel Bender для Flex (на английском языке)

## Основные сведения о шейдерах Pixel Bender

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Adobe Pixel Bender — это язык программирования, который служит для создания или обработки графического содержимого. С помощью Pixel Bender можно создать ядро, которое также называется шейдером. Шейдер определяет одну функцию, которая выполняется индивидуально для каждого пиксела изображения. В результате каждого вызова функции цвет пиксела в конкретном месте изображения меняется. Можно настроить операцию, задав вводимые изображения и значения параметров. В ходе одного выполнения шейдера ввод и значения параметров остаются неизменными. Изменяются только координаты пиксела, цвет которого является результатом вычисления функции.

Где это возможно, функция шейдера вызывается параллельно для нескольких пикселных координат вывода. Это ускоряет работу шейдера и может повысить быстродействие в целом.

В ActionScript с помощью шейдера можно без труда создать эффекты трех типов:

- заливка рисунка;

- режим наложения;
- фильтр.

Шейдер также может выполняться в автономном режиме. В автономном режиме результат выполнения шейдера получается напрямую, а не через предварительное определение предполагаемого использования. Результат можно получить в виде графических данных, либо в виде двоичных или числовых данных. Данные необязательно должны быть графическими. Таким образом можно ввести в шейдер набор данных. Шейдер обрабатывает данные, после чего можно просмотреть возвращенные им обработанные данные.

Поддержка Pixel Bender доступна начиная с Flash Player 10 и Adobe AIR 1.5. Наложения, фильтры и заливки Pixel Bender не поддерживаются при визуализации с использованием графического процессора. На мобильных устройствах шейдеры Pixel Bender выполняются при визуализации с использованием ЦП. Однако производительность существенно отличается от производительности на персональном компьютере. Многие программы шейдеров могут выполняться со скоростью несколько фреймов секунду.

### Важные понятия и термины

В следующем справочном списке содержатся важные термины, которые встречаются при создании и использовании шейдеров Pixel Bender.

**Ядро** Применительно к Pixel Bender, ядро — это то же самое, что шейдер. С помощью Pixel Bender код определяет ядро, которое, в свою очередь, определяет одну функцию, которая выполняется индивидуально для каждого пиксела изображения.

**Байт-код Pixel Bender** В процессе компиляции ядро Pixel Bender преобразуется в байт-код Pixel Bender. Обращение к байт-коду и его выполнение осуществляются во время выполнения.

**Язык Pixel Bender** Язык программирования, используемый для создания ядра Pixel Bender.

**Pixel Bender Toolkit** приложение, которое используется для создания файла байт-кода Pixel Bender из исходного кода Pixel Bender. Этот набор инструментов позволяет писать, тестировать и компилировать исходный код Pixel Bender.

**Шейдер** В данном документе шейдером называется набор функций, написанный на языке Pixel Bender. Код шейдера создает визуальный эффект или выполняет вычисление. И в том, и в другом случае шейдер возвращает набор данных (обычно это пикселы изображения). Шейдер выполняет одну и ту же операцию в каждой точке данных. При этом отличаются только координаты выводимого пиксела. При написании шейдера ActionScript не используется. Он пишется на языке Pixel Bender и компилируется в байт-код Pixel Bender. Шейдер можно встроить в SWF-файл во время компиляции или загрузить как внешний файл во время выполнения. В любом случае он вызывается кодом ActionScript путем создания объекта Shader, который затем связывается с байт-кодом шейдера.

**Ввод шейдера** Сложный ввод, как правило, графические данные растрового изображения, которые передаются шейдеру для выполнения вычислений. Для каждой переменной ввода, определенной в шейдере, используется одно значение (то есть одно изображение или набор двоичных данных) в процессе выполнения шейдера.

**Параметр шейдера** Одно значение (или небольшой набор значений) передаваемых шейдеру для выполнения вычислений. Каждое значение параметра определяется для одного выполнения шейдера, это значение не меняется на протяжении всей операции.

**Работа с примерами кода**

Может потребоваться тестирование приведенных примеров кода. Тестирование кода включает его выполнение и просмотр результатов в созданном SWF-файле. Во всех примерах создается содержимое с помощью API-интерфейса рисования, который использует эффект шейдера или изменяется под его действием.

Большая часть примеров состоит из двух частей. В первой части представлен исходный код Pixel Bender для шейдера, используемого в примере. Сначала с помощью Pixel Bender Toolkit необходимо скомпилировать исходный код в файл байт-кода Pixel Bender. Чтобы создать файл байт-кода Pixel Bender, выполните следующие действия.

1. Откройте программу Adobe Pixel Bender Toolkit. При необходимости в меню «Build» (Собрать) выберите «Turn on Flash Player warnings and errors» (Включить предупреждения и ошибки проигрывателя Flash Player).
2. Скопируйте код Pixel Bender и вставьте его в панель редактора кода Pixel Bender Toolkit.
3. В меню «File» (Файл) выберите «Export kernel filter for Flash Player» (Экспортировать ядро фильтра для проигрывателя Flash Player).
4. Сохраните файл байт-кода Pixel Bender в том же каталоге, что и документ Flash. Имя файла должно соответствовать имени, использованному в описании примера.

Вторая часть примера содержит код ActionScript, написанный в качестве файла класса. Чтобы проверить работу примера в Flash Professional, выполните следующие действия.

1. Создать пустой документ Flash и сохранить его на компьютере.
2. Создайте новый файл ActionScript и сохраните его в том же каталоге, что и документ Flash. Имя файла должно соответствовать имени класса в тестируемом коде. Например, если пример кода определяет класс с именем MyApplication, выберите имя MyApplication.as при сохранении файла ActionScript.
3. Скопируйте код в файл ActionScript и сохраните его.
4. В документе Flash щелкните пустую часть рабочей области или рабочего пространства, чтобы активировать инспектор свойств.
5. В поле «Класс документа» инспектора свойств введите имя класса ActionScript, скопированного из текста.
6. Запустите программу, выбрав «Управление» > «Тестировать ролик»

   Результаты выполнения примера появятся в окне предварительного просмотра.

Эти приемы проверки кодов из примеров подробно объясняются в главе ««Работа с примерами ActionScript» на странице 1185».

# Загрузка или встраивание шейдера

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Чтобы использовать шейдер Pixel Bender в ActionScript, сначала нужно получить к нему доступ через код ActionScript. Так как шейдер создается с помощью Adobe Pixel Bender Toolkit на языке Pixel Bender, код ActionScript не может обратиться к нему напрямую. Сначала нужно создать экземпляр класса Shader, который представляет шейдер Pixel Bender для ActionScript. Объект Shader позволяет получать информацию о шейдере, например, требует он параметров или значений ввода изображения. Объект Shader передается другим объектам, к которым требуется применить шейдер. Например, чтобы использовать шейдер в качестве фильтра, объект Shader нужно назначить свойству `shader` объекта ShaderFilter. А чтобы использовать шейдер в качестве заливки, объект Shader нужно передать в качестве аргумента методу `Graphics.beginShaderFill()`.

Код ActionScript может получить шейдер, созданное в Adobe Pixel Bender Toolkit (файл .pbj) двумя способами.

- Загрузка во время выполнения: файл шейдера: можно загрузить как внешний ресурс с помощью объекта URLLoader. Этот прием напоминает загрузку внешнего ресурса, например текстового файла. В следующем примере демонстрируется загрузка файла байт-кода шейдера в среду выполнения и его привязка к экземпляру Shader.

```
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.BINARY;
loader.addEventListener(Event.COMPLETE, onLoadComplete);
loader.load(new URLRequest("myShader.pbj"));

var shader:Shader;

function onLoadComplete(event:Event):void {
    // Create a new shader and set the loaded data as its bytecode
    shader = new Shader();
    shader.byteCode = loader.data;

    // You can also pass the bytecode to the Shader() constructor like this:
    // shader = new Shader(loader.data);

     // do something with the shader
}
```

- Встраивание в SWF-файл: шейдер можно встроить в SWF-файл во время компиляции с помощью тега метаданных `[Embed]`. Тег метаданных `[Embed]` доступен только в том случае, если при компиляции SWF-файла используется Flex SDK. В теге `[Embed]` параметр `source` указывает на файл шейдера, а параметр `mimeType` является `"application/octet-stream"`, как в следующем примере.

```
[Embed(source="myShader.pbj", mimeType="application/octet-stream")]
var MyShaderClass:Class;

// ...

// create a shader and set the embedded shader as its bytecode
var shader:Shader = new Shader();
shader.byteCode = new MyShaderClass();

// You can also pass the bytecode to the Shader() constructor like this:
// var shader:Shader = new Shader(new MyShaderClass());

// do something with the shader
```

В любом из двух случаев происходит привязка байт-кода необработанного шейдера (свойства `URLLoader.data` или экземпляра класса данных `[Embed]`) к экземпляру Shader. Как показано в предыдущих примерах, байт-код можно присвоить экземпляру Shader двумя способами. Можно передать байт-код шейдера в качестве аргумента конструктору `Shader()`. Либо можно задать его в качестве свойства `byteCode` экземпляра Shader.

После того как шейдер Pixel Bender создано и связано с объектом Shader, его можно использовать для создания разных эффектов. Затенение можно использовать в качестве фильтра, режима наложения, растровой заливки, а также для автономной обработки растрового изображения и других данных. Также можно использовать свойство `data` объекта Shader, чтобы получить метаданные шейдера, задать вводимые изображения и значения параметров.

# Получение метаданных шейдера

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Создавая ядро шейдера Pixel Bender, автор может указать метаданные о шейдере в исходном коде Pixel Bender. Используя шейдер в ActionScript, шейдер можно проанализировать и извлечь метаданные.

После того как создается экземпляр Shader и связывается с шейдером Pixel Bender, создается объект ShaderData, содержащий данные о шейдере, и сохраняется в свойстве `data` этого объекта Shader. Класс ShaderData не определяет собственных свойств. Однако во время выполнения в объект ShaderData динамически добавляется свойство для каждого значения метаданных, определенного в исходном коде шейдера. Каждому свойству присваивается то же имя, которое указано в метаданных. К примеру, предположим, что исходный код шейдера Pixel Bender включает следующее определение метаданных:

```
namespace : "Adobe::Example";
vendor : "Bob Jones";
version : 1;
description : "Creates a version of the specified image with the specified brightness.";
```

Объект ShaderData, созданный для этого шейдера, создается со следующими свойствами и значениями:

- `namespace` (String): `"Adobe::Example"`
- `vendor` (String): `"Bob Jones"` (Боб Джоунз)
- `version` (String): `"1"`
- `description` (String): `"Creates a version of the specified image with the specified brightness"` (Создает версию заданного изображения с требуемой яркостью)

Так как свойства метаданных добавляются в объект ShaderData динамически, можно использовать цикл `for..in` для анализа объекта ShaderData. С помощью этого приема можно выяснить, имеет ли шейдер какие-либо метаданные и каковы их значения. Помимо свойств метаданных, объект ShaderData может иметь свойства, представляющие ввод и параметры, определенные в шейдере. Анализируя объект ShaderData с помощью цикла `for..in`, проверьте тип данных для каждого свойства, чтобы определить, является ли оно вводом (экземпляр ShaderInput), параметром (экземпляр ShaderParameter) или метаданными (экземпляр String). В следующем примере демонстрируется использование цикла `for..in` для анализа динамических свойств, содержащихся в свойстве `data` шейдера. Каждое значение метаданных добавляется в экземпляр Vector с именем `metadata`. Обратите внимание, что для этого примера необходимо предварительно создать экземпляр Shader с именем `myShader`.

```
var shaderData:ShaderData = myShader.data;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (!(shaderData[prop] is ShaderInput) && !(shaderData[prop] is ShaderParameter))
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the metadata
```

Версию этого примера с извлечением вводов и параметров шейдера см. в разделе «Определение вводов и параметров шейдера» на странице 329. Дополнительные сведения о свойствах ввода и параметров см. в разделе «Указание значений вводов и параметров шейдера» на странице 329.

# Указание значений вводов и параметров шейдера

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Шейдер Pixel Bender во многих случаях по определению использует одно или несколько изображений, которые применяются при обработке шейдера. Например, шейдер, как правило, принимает исходное изображение и выводит его, применив тот или иной эффект. В зависимости от способа использования шейдера, вводимое значение может задаваться автоматически или передаваться явно. Подобным образом шейдер часто задает параметры, которые используются для настройки его вывода. Перед использованием шейдера необходимо также явно задать значения для всех параметров.

Чтобы задать вводы и параметры шейдера и определить, ожидает он вводов или параметров, нужно использовать свойство `data` объекта Shader. Свойство `data` является экземпляром класса ShaderData.

## Определение вводов и параметров шейдера

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Прежде чем задавать значения вводов и параметров шейдера, необходимо выяснить, что ожидает используемый шейдер: вводимое изображение или параметры. Каждый экземпляр Shader имеет свойство `data`, содержащее объект ShaderData. Если шейдер определяет вводы или параметры, их можно получить в качестве свойств этого объекта ShaderData. Имена свойств совпадают с именами вводов и параметров в исходном коде шейдера. Например, если шейдер определяет ввод с именем `src`, объект ShaderData имеет свойство с именем `src`, представляющее ввод. Каждое свойство, представляющее ввод, является экземпляром ShaderInput, а каждое свойство, представляющее параметр, — экземпляром ShaderParameter.

В идеале автор шейдера предоставляет его документацию, в которой указано, какие значения вводимых изображений и параметры ожидает шейдер, что они собой представляют, их допустимые значения и т. д.

Однако, если такой документации нет (как и исходного кода), данные шейдера можно проанализировать, чтобы определить его вводы и параметры. Свойства, представляющие вводы и параметры, добавляются в объект ShaderData динамически. Следовательно, объект ShaderData можно проанализировать с помощью цикла `for..in`, чтобы узнать, определяет ли связанный с ним шейдер какие-либо вводы и параметры. Как описано в разделе ««Получение метаданных шейдера» на странице 328», все метаданные, определенные для шейдера, также можно получить в качестве динамического свойства, добавляемого в свойство `Shader.data`.

Используя этот прием для определения вводов и параметров шейдера, проверьте тип данных динамических свойств. Если свойство является экземпляром ShaderInput, оно представляет ввод. Если свойство является экземпляром ShaderParameter, оно представляет параметр. В остальных случаях оно представляет значение метаданных. В следующем примере демонстрируется использование цикла `for..in` для анализа динамических свойств, содержащихся в свойстве `data` шейдера. Каждый ввод (объект ShaderInput) добавляется в экземпляр Vector с именем `inputs`. Каждый параметр (объект ShaderParameter) добавляется в экземпляр Vector с именем `parameters`. А свойства метаданных добавляются в экземпляр Vector с именем `metadata`. Обратите внимание, что для этого примера необходимо предварительно создать экземпляр Shader с именем `myShader`.

```
var shaderData:ShaderData = myShader.data;
var inputs:Vector.<ShaderInput> = new Vector.<ShaderInput>();
var parameters:Vector.<ShaderParameter> = new Vector.<ShaderParameter>();
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (shaderData[prop] is ShaderInput)
    {
        inputs[inputs.length] = shaderData[prop];
    }
    else if (shaderData[prop] is ShaderParameter)
    {
        parameters[parameters.length] = shaderData[prop];
    }
    else
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the inputs or properties
```

## Указание вводимых значений шейдера

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Во многих случаях шейдер ожидает одно или несколько вводимых изображений, которые используются в ходе обработки шейдера. Однако при использовании объекта Shader ввод часто указывается автоматически. Предположим, что шейдер ожидает один ввод и что он используется в качестве фильтра. Когда фильтр применяется к экранному объекту или объекту BitmapData, этот объект автоматически задается в качестве ввода. В таком случае не требуется явно задавать вводимое значение.

Однако в некоторых случаях, особенно если шейдер определяет несколько вводов, значение для ввода необходимо задавать явно. Каждый ввод, определенный в шейдере, представлен в ActionScript объектом ShaderInput. Объект ShaderInput является свойством экземпляра ShaderData в свойстве `data` объекта Shader, как описано в разделе ««Определение вводов и параметров шейдера» на странице 329». Предположим, что шейдер определяет ввод с именем `src` и что он связан с объектом Shader с именем `myShader`. В этом случае объект ShaderInput, соответствующий вводу `src`, можно получить с использованием следующего идентификатора:

```
myShader.data.src
```

Каждый объект ShaderInput имеет свойство `input`, с помощью которого задается значение для ввода. Свойству `input` назначается экземпляр BitmapData, чтобы задать графические данные. Также свойству `input` можно назначать экземпляр BitmapData или Vector.<Number> для задания бинарных или числовых данных. Дополнительные сведения об использовании экземпляра BitmapData или Vector.Экземпляр <Number> в качестве ввода, см. описание `ShaderInput.input` в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

Помимо свойства `input`, объект ShaderInput имеет свойства, с помощью которых можно определить, какой тип изображения ожидается в качестве ввода. К их числу относятся свойства `width`, `height` и `channels`. Каждый объект ShaderInput также имеет свойство `index`, с помощью которого можно определить, должно ли значение ввода передаваться явно. Если шейдер ожидает больше вводов, чем задается автоматически, значит, для оставшихся вводов значения нужно задавать явно. Дополнительные сведения о разных способах применения шейдера и о том, задаются ли значения автоматически, см. в разделе «Использование шейдера» на странице 334.

## Указание значений параметров шейдера

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

В некоторых случаях шейдер определяет значения параметров, которые используются для создания вывода. Например, шейдер, изменяющий яркость изображения, может указать параметр, определяющий степень влияния операции на яркость. Один параметр, определенный в шейдере, может ожидать одно или несколько значений в соответствии со своим определением в исходном коде. Каждый параметр, определенный в шейдере, представлен в ActionScript объектом ShaderParameter. Объект ShaderParameter является свойством экземпляра ShaderData в свойстве data объекта Shader, как описано в разделе «Определение вводов и параметров шейдера» на странице 329. Предположим, что шейдер определяет параметр с именем `brightness` и что он связан с объектом Shader с именем `myShader`. В этом случае объект ShaderParameter, соответствующий вводу `brightness`, можно получить с использованием следующего идентификатора:

```
myShader.data.brightness
```

Чтобы задать одно или несколько значений параметра шейдера, создайте массив ActionScript с одним или несколькими значениями и назначьте его свойству `value` объекта ShaderParameter. Свойство `value` определяется как экземпляр Array, так как один параметр шейдера может требовать несколько значений. Даже если параметр шейдера ожидает только одно значение, его необходимо поместить в объект Array, чтобы назначить его свойству `ShaderParameter.value`. Следующая строка задает одно значение как свойство `value`.

```
myShader.data.brightness.value = [75];
```

Если исходный код Pixel Bender для шейдера определяет значение по умолчанию для параметра, то создается массив, который содержит одно или несколько значений по умолчанию и назначается свойству `value` объекта ShaderParameter, когда создается объект Shader. После назначения массива свойству `value` значение параметра можно изменить, указав другое значение для элемента массива (даже если это массив по умолчанию). Для этого не нужно создавать новый массив и назначать его свойству `value`.

В следующем примере значение параметра шейдера задается в коде ActionScript. Шейдер определяет параметр с именем `color`. Параметр `color` объявляется как переменная `float4` в исходном коде Pixel Bender, то есть это массив из четырех чисел с плавающей запятой. В этом примере значение параметра `color` постоянно изменяется, и при каждом изменении на экране с помощью шейдера рисуется цветной прямоугольник. В результате можно наблюдать анимированное изменение цвета.

***Примечание.*** *Код для этого примера написан Райаном Тэйлором (Ryan Taylor). Мы благодарим Райана за предоставление этого кода. Ознакомиться с портфолио и статьями Райана можно по адресу www.boostworthy.com.*

Код ActionScript строится на основе трех методов.

- `init()`: в методе `init()` код загружает файл байт-кода Pixel Bender, содержащий шейдер. После загрузки файла вызывается метод `onLoadComplete()`.
- `onLoadComplete()`: в методе`onLoadComplete()` код создает объект Shader с именем `shader`. Он также создает экземпляр Sprite с именем `texture`. В методе `renderShader()` код рисует результат шейдера в объекте `texture` по одному разу в каждом кадре.
- `onEnterFrame()`: метод `onEnterFrame()` вызывается один раз для каждого кадра для создания эффекта анимации. В этом методе код задает значению параметра шейдера новый цвет, а затем вызывает метод `renderShader()`, который рисует на экране вывод шейдера в виде прямоугольника.
- `renderShader()`: в методе `renderShader()` код вызывает метод `Graphics.beginShaderFill()`, чтобы указать заливку шейдером. Затем он рисует прямоугольник, заливка которого определяется выводом шейдера (сгенерированный цвет). Дополнительные сведения об этом применении шейдера см. в разделе «Использование шейдера в качестве заливки» на странице 335.

Ниже приводится код ActionScript для этого примера. Используйте этот класс в качестве основного класса приложения для проекта, созданного только на базе ActionScript в Flash Builder, или в качестве класса документа для FLA-файла в инструменте Flash Professional.

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ColorFilterExample extends Sprite
    {
        private const DELTA_OFFSET:Number = Math.PI * 0.5;
        private var loader:URLLoader;
        private var shader:Shader;
        private var texture:Sprite;
        private var delta:Number = 0;

        public function ColorFilterExample()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ColorFilter.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            texture = new Sprite();
```

```
            addChild(texture);

            addEventListener(Event.ENTER_FRAME, onEnterFrame);
        }
        private function onEnterFrame(event:Event):void
        {
            shader.data.color.value[0] = 0.5 + Math.cos(delta - DELTA_OFFSET) * 0.5;
            shader.data.color.value[1] = 0.5 + Math.cos(delta) * 0.5;
            shader.data.color.value[2] = 0.5 + Math.cos(delta + DELTA_OFFSET) * 0.5;
            // The alpha channel value (index 3) is set to 1 by the kernel's default
            // value. This value doesn't need to change.

            delta += 0.1;

            renderShader();
        }

        private function renderShader():void
        {
            texture:graphics.clear();
            texture.graphics.beginShaderFill(shader);
            texture.graphics.drawRect(0, 0, stage.stageWidth, stage.stageHeight);
            texture.graphics.endFill();
        }
    }
}
```

Ниже представлен исходный код для ядра шейдера ColorFilter, который использовался для создания файла байт-кода Pixel Bender с именем «ColorFilter.pbj».

```
<languageVersion : 1.0;>
kernel ColorFilter
<
    namespace : "boostworthy::Example";
    vendor : "Ryan Taylor";
    version : 1;
    description : "Creates an image where every pixel has the specified color value.";
>
{
    output pixel4 result;

    parameter float4 color
    <
        minValue:float4(0, 0, 0, 0);
        maxValue:float4(1, 1, 1, 1);
        defaultValue:float4(0, 0, 0, 1);
    >;

    void evaluatePixel()
    {
        result = color;
    }
}
```

Если используется шейдер, параметры которого не задокументированы, то можно проверить свойство `type` объекта ShaderParameter, чтобы выяснить, сколько элементов какого типа должны быть включены в массив. Свойство `type` указывает тип данных параметра в соответствии с определением шейдера. Список количества и типа элементов, ожидаемых параметрами каждого типа, см. в описании свойства `ShaderParameter.value` в справочнике ActionScript® 3.0 для Adobe® Flash® Professional CS5.

Каждый объект ShaderParameter также имеет свойство `index`, которое указывает место данного параметра в ряду всех параметров шейдера. Помимо этих свойств объект ShaderParameter может иметь дополнительные свойства, содержащие значения метаданных, заданные автором шейдера. Например, автор может задать в качестве метаданных минимальное, максимальное значение параметра, а также его значение по умолчанию. Все значения метаданных, заданные автором, добавляются в объект ShaderParameter в качестве динамических свойств. Чтобы проанализировать эти свойства, с помощью цикла `for..in` пройдите через все динамические свойства объекта ShaderParameter и выявите его метаданные. В следующем примере демонстрируется использование цикла `for..in` для выявления метаданных объекта ShaderParameter. Каждое значение метаданных добавляется в экземпляр Vector с именем `metadata`. Обратите внимание, что для выполнения этого примера необходимо предварительно создать экземпляр Shader с именем `myShader`, который содержит параметр с именем `brightness`.

```
var brightness:ShaderParameter = myShader.data.brightness;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in brightness)
{
    if (brightness[prop] is String)
    {
        metadata[metadata.length] = brightness[prop];
    }
}

// do something with the metadata
```

# Использование шейдера

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

После того как шейдер Pixel Bender shader стал доступным для кода ActionScript в виде объекта Shader, его можно использовать несколькими способами.

- Заливка шейдером: шейдер определяет порцию заливки для фигуры, нарисованной с помощью API-интерфейса рисования
- Режим наложения: шейдер определяет наложение между двумя пересекающимися экранными объектами
- Фильтр: шейдер определяет фильтр, изменяющий внешний вид визуального содержимого
- Автономная обработка шейдера: обработка шейдера выполняется без указания предполагаемого использования вывода. По желанию шейдер может выполняться в фоновом режиме, а вывод становится доступным после завершения обработки. Этот прием также можно использовать для генерации растровых данных, а также для обработки невизуальных данных.

## Использование шейдера в качестве заливки

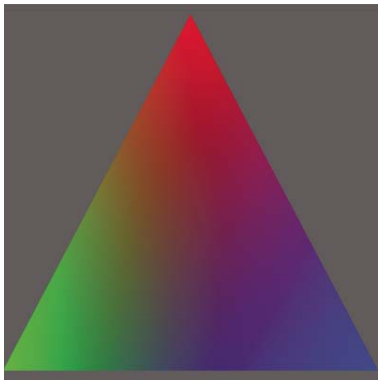
**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Когда шейдер используется для создания заливки рисунка, необходимо создать векторную фигуру с помощью API-методов рисования. Вывод шейдера используется для заливки фигуры так же, как растровое изображение используется в качестве растровой заливки с помощью API-интерфейса рисования. Чтобы создать заливку шейдером, в том месте кода, где начинается отрисовка фигуры, вызовите метод `beginShaderFill()` объекта Graphics. Передайте объект Shader в качестве первого аргумента методу `beginShaderFill()`, как показано ниже.

```
var canvas:Sprite = new Sprite();
canvas.graphics.beginShaderFill(myShader);
canvas.graphics.drawRect(10, 10, 150, 150);
canvas.graphics.endFill();
// add canvas to the display list to see the result
```

Когда шейдер используется в качестве заливки рисунка, необходимо задать значения входного изображения и значения параметров, необходимых шейдеру.

В следующем примере демонстрируется использование шейдера в качестве заливки рисунка. В данном примере шейдер создает градиент с тремя вершинами. Этот градиент имеет три цвета, каждый из них находится у вершины треугольника, между которыми образуется заливка градиентом. В дополнение, цвета меняются местами, создавая эффект анимированного вращения цветов.



***Примечание.*** *Код для этого примера написан Петри Лескиненом (Petri Leskinen). Мы благодарим Петри за предоставление этого кода. Примеры и учебные материалы, предоставленные Петри, можно найти по адресу http://pixelero.wordpress.com.*

Код ActionScript строится на трех методах.

- `init()`: метод `init()` вызывается при загрузке приложения. В этом методе код задает начальные значения для объектов Point, представляющих вершины треугольника. Также код создает экземпляр Sprite с именем `canvas`. Позднее, в методе `updateShaderFill()`, код рисует результат шейдера в объекте `canvas` по одному разу в каждом кадре. В завершение, код загружает файл байт-кода шейдера.
- `onLoadComplete()`: в методе`onLoadComplete()` код создает объект Shader с именем `shader`. Также задаются начальные значения параметров. В завершение код добавляет метод `updateShaderFill()` в качестве прослушивателя события `enterFrame`, то есть он вызывается в каждом кадре один раз для создания эффекта анимации.

- `updateShaderFill()`: метод `updateShaderFill()` вызывается один раз для каждого кадра для создания эффекта анимации. В этом методе код вычисляет и задает значения параметров шейдера. После этого код вызывает метод `beginShaderFill()`, чтобы создать заливку шейдером, а затем вызываются другие API-методы для отрисовки результата шейдера в треугольнике.

Ниже приводится код ActionScript для этого примера. Используйте этот класс в качестве основного класса приложения для проекта, созданного только на базе ActionScript в Flash Builder, или в качестве класса документа для FLA-файла в инструменте Flash Professional.

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Point;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ThreePointGradient extends Sprite
    {
        private var canvas:Sprite;
        private var shader:Shader;
        private var loader:URLLoader;

        private var topMiddle:Point;
        private var bottomLeft:Point;
        private var bottomRight:Point;

        private var colorAngle:Number = 0.0;
        private const d120:Number = 120 / 180 * Math.PI; // 120 degrees in radians


        public function ThreePointGradient()
        {
            init();
        }

        private function init():void
        {
            canvas = new Sprite();
            addChild(canvas);

            var size:int = 400;
            topMiddle = new Point(size / 2, 10);
            bottomLeft = new Point(0, size - 10);
            bottomRight = new Point(size, size - 10);

            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ThreePointGradient.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);
```

```
            shader.data.point1.value = [topMiddle.x, topMiddle.y];
            shader.data.point2.value = [bottomLeft.x, bottomLeft.y];
            shader.data.point3.value = [bottomRight.x, bottomRight.y];

            addEventListener(Event.ENTER_FRAME, updateShaderFill);
        }

        private function updateShaderFill(event:Event):void
        {
            colorAngle += .06;

            var c1:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle);
            var c2:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle + d120);
            var c3:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle - d120);

            shader.data.color1.value = [c1, c2, c3, 1.0];
            shader.data.color2.value = [c3, c1, c2, 1.0];
            shader.data.color3.value = [c2, c3, c1, 1.0];

            canvas.graphics.clear();
            canvas.graphics.beginShaderFill(shader);

            canvas.graphics.moveTo(topMiddle.x, topMiddle.y);
            canvas.graphics.lineTo(bottomLeft.x, bottomLeft.y);
            canvas.graphics.lineTo(bottomRight.x, bottomLeft.y);

            canvas.graphics.endFill();
        }
    }
}
```

Ниже представлен исходный код для ядра шейдера ThreePointGradient, который используется для создания файла байт-кода Pixel Bender с именем «ThreePointGradient.pbj».

```
<languageVersion : 1.0;>
kernel ThreePointGradient
<
    namespace : "Petri Leskinen::Example";
    vendor : "Petri Leskinen";
    version : 1;
    description : "Creates a gradient fill using three specified points and colors.";
>
{
    parameter float2 point1 // coordinates of the first point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 0);
    >;

    parameter float4 color1 // color at the first point, opaque red by default
    <
        defaultValue:float4(1.0, 0.0, 0.0, 1.0);
    >;

    parameter float2 point2 // coordinates of the second point
```

```
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 500);
    >;

    parameter float4 color2 // color at the second point, opaque green by default
    <
        defaultValue:float4(0.0, 1.0, 0.0, 1.0);
    >;

    parameter float2 point3 // coordinates of the third point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 500);
    >;

    parameter float4 color3 // color at the third point, opaque blue by default
    <
        defaultValue:float4(0.0, 0.0, 1.0, 1.0);
    >;

    output pixel4 dst;

    void evaluatePixel()
    {
        float2 d2 = point2 - point1;
        float2 d3 = point3 - point1;

        // transformation to a new coordinate system
        // transforms point 1 to origin, point2 to (1, 0), and point3 to (0, 1)
        float2x2 mtrx = float2x2(d3.y, -d2.y, -d3.x, d2.x) / (d2.x * d3.y - d3.x * d2.y);
        float2 pNew = mtrx * (outCoord() - point1);

        // repeat the edge colors on the outside
        pNew.xy = clamp(pNew.xy, 0.0, 1.0); // set the range to 0.0 ... 1.0

        // interpolating the output color or alpha value
        dst = mix(mix(color1, color2, pNew.x), color3, pNew.y);
    }
}
```

***Примечание.*** *Если заливка шейдером применяется в режиме визуализации с использованием графического процессора, цвет области заполнения будет голубым.*

Дополнительные сведения о рисовании фигур с помощью API-интерфейса рисования см. в разделе «Использование API рисования» на странице 239.

## Использование шейдера в качестве режима наложения

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Использование шейдера в качестве режима наложения очень похоже на применение других режимов наложения. Шейдер определяет внешний вид, полученный в результате визуального наложения двух экранных объектов. Чтобы использовать шейдер в качестве режима наложения, нужно назначить объект Shader свойству `blendShader` экранного объекта, который находится на переднем плане. Если задать свойству `blendShader` любое значение, кроме `null`, свойство `blendMode` экранного объекта автоматически получает значение `BlendMode.SHADER`. Следующий код демонстрирует использование шейдера в качестве режима наложения. Обратите внимание, что в этом примере есть экранный объект с именем `foreground`, который находится в списке отображения в том же контейнере, чтобы и другой экранный объект, и объект `foreground` пересекается с другим содержимым.

```
foreground.blendShader = myShader;
```

При использовании шейдера в качестве режима наложения необходимо определить не меньше двух значений ввода. Как показано в примере, код не задает значений ввода. Вместо этого, в качестве ввода шейдера автоматически используются два накладываемых изображения. Изображение, находящееся на первом плане, задается как второе значение ввода (Это экранный объект, к которому применяется режим наложения.) Фоновое изображение создается путем получения составного значения всех пикселов, находящихся за пределами ограничивающего прямоугольника изображения переднего плана. Это фоновое изображение задается как первое значение ввода. Если используется шейдер, который ожидает больше двух входных изображений, необходимо задать остальные значения.

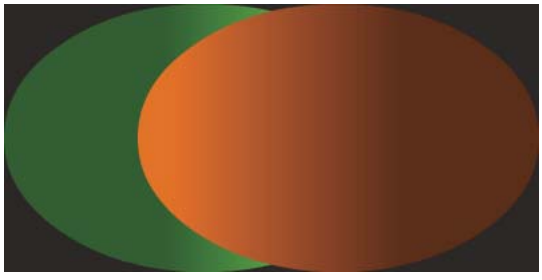
Следующий код демонстрирует использование шейдера в качестве режима наложения. В этом примере используется режим наложения «Осветлить» на базе свечения. В результате наложения значение самого светлого пиксела любого из объединяемых объектов становится значением пиксела, отображаемого на экране.

***Примечание.*** *Код для этого примера написан Марио Клингеманном (Mario Klingemann). Мы благодарим Марио за предоставление этого кода. Другие работы и статьи Марио можно найти по адресу www.quasimondo.com.*

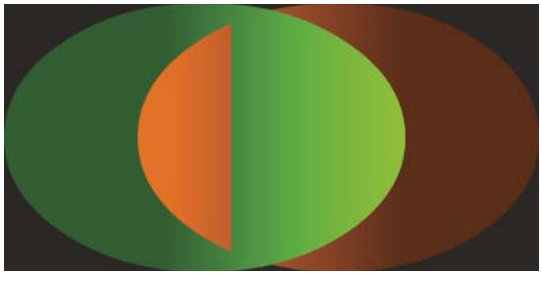
Основной код ActionScript строится на двух методах.

- `init()`: метод `init()` вызывается при загрузке приложения. В этом методе код загружает файл байт-кода шейдера.
- `onLoadComplete()`: в методе`onLoadComplete()` код создает объект Shader с именем `shader`. Затем отрисовываются три объекта. Первый объект, `backdrop`, представляет собой темно-серый фон за объединяемыми объектами. Второй объект, `backgroundShape` представляет зеленый эллипс с заливкой градиентом. Третий объект, `foregroundShape`, представляет оранжевый эллипс с заливкой градиентом.

Эллипс `foregroundShape` является объектом переднего плана для этого наложения. Фоновое изображение наложения состоит из части объекта `backdrop` и части объекта `backgroundShape`, которые попадают в область ограничивающего прямоугольника объекта `foregroundShape`. Объект `foregroundShape` находится перед всеми остальными объектами в списке отображения. На частично перекрывает объект `backgroundShape` и полностью перекрывает объект `backdrop`. Если не применяется режим наложения, в результате такого перекрытия оранжевый эллипс (`foregroundShape`) отображается полностью, а часть зеленого эллипса (`backgroundShape`) скрыта под ним.



Однако при применении режима наложения более яркая часть зеленного эллипса «просвечивает» через оранжевый, так как она светлее перекрывающей его части объекта `foregroundShape`.



Ниже приводится код ActionScript для этого примера. Используйте этот класс в качестве основного класса приложения для проекта, созданного только на базе ActionScript в Flash Builder, или в качестве класса документа для FLA-файла в инструменте Flash Professional.

```
package
{
    import flash.display.BlendMode;
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class LumaLighten extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function LumaLighten()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("LumaLighten.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            var backdrop:Shape = new Shape();
            var g0:Graphics = backdrop.graphics;
            g0.beginFill(0x303030);
            g0.drawRect(0, 0, 400, 200);
            g0.endFill();
            addChild(backdrop);

            var backgroundShape:Shape = new Shape();
            var g1:Graphics = backgroundShape.graphics;
            var c1:Array = [0x336600, 0x80ff00];
            var a1:Array = [255, 255];
            var r1:Array = [100, 255];
            var m1:Matrix = new Matrix();
            m1.createGradientBox(300, 200);
            g1.beginGradientFill(GradientType.LINEAR, c1, a1, r1, m1);
            g1.drawEllipse(0, 0, 300, 200);
```

```
            g1.endFill();
            addChild(backgroundShape);

            var foregroundShape:Shape = new Shape();
            var g2:Graphics = foregroundShape.graphics;
            var c2:Array = [0xff8000, 0x663300];
            var a2:Array = [255, 255];
            var r2:Array = [100, 255];
            var m2:Matrix = new Matrix();
            m2.createGradientBox(300, 200);
            g2.beginGradientFill(GradientType.LINEAR, c2, a2, r2, m2);
            g2.drawEllipse(100, 0, 300, 200);
            g2.endFill();
            addChild(foregroundShape);

            foregroundShape.blendShader = shader;
            foregroundShape.blendMode = BlendMode.SHADER;
        }
    }
}
```

Ниже представлен исходный код для ядра шейдера LumaLighten, который используется для создания файла байт-кода Pixel Bender с именем «LumaLighten.pbj».

```
<languageVersion : 1.0;>
kernel LumaLighten
<
    namespace : "com.quasimondo.blendModes";
    vendor : "Quasimondo.com";
    version : 1;
    description : "Luminance based lighten blend mode";
>
{
    input image4 background;
    input image4 foreground;

    output pixel4 dst;

    const float3 LUMA = float3(0.212671, 0.715160, 0.072169);

    void evaluatePixel()
    {
        float4 a = sampleNearest(foreground, outCoord());
        float4 b = sampleNearest(background, outCoord());
        float luma_a = a.r * LUMA.r + a.g * LUMA.g + a.b * LUMA.b;
        float luma_b = b.r * LUMA.r + b.g * LUMA.g + b.b * LUMA.b;

        dst = luma_a > luma_b ? a : b;
    }
}
```

Дополнительные сведения об использовании режимов наложения см. в разделе «Применение режимов смешивания» на странице 200.

***Примечание.*** *Когда программа шейдера Pixel Bender запускается в режиме наложения в Flash Player или AIR, функции сэмплинга и* `outCoord()` *ведут себя не так, как в других контекстах. В режиме наложения функция сэмплинга всегда возвращает текущий пиксел, вычисляемый шейдером. Например, нельзя добавить смещение в* `outCoord()` *с целью сэмплирования соседнего пиксела. Подобным образом, если функция* `outCoord()` *используется за пределами функции сэмплинга, ее координаты всегда равны 0. Поэтому, например, нельзя использовать положение пиксела с целью воздействия на совмещение накладываемых изображений.*

## Использование шейдера в качестве графического фильтра

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Использование шейдера в качестве графического фильтра подобно использованию других фильтров в ActionScript. Когда шейдер применяется в качестве фильтра, ему передается фильтруемое изображение (экранный объект или объект BitmapData). Шейдер использует входное изображение для создания вывода фильтра, который обычно представляет собой измененную версию исходного изображения. Если фильтруемое изображение является экранным объектом, вывод шейдера отображается на экране вместо него. Если фильтруемый объект является объектом BitmapData, результат шейдера становится содержимым объекта BitmapData, для которого вызван метод `applyFilter()`.

Для использования шейдера в качестве фильтра сначала нужно создать объект Shader, как описано в разделе «Загрузка или встраивание шейдера» на странице 327. После этого нужно создать объект ShaderFilter, связанный с объектом Shader. Объект ShaderFilter представляет собой фильтр, который применяется к фильтруемому объекту. Он применяется к объекту так же, как и обычный фильтр. Этот объект передается свойству `filters` экранного объекта или методу `applyFilter()`, вызываемому для объекта BitmapData. Например, следующий код создает объект ShaderFilter и применяет фильтр к экранному объекту с именем `homeButton`.

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
homeButton.filters = [myFilter];
```

При использовании шейдера в качестве фильтра необходимо определить хотя бы одно значение ввода. Как показано в примере, код не задает значения ввода. Вместо этого, в качестве входного изображения задается фильтруемый экранный объект или объект BitmapData. Если используется шейдер, который ожидает больше одного входного изображения, необходимо задать остальные значения.

В некоторых случаях фильтр изменяет размеры исходного изображения. Например, типичный эффект тени добавляет к изображению дополнительные пикселы, содержащие тень. При использовании шейдера, который меняет размеры изображения, необходимо задать свойства `leftExtension`, `rightExtension`, `topExtension` и `bottomExtension`, чтобы указать, насколько должен измениться размер изображения.

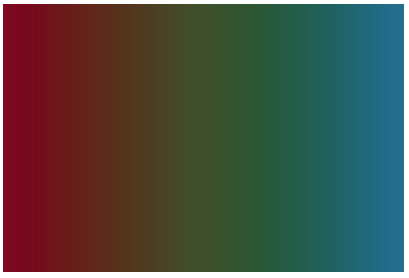
Следующий пример демонстрирует использование шейдера в качестве фильтра. Фильтр инвертирует значения красного, зеленого и синего каналов изображения. Результат представляет собой «негатив» изображения.

***Примечание.*** *В данном примере в качестве шейдера используется ядро Pixel Bender invertRGB.pbk, которое включено в Pixel Bender Toolkit. Исходный код этого ядра можно загрузить в каталоге установки Pixel Bender Toolkit. Скомпилируйте исходный код и сохраните файл байт-кода в том же каталоге, что исходный код ActionScript.*

Основной код ActionScript строится на двух методах.

- `init()`: метод `init()` вызывается при загрузке приложения. В этом методе код загружает файл байт-кода шейдера.

- `onLoadComplete()`: в методе`onLoadComplete()` код создает объект Shader с именем `shader`. Затем он создает и отрисовывает содержимое объекта `target`. Объект `target` представляет собой прямоугольник, заполненный линейным градиентом цвета: красный слева, желто-зеленый в середине и голубой справа. Нефильтрованный объект выглядит так:



После применения фильтра цвета инвертируются, и прямоугольник выглядит так:

В данном примере в качестве шейдера используется ядро Pixel Bender «invertRGB.pbk», которое включено в Pixel Bender Toolkit. сходный код находится в файле «invertRGB.pbk» в каталоге установки Pixel Bender Toolkit. Скомпилируйте исходный код и сохраните файл байт-кода с именем «invertRGB.pbj» в той же папке, что исходный код ActionScript.

Ниже приводится код ActionScript для этого примера. Используйте этот класс в качестве основного класса приложения для проекта, созданного только на базе ActionScript в Flash Builder, или в качестве класса документа для FLA-файла в инструменте Flash Professional.

```
package
{
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.filters.ShaderFilter;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class InvertRGB extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function InvertRGB()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("invertRGB.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            var target:Shape = new Shape();
            addChild(target);

            var g:Graphics = target.graphics;
            var c:Array = [0x990000, 0x445500, 0x007799];
            var a:Array = [255, 255, 255];
            var r:Array = [0, 127, 255];
            var m:Matrix = new Matrix();
            m.createGradientBox(w, h);
            g.beginGradientFill(GradientType.LINEAR, c, a, r, m);
            g.drawRect(10, 10, w, h);
            g.endFill();

            var invertFilter:ShaderFilter = new ShaderFilter(shader);
            target.filters = [invertFilter];
        }
    }
}
```

Дополнительные сведения о применении фильтров см. в разделе «Создание и применение фильтров» на странице 290.

## Использование шейдера в автономном режиме

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Когда шейдер используется в автономном режиме, его обработка выполняется независимо от предполагаемого применения вывода. Нужно указать шейдер для выполнения, задать значения ввода и параметры, выбрать объект, в который будет помещен вывод. Использование шейдера в автономном режиме может потребоваться по двум причинам.

- Обработка неграфических данных. В автономном режиме шейдеру можно передать произвольные двоичные или числовые данные вместо данных растрового изображения. Можно указать, что шейдер должен вернуть полученный результат как двоичные или числовые данные, в дополнение к данным растрового изображения.
- Фоновая обработка. При работе в автономном режиме шейдер по умолчанию выполняется асинхронно. Это означает, что шейдер выполняется в фоне, не вмешиваясь в работу приложения. Вывод передается коду после завершения обработки. Так можно использовать шейдер, на выполнение которого требуется много времени, чтобы он не «замораживал» пользовательский интерфейс приложения и другие операции обработки.

Объект ShaderJob служит для выполнения шейдера в автономном режиме. Сначала создается объект ShaderJob и связывается с объектом Shader, представляющим шейдер, который нужно выполнить.

```
var job:ShaderJob = new ShaderJob(myShader);
```

Затем задаются значения ввода или параметры, необходимые для работы шейдера. Если шейдер выполняется в фоновом режиме, также необходимо зарегистрировать прослушиватель для события `complete` объекта ShaderJob. Прослушиватель вызывается после завершения работы шейдера.

```
function completeHandler(event:ShaderEvent):void
{
    // do something with the shader result
}

job.addEventListener(ShaderEvent.COMPLETE, completeHandler);
```

Затем создается объект, в который будет записан вывод шейдера после завершения операции. Этот объект назначается свойству `target` объекта ShaderJob.

```
var jobResult:BitmapData = new BitmapData(100, 75);
job.target = jobResult;
```

Назначьте свойству `target` объект BitmapData, если ShaderJob используется для обработки изображения. Если же обрабатываются двоичные или числовые данные, назначьте объект ByteArray или Vector.<Number> свойству `target`. В этом случае необходимо задать свойства `width` и `height` объекта ShaderJob, чтобы указать объем данных для вывода в объект `target`.

***Примечание.*** *Чтобы одновременно задать свойства `shader`, `target`, `width` и `height` объекта ShaderJob, нужно передать аргументы конструктору `ShaderJob()`, например: `var job:ShaderJob = new ShaderJob(myShader, myTarget, myWidth, myHeight);`*

Когда потребуется выполнить шейдер, вызовите метод `start()` объекта ShaderJob.

```
job.start();
```

По умолчанию при вызове метода `start()` запускается асинхронное выполнение ShaderJob. В этом случае выполнение программы сразу переходит к следующей строке кода, а не ожидает завершения обработки шейдера. После завершения операции шейдера объект ShaderJob вызывает свои прослушиватели события `complete`, сообщая им о готовности. На этом этапе (то есть в теле прослушивателя события `complete`) объект `target` содержит результат операции шейдера.

***Примечание.*** *Вместо использования объекта, назначенного свойству `target`, вывод шейдера непосредственно можно получить из объекта события, переданного методу прослушивателя. Объект события является экземпляром ShaderEvent. Объект ShaderEvent имеет три свойства, которые можно использовать для получения вывода в зависимости от типа данных объекта, заданного в свойстве `target`: `ShaderEvent.bitmapData`, `ShaderEvent.byteArray` и `ShaderEvent.vector`.*

Также можно передать аргумент `true` методу `start()`. В этом случае операция шейдера выполняется синхронно. Выполнение всего кода (включая взаимодействие с пользовательским интерфейсом и другими событиями) приостанавливается до завершения обработки шейдера. После выполнения операции шейдера объект `target` получает вывод шейдера, и программа переходит к следующей строке кода.

```
job.start(true);
```

# Глава 16. Работа с фрагментами роликов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс MovieClip — это основной класс, используемый для создания анимации и символа фрагмента ролика в среде разработки Adobe® Flash®. Ему присущи все функции и виды поведения экранных объектов, но в дополнение к ним он имеет свойства и методы для управления своей временной шкалой.

## Основные сведения о фрагментах роликов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Фрагменты роликов являются ключевым элементом для тех, кто создает анимированное содержимое с помощью инструмента разработки Flash и хочет управлять им с использованием ActionScript. При создании символа фрагмента ролика в Flash программа добавляет его в библиотеку документа Flash. По умолчанию символ становится экземпляром класса MovieClip и в этом качестве получает свойства и методы класса MovieClip.

Когда экземпляр символа фрагмента ролика помещается в рабочую область, фрагмент ролика автоматически продвигается по временной шкале, если он содержит больше одного кадра и его воспроизведение не изменено с помощью ActionScript. Именно временная шкала отличает класс MovieClip от других и позволяет создавать анимацию посредством движения и анимации фигур в инструменте разработки Flash. А для экранного объекта, являющегося экземпляром класса Sprite, напротив, анимацию можно создавать только путем программированного изменения значений объекта.

В предыдущих версиях ActionScript, класс MovieClip являлся базовым классом всех экземпляров в рабочей области. В ActionScript 3.0 фрагмент ролика — это лишь один из многих экранных объектов, появляющихся на экране. Если временная шкала не является обязательной для функционирования экранного объекта, то использование класса Shape или Sprite вместо класса MovieClip может увеличить производительность визуализации. Дополнительные сведения по выбору соответствующего экранного объекта для конкретной задачи см. в разделе «Выбор подкласса DisplayObject» на странице 185.

### Важные понятия и термины

Ниже приводится список важных терминов, необходимых при работе с фрагментами роликов.

**AVM1 SWF** SWF-файл, созданный с помощью ActionScript 1.0 или ActionScript 2.0, как правило, для проигрывателя Flash Player 8 или более ранних версий.

**AVM2 SWF** SWF-файл, созданный с помощью ActionScript 3.0 для Adobe Flash Player 9 и более поздних версий или Adobe AIR.

**Внешний SWF** SWF-файл, который создан отдельно от SWF-файла проекта и предназначен для загрузки и воспроизведения в SWF-файле проекта.

**Кадр** Наименьшее деление времени на временной шкале. Как и на киноленте, каждый кадр похож на снимок анимации в определенное время, и когда кадры последовательно воспроизводятся в быстром темпе, создается эффект анимации.

**Временная шкала** Образное представление последовательности кадров, образующих последовательность анимации фрагмента ролика. Временная шкала объекта MovieClip является эквивалентом временной шкалы в инструменте разработки Flash.

**Точка воспроизведения** Маркер, указывающий место (кадр) на временной шкале, которое воспроизводится в данный момент.

# Работа с объектами MovieClip

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При публикации SWF-файла Flash преобразует все экземпляры символа фрагмента ролика, помещенные в рабочую область, в объекты MovieClip. Символ фрагмента ролика можно сделать доступным для ActionScript, присвоив ему имя в поле «Имя экземпляра» в инспекторе свойств. При создании SWF-файла Flash генерирует код, создающий экземпляр MovieClip в рабочей области, и объявляет переменную с использованием имени экземпляра. Если вы присвоили имена фрагментам роликов, которые вложены в другие названные фрагменты роликов, дочерние фрагменты обрабатываются как свойства родительного фрагмента ролика. Таким образом, чтобы вызвать дочерний фрагмент ролика, нужно использовать синтаксис записи через точку. Например, если фрагмент ролика с именем экземпляра `childClip` вложен в другой фрагмент ролика с именем `parentClip`, анимацию временной шкалы дочернего фрагмента можно воспроизвести, вызвав этот код:

```
parentClip.childClip.play();
```

***Примечание.*** *Дочерние экземпляры, помещенные в рабочую область в инструменте разработки Flash, нельзя вызвать с помощью кода из конструктора родительского экземпляра, так как они еще не созданы на момент выполнения кода. Перед обращением к потомку предок должен либо создать дочерний экземпляр с помощью кода, либо отложить вызов, пока функция обратного вызова, прослушивающая события потомка, не отправит его событие* `Event.ADDED_TO_STAGE`.

Хотя некоторые устаревшие методы и свойства класса MovieClip в ActionScript 2.0 остались прежними, в другие были внесены изменения. Все свойства, в названии которых использовался префикс с символом подчеркивания, были переименованы. Например, свойства `_width` и `_height` теперь называются `width` и `height`, а `_xscale` и `_yscale` стали называтьсяscaleX и `scaleY`. Полный список свойств и методов класса MovieClip см. в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

# Управление воспроизведением фрагмента ролика

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Flash использует концепцию временной шкалы для передачи анимации или изменения состояния. Любой визуальный элемент, использующий временную шкалу, должен являться объектом MovieClip или расширением класса MovieClip. Хотя ActionScript может предписать любому фрагменту ролика начать, остановить воспроизведение или перейти к другой точке на временной шкале, с помощью кода нельзя динамически создать временную шкалу или добавлять содержимое в определенные кадры — это можно сделать только в инструменте разработки Flash.

Когда воспроизводится экземпляр MovieClip, точка воспроизведения перемещается вперед по временной шкале со скоростью, определяемой частотой кадров SWF-файла. Эту настройку можно также переопределить, задав свойство `Stage.frameRate` в ActionScript.

## Начало и остановка воспроизведения фрагментов роликов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Методы `play()` и `stop()` обеспечивают основные элементы управления продвижением фрагмента ролика по временной шкале. Предположим, в рабочей области есть символ фрагмента ролика, содержащий анимацию велосипеда, проезжающего от одного конца экрана до другого, с именем экземпляра `bicycle`. Если присоединить следующий код к ключевому кадру на основной временной шкале,

```
bicycle.stop();
```

велосипед не будет двигаться (его анимация не будет воспроизводиться). Движение велосипеда можно начать посредством другого взаимодействия. Например, если добавить кнопку с именем `startButton`, следующий код в ключевом кадре на основной временной шкале позволит нажатием кнопки воспроизвести анимацию.

```
// This function will be called when the button is clicked. It causes the
// bicycle animation to play.
function playAnimation(event:MouseEvent):void
{
    bicycle.play();
}
// Register the function as a listener with the button.
startButton.addEventListener(MouseEvent.CLICK, playAnimation);
```

## Перемотка вперед и назад

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Контролировать воспроизведение фрагмента ролика можно не только с помощью методов `play()` и `stop()`. Также можно перемещать точку воспроизведения вперед или назад по временной шкале с помощью методов `nextFrame()` и `prevFrame()`. Вызов любого из этих методов останавливает воспроизведение и перемещает точку воспроизведения соответственно на один кадр вперед или назад.

Использование метода `play()` аналогично вызову `nextFrame()` каждый раз, когда отправляется событие `enterFrame` фрагмента ролика. Подобным образом можно воспроизвести фрагмент ролика `bicycle` в обратном направлении, если создать прослушиватель события `enterFrame` и с помощью функции прослушивателя предписать экземпляру `bicycle` переходить к предыдущему кадру, как показано в следующем примере.

```
// This function is called when the enterFrame event is triggered, meaning
// it's called once per frame.
function everyFrame(event:Event):void
{
    if (bicycle.currentFrame == 1)
    {
        bicycle.gotoAndStop(bicycle.totalFrames);
    }
    else
    {
        bicycle.prevFrame();
    }
}
bicycle.addEventListener(Event.ENTER_FRAME, everyFrame);
```

Если фрагмент ролика содержит более одного кадра, то при обычном воспроизведении он будет воспроизводиться бесконечно, то есть после прохождения последнего кадра он будет возвращаться к кадру 1. Когда используется метод `prevFrame()` или `nextFrame()`, это поведение не включается автоматически (вызов метода `prevFrame()` при достижении кадра 1 не переводит точку воспроизведения на последний кадр). Условие `if` в описанном выше примере проверяет, дошла ли точка воспроизведения до первого кадра, и устанавливает ее на последний кадр, чтобы создать непрерывный цикл обратного воспроизведения фрагмента ролика.

## Переход к другому кадру с помощью меток кадров

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Отправить точку воспроизведения фрагмента ролика на другой кадр совсем не трудно. Вызов метода `gotoAndPlay()` или `gotoAndStop()` переведет точку воспроизведения фрагмента ролика к кадру с порядковым номером, указанным в качестве параметра. Также можно передать строку, соответствующую имени метки кадра. Любому кадру на временной шкале можно присвоить метку. Для этого выберите кадр на временной шкале и введите имя в поле «Метка кадра» в инспекторе свойств.

Преимущества использования меток, а не номеров кадров, особенно очевидны при создании сложного фрагмента ролика. Когда количество кадров, слоев и анимаций движения становится очень большим, следует присвоить важным кадрам описательные метки, обозначающие изменение поведения фрагмента ролика (например, «выкл.», «пешком», «бегом»). Это улучшает удобочитаемость кода и предоставляет гибкость, так как вызовы ActionScript, переходящие к кадру с меткой, указывают на уникальную точку (метку), а не на номер кадра. Если позднее вы решите переместить определенный сегмент анимации в другой кадр, то вносить изменения в код ActionScript не потребуется, так как для кадров в новом местоположении сохранится прежняя метка.

Для представления меток кадров в коде ActionScript 3.0 используется класс FrameLabel. Каждый экземпляр этого класса представляет одну метку кадра и имеет свойство `name`, представляющее имя метки кадра, заданное в инспекторе свойство, и свойство `frame`, представляющее номер кадра на временной шкале, которому присвоена метка.

Чтобы получить доступ к экземплярам FrameLabel, связанным с экземпляром фрагмента ролика, класс MovieClip включает два свойства, напрямую возвращающие объекты FrameLabel. Свойство `currentLabels` возвращает массив, содержащий все объекты FrameLabel на временной шкале фрагмента ролика. Свойство `currentLabel` возвращает строку с именем последней пройденной метки кадра на временной шкале.

Допустим, что вы создали фрагмент ролика с именем `robot` и пометили различные состояния его анимации. Можно настроить условие, которое проверяет свойство `currentLabel` для получения текущего состояния объекта `robot`, как в следующем коде.

```
if (robot.currentLabel == "walking")
{
    // do something
}
```

Проигрыватель Flash Player 11.3 и приложение AIR 3.3 добавили событие `frameLabel` в класс FrameLabel. Экземпляру FrameLabel, представляющему метку кадра, можно назначить обработчик событий. Событие отправляется, когда указатель воспроизведения достигает кадра.

В следующем примере создается экземпляр FrameLabel для второй метки кадра в массиве меток кадров для MovieClip. Затем регистрируется обработчик события `frameLabel`:

```
var myFrameLabel:FrameLabel = robot.currentLabels[1];
myFrameLabel.addEventListener(Event.FRAME_LABEL, onFrameLabel);

function onFrameLabel(e:Event):void {
    //do something
}
```

## Работа с монтажными кадрами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В среде разработки Flash монтажные кадры можно использовать для разделения последовательных временных шкал, через которые проходит SWF-файл. Используя второй параметр методов `gotoAndPlay()` и `gotoAndStop()`, можно задать монтажный кадр, к которому должна перейти точка воспроизведения. Все FLA-файлы начинаются только с исходного монтажного кадра, но можно создавать и другие монтажные кадры.

Использование монтажных кадров не всегда целесообразно, так как они имеют ряд недостатков. Документ Flash, содержащий много монтажных кадров, сложно обрабатывать, особенно в среде с несколькими авторами. Создание нескольких монтажных кадров также приводит к неэффективному использованию пропускной способности, так как в процессе публикации все монтажные кадры объединяются в одну временную шкалу. Это приводит к последовательной загрузке всех монтажных кадров, даже если они никогда не воспроизводились. По этим причинам использование нескольких монтажных кадров, как правило, не рекомендуется, за исключением случаев, когда требуется организовать несколько продолжительных анимаций на базе временной шкалы.

Свойство `scenes` класса MovieClip возвращает массив объектов Scene, представляющих все монтажные кадры в SWF-файле. Свойство `currentScene` возвращает объект Scene, представляющий монтажный кадр, который воспроизводится в данный момент.

Класс Scene имеет несколько свойств, предоставляющих информацию о монтажном кадре. Свойство `labels` возвращает массив объектов FrameLabel, представляющих метки кадров в данном монтажном кадре. Свойство `name` возвращает имя монтажного кадра в виде строки. Свойство `numFrames` возвращает целое число, представляющее общее количество кадров в монтажном кадре.

# Создание объектов MovieClip с помощью ActionScript

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Один из способов добавления содержимого на экран в Flash заключается в перетаскивании ресурсов из библиотеки в рабочую область, но это не единственный поток операций. Для сложных проектов опытные разработчики, как правило, предпочитают создавать фрагменты роликов с использованием программного кода. Этот подход дает несколько преимуществ: простота повторного использования кода, быстрота компиляции и более сложные модификации, которые доступны только в ActionScript.

API-интерфейс списка отображения в ActionScript 3.0 упрощает процесс динамического создания объектов MovieClip. Возможность непосредственного создания экземпляра MovieClip независимо от процесса его добавления в список отображения обеспечивает гибкость и простоту, сохраняя полный контроль.

При создании фрагмента ролика (или другого экранного объекта) с помощью ActionScript 3.0 его экземпляр отображается на экране только после добавления в список отображения посредством вызова метода `addChild()` или `addChildAt()` для контейнера экранных объектов. Это позволяет создать фрагмент ролика, задать его свойства и даже вызвать те или иные методы, прежде чем он будет визуализирован на экране. Дополнительные сведения по работе со списком отображения см. в разделе «Работа с контейнерами экранных объектов» на странице 172.

## Экспорт библиотечных символов для ActionScript

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

По умолчанию экземпляры символов фрагментов роликов в библиотеке документа Flash не могут создаваться динамически (то есть только с использованием ActionScript). Это вызвано тем, что каждый символ, экспортированный для использования в ActionScript, увеличивает размер SWF-файла. К тому же, некоторые символы не должны использоваться в рабочей области. По этой причине необходимо указать, что символ должен быть экспортирован для ActionScript, чтобы он стал доступным в ActionScript.

**Экспорт символов для ActionScript**

1. Выберите символ на панели «Библиотека» и откройте диалоговое окно «Свойства символа».
2. При необходимости активируйте дополнительные настройки.
3. В разделе «Связывание» установите флажок «Экспорт для ActionScript».

   Это активирует поля «Класс» и «Базовый класс».

   По умолчанию поле «Класс» заполняется названием символа без пробелов (например, символ с именем «Tree House» будет преобразован в «TreeHouse»). Чтобы указать, что символ должен использовать поведение пользовательского класса, введите в этом поле полное имя класса, включая пакет. Если требуется создавать экземпляры символа в ActionScript, но не нужно добавлять дополнительное поведение, имя класса можно оставить без изменений.

   Поле «Базовый класс» по умолчанию заполнено значением `flash.display.MovieClip`. Если требуется, чтобы символ расширял функции другого пользовательского класса, вместо этого значения можно указать имя класса, если он расширяет класс Sprite (или MovieClip).
4. Нажмите кнопку «OK», чтобы сохранить изменения.

   Если на этом этапе Flash не может найти связанный SWC-файл или внешний файл ActionScript с определением указанного класса (например, если не требовалось добавлять дополнительное поведение для символа), отображается предупреждение:

   *Определение для этого класса не найдено в подкаталоге классов. Оно будет автоматически сгенерировано в SWF-файле при экспорте.*

   Это предупреждение можно игнорировать, если библиотечный символ не требует уникальных функций в дополнение к возможностям класса MovieClip.

Если не предоставить для символа класс, Flash автоматически создаст для него класс, эквивалентный этому:

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

    public class ExampleMovieClip extends MovieClip
    {
        public function ExampleMovieClip()
        {
        }
    }
}
```

Если требуется добавить для символа дополнительные возможности ActionScript, добавьте соответствующие свойства и методы в приведенную ниже структуру кода. Предположим, что имеется символ фрагмента ролика, содержащий круг с шириной и высотой 50 пикселов, и этот символ должен экспортироваться для ActionScript с классом Circle. При добавлении в файл Circle.as следующий код расширяет класс MovieClip и предоставляет для символа дополнительные методы `getArea()` и `getCircumference()`.

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

    public class Circle extends MovieClip
    {
        public function Circle()
        {
        }

        public function getArea():Number
        {
        // The formula is Pi times the radius squared.
        return Math.PI * Math.pow((width / 2), 2);
        }

        public function getCircumference():Number
        {
        // The formula is Pi times the diameter.
        return Math.PI * width;
        }
    }
}
```

Следующий код, добавленный в ключевой кадр в кадре 1 документа Flash, создаст экземпляр символа и добавит его в список отображения.

```
var c:Circle = new Circle();
addChild(c);
trace(c.width);
trace(c.height);
trace(c.getArea());
trace(c.getCircumference());
```

Этот код демонстрирует создание экземпляров с помощью ActionScript вместо перетаскивания отдельных ресурсов в рабочую область. Код создает круг, который обладает всеми свойствами фрагмента ролика, а также пользовательскими методами, определенными в классе Circle. Это самый простой пример. Библиотечный символ может задавать любое количество свойств и методов в своем классе.

Создание экземпляров на базе ActionScript — это очень мощное средство, так как оно позволяет динамически создавать большое число экземпляров, что было бы утомительно делать вручную. Кроме того, этот метод обеспечивает гибкость, так как в ходе создания каждого экземпляра можно настроить его свойства. Чтобы полностью оценить эти преимущества, можно создать несколько экземпляров Circle с помощью цикла. Используя описанные ранее символ и класс Circle в библиотеке документа Flash, поместите следующий код в ключевой кадр первого кадра.

```
import flash.geom.ColorTransform;

var totalCircles:uint = 10;
var i:uint;
for (i = 0; i < totalCircles; i++)
{
    // Create a new Circle instance.
    var c:Circle = new Circle();
    // Place the new Circle at an x coordinate that will space the circles
    // evenly across the Stage.
    c.x = (stage.stageWidth / totalCircles) * i;
    // Place the Circle instance at the vertical center of the Stage.
    c.y = stage.stageHeight / 2;
    // Change the Circle instance to a random color
    c.transform.colorTransform = getRandomColor();
    // Add the Circle instance to the current timeline.
    addChild(c);
}

function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

Этот пример демонстрирует, как можно быстро создать и настроить несколько экземпляров символа с помощью кода. Каждый экземпляр размещается на базе текущего счетчика в цикле, и каждому из них присваивается произвольный цвет с помощью свойства `transform` (унаследованное классом Circle в результате расширения класса MovieClip).

# Загрузка внешнего SWF-файла

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В ActionScript 3.0 SWF-файлы загружаются с помощью класса Loader. Чтобы загрузить внешний SWF-файл, код ActionScript должен выполнить четыре действия:

1. создать новый объект URLRequest с URL-адресом файла;
2. создать новый объект Loader;
3. вызвать метод `load()` объекта Loader, передав в качестве параметра экземпляр URLRequest.

4 вызвать метод `addChild()` для контейнера экранных объектов (такого как основная временная шкала документа Flash), чтобы добавить экземпляр Loader в список отображения.

В итоге код выглядит следующим образом:

```
var request:URLRequest = new URLRequest("http://www.[yourdomain].com/externalSwf.swf");
var loader:Loader = new Loader()
loader.load(request);
addChild(loader);
```

Этот же код можно использовать для загрузки графических файлов, например изображений в формате JPEG, GIF или PNG. Для этого нужно указать URL-адрес изображения, а не SWF-файла. В отличие от изображения, SWF-файл может содержать ActionScript. Таким образом, несмотря на сходство процессов загрузки SWF-файла и изображения, при загрузке внешнего SWF-файла оба SWF-файла, загружаемый и загружающий, должны находиться в одной изолированной программной среде, если SWF воспроизводится в Flash Player или AIR и для работы с внешним SWF-файлом планируется использовать ActionScript. Кроме того, если внешний SWF-файл содержит классы из того же пространства имен, что и классы в загружающем SWF-файле, возможно, потребуется создать новый домен приложения для загружаемого SWF-файла, чтобы предотвратить конфликты пространства имен. Дополнительные сведения по безопасности и рекомендации по домену программы см. в разделах ««Работа с доменами приложений» на странице 158» и ««Загрузка содержимого» на странице 1142».

После успешной загрузки внешний SWF-файл можно получить через свойство `Loader.content`. Если внешний SWF-файл опубликован для ActionScript 3.0, это будет фрагмент ролика или спрайт в зависимости от того, какой класс он расширяет.

В среде Adobe AIR для iOS загрузка SWF-файла имеет некоторые отличия по сравнению с другими платформами. Дополнительные сведения см. в разделе «Загрузка SWF-файлов в среде AIR для iOS» на странице 218.

## Особенности загрузки SWF-файлов предыдущих версий

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если внешний SWF-файл опубликован с помощью одной из предшествующих версий ActionScript, следует учитывать важные ограничения. В отличие от SWF-файла версии ActionScript 3.0, выполняемого в среде AVM2 (ActionScript Virtual Machine 2), SWF-файл, опубликованный для ActionScript 1.0 или 2.0, выполняется в среде AVM1 (ActionScript Virtual Machine 1).

Между загрузкой SWF-файла ActionScript 1.0 или 2.0 в SWF-файл ActionScript 3.0 по сравнению с загрузкой SWF-файла ActionScript 3.0 имеются важные различия. Проигрыватель Flash Player обеспечивает полную обратную совместимость с ранее опубликованным содержимым. Любое содержимое, работающее в предыдущих версиях Flash Player, работает и в версиях Flash Player, поддерживающих ActionScript 3.0. Тем не менее, существует ряд ограничений.

- Код ActionScript 3.0 может загружать SWF-файл, написанный на языке ActionScript 1.0 или 2.0. После того как SWF-файл, написанный на языке ActionScript 1.0 или 2.0, будет успешно загружен, загруженный объект (свойство `Loader.content` ) становится объектом AVM1Movie. Экземпляр AVM1Movie отличается от экземпляра MovieClip. Это экранный объект, но, в отличие от фрагмента ролика, он не включает методов и свойств, связанных с временной шкалой. Родительский SWF-файл версии AVM2 не может иметь доступа к свойствам, методам или объектам загруженного объекта AVM1Movie.
- SWF-файлы, написанные в ActionScript 1.0 или 2.0, не могут загружать SWF-файлы, написанные в ActionScript 3.0. Это означает, что SWF-файлы, созданные в Adobe Flash 8 или Flex Builder 1.5, а также в более ранних версиях этих программ, не смогут загружать SWF-файлы в ActionScript 3.0.

Единственным исключением из этого правила является то, что SWF-файл ActionScript 2.0 может заменить себя SWF-файлом ActionScript 3.0, если для этого SWF-файла ActionScript 2.0 не выполнялась какая-либо загрузка ни на один из его уровней. В SWF-файле ActionScript 2.0 это можно выполнить с помощью вызова `loadMovieNum()`, передав значение 0 параметру `level`.

- В общем случае для SWF-файлов, написанных в ActionScript 1.0 или 2.0, необходимо выполнять перенос, если их предполагается использовать совместно с SWF-файлами ActionScript 3.0. Например, предположим, что создается проигрыватель мультимедийных данных с помощью ActionScript 2.0. Этот проигрыватель загружает различное содержимое, которое также было создано с помощью ActionScript 2.0. Нельзя создать новое содержимое в ActionScript 3.0 и загрузить его в этот проигрыватель мультимедийных данных. Необходимо выполнить перенос видеопроигрывателя в ActionScript 3.0.

  Однако если проигрыватель мультимедийных данных создан в ActionScript 3.0, он сможет выполнять отдельные загрузки содержимого ActionScript 2.0.

В следующих таблицах показаны ограничения предыдущих версий проигрывателя Flash Player в отношении загрузки более нового содержимого и выполнения программного кода, а также ограничения для взаимодействия между сценариями в SWF-файлах, написанных в различных версиях ActionScript.

| **Поддерживаемые функции** | **Flash Player 7** | **Flash Player 8** | **Для Flash Player 9 и 10** |
|---|---|---|---|
| Может загружать SWF-файлы, опубликованные для | 7 или более ранней версии | 8 или более ранней версии | 9 (или 10) и более ранних |
| Содержит этот AVM | AVM1 | AVM1 | AVM1 и AVM2 |
| Выполняет SWF-файлы, записанные в ActionScript | 1.0 и 2.0 | 1.0 и 2.0 | 1.0, 2.0 и 3.0 |

В следующей таблице «Поддерживаемые функции» относится к содержимому, выполняемому в проигрывателях Flash Player 9 или более поздней версии. Содержимое, выполняемое в проигрывателе Adobe Flash Player 8 или более ранней версии, может быть загружено, просмотрено и выполнено, а также использовано при вызове из других сценариев только в ActionScript 1.0 и 2.0.

| **Поддерживаемые функции** | **Содержимое, созданное в ActionScript 1.0 и 2.0** | **Содержимое, созданное в ActionScript 3.0** |
|---|---|---|
| Может загружать и выполнять программный код в содержимом, созданном в | ActionScript 1.0 и 2.0 | ActionScript 1.0 и 2.0, а также ActionScript 3.0 |
| Может вызывать из сценариев содержимое, созданное в | ActionScript 1.0 и 2.0 (ActionScript 3.0 через LocalConnection) | ActionScript 1.0 и 2.0 через LocalConnection.<br>ActionScript 3.0 |

# Пример фрагмента ролика: RuntimeAssetsExplorer

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Функция «Экспорт для ActionScript» особенно полезна для библиотек, которые могут использоваться для нескольких проектов. Если ПО Flash Player или AIR выполняет SWF-файл, символы, экспортированные в ActionScript, доступны любому SWF-файлу, находящемуся в той же изолированной программной среде, что и загружающий его SWF-файл. Таким образом, один документ Flash может сгенерировать SWF-файл, предназначенный исключительно для хранения графических объектов. Этот прием особенно полезен для

больших проектов, в рамках которых дизайнеры, создающие визуальные ресурсы, могут работать параллельно с дизайнерами, создающими «упаковочный» SWF-файл, загружающий SWF-файл с графическими ресурсами во время выполнения. Этот метод можно использовать для обработки последовательности разных версий файлов, в которых графические ресурсы не зависят от процесса разработки программы.

Приложение RuntimeAssetsExplorer загружает любой SWF-файл, являющийся подклассом для RuntimeAsset, и позволяет просматривать доступные активы этого SWF-файла. Пример иллюстрирует следующие процессы:

- загрузка внешнего SWF-файла с помощью метода `Loader.load()`;
- динамическое создание библиотечного символа, экспортированного для ActionScript;
- управление воспроизведением экземпляра MovieClip с помощью ActionScript.

Прежде чем начинать, обратите внимание на то, что все SWF-файлы, предназначенные для воспроизведения в проигрывателе Flash Player, должны находиться в одной изолированной программной среде. Дополнительные сведения см. в разделе «Изолированные программные среды» на странице 1123.

Получить файлы приложения для этого примера можно на странице образцов Flash Professional. Файлы приложения RuntimeAssetsExplorer находятся в папке Samples/RuntimeAssetsExplorer. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| RuntimeAssetsExample.mxml<br>или<br>RuntimeAssetsExample.fla | Пользовательский интерфейс приложения для Flex (MXML) или Flash (FLA). |
| RuntimeAssetsExample.as | Класс документа для приложения Flash (FLA). |
| GeometricAssets.as | Пример класса, реализующего интерфейс RuntimeAsset. |
| GeometricAssets.fla | FLA-файл, связанный с классом GeometricAssets (класс документа для FLA), который содержит символы, экспортированные для ActionScript. |
| com/example/programmingas3/runtimeassetexplorer/RuntimeLibrary.as | Интерфейс, определяющий обязательные методы, которые должны иметь все SWF-файлы, содержащие динамические ресурсы, загружаемые в контейнер обозревателя во время выполнения. |
| com/example/programmingas3/runtimeassetexplorer/AnimatingBox.as | Класс библиотечного символа в виде вращающейся коробки. |
| com/example/programmingas3/runtimeassetexplorer/AnimatingStar.as | Класс библиотечного символа в виде вращающейся звезды. |

## Реализация динамического интерфейса библиотеки

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы обозреватель правильно взаимодействовал с SWF-библиотекой, необходимо создать формальную структуру библиотек динамических ресурсов. Это достигается путем создания интерфейса, подобного классу в том, что он представляет собой схему методов, разграничивающих желаемую структуру, и отличного в том, что он не включает тел методов. Этот интерфейс обеспечивает связь между библиотекой динамических ресурсов и обозревателем. Каждый SWF-файл с динамическими ресурсами, загруженный в обозреватель, внедряет свой интерфейс. Дополнительные сведения об интерфейсах и их практическом применении см. в разделе «Интерфейсы» в руководстве *Изучение ActionScript 3.0*.

Интерфейс RuntimeLibrary будет очень простым. Нам просто требуется функция, которая может предоставить обозревателю массив путей к классам для символов, предназначенных для экспорта и доступных в библиотеке динамических ресурсов. С этой целью в интерфейсе предусмотрен только один метод: `getAssets()`.

```
package com.example.programmingas3.runtimeassetexplorer
{
    public interface RuntimeLibrary
    {
        function getAssets():Array;
    }
}
```

## Создание SWF-файла библиотеки

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Благодаря определению интерфейса RuntimeLibrary мы получили возможность создать несколько SWF-файлов для библиотеки ресурсов, которые могут загружаться в другой SWF-файл. Создание отдельной SWF-библиотеки ресурсов включает четыре этапа:

- Создание класса для SWF-файла библиотеки ресурсов
- Создание классов для отдельных ресурсов, содержащихся в библиотеке
- Создание фактических графических ресурсов
- Связывание графических элементов с классами и публикация SWF-библиотеки

### Создание класса для реализации интерфейса RuntimeLibrary

Далее мы создадим класс GeometricAssets, который реализует интерфейс RuntimeLibrary. Это будет класс документа FLA. Код для этого класса очень похож на интерфейс RuntimeLibrary. Отличие между ними заключается в том, что в определении класса метод `getAssets()` включает тело метода.

```
package
{
    import flash.display.Sprite;
    import com.example.programmingas3.runtimeassetexplorer.RuntimeLibrary;

    public class GeometricAssets extends Sprite implements RuntimeLibrary
    {
        public function GeometricAssets() {

        }
        public function getAssets():Array {
            return [ "com.example.programmingas3.runtimeassetexplorer.AnimatingBox",
                     "com.example.programmingas3.runtimeassetexplorer.AnimatingStar" ];
        }
    }
}
```

Если бы потребовалось создать вторую библиотеку динамических ресурсов, мы бы создали еще один FLA-файл на базе другого класса (например, AnimationAssets), который обеспечивает собственную реализацию метода `getAssets()`.

### Создание классов для каждого ресурса MovieClip

В данном примере мы просто расширим класс MovieClip, не добавляя функций в пользовательские ресурсы. Следующий код для AnimatingStar аналогичен коду для AnimatingBox.

```
package com.example.programmingas3.runtimeassetexplorer
{
    import flash.display.MovieClip;

    public class AnimatingStar extends MovieClip
    {
        public function AnimatingStar() {
        }
    }
}
```

### Публикация библиотеки

Теперь мы подключим ресурсы на базе MovieClip к новому классу путем создания нового FLA-файла и ввода GeometricAssets в поле «Класс документа» в инспекторе свойств. Для данного примера мы создадим две очень простые фигуры, использующие анимацию с временной шкалой (вращение по часовой стрелке на протяжении 360 кадров). Для символов `animatingBox` и `animatingStar` установлен флажок «Экспорт для ActionScript», а в поле «Класс» указаны соответствующие пути к классам, заданные при реализации `getAssets()`. Базовый класс по умолчанию `flash.display.MovieClip` остается без изменений, так как нам требуется создать подклассы на базе стандартных методов MovieClip.

Настроив параметры экспорта символа, опубликуйте FLA-файл. Первая библиотека динамических ресурсов готова. Этот SWF-файл можно загрузить в другой SWF-файл AVM2, и символы AnimatingBox и AnimatingStar будут доступны для нового SWF-файла.

## Загрузка библиотеки в другой SWF-файл

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

На последнем этапе необходимо организовать пользовательский интерфейс для обозревателя ресурсов. В данном примере путь к библиотеке динамических ресурсов жестко закодирован как переменная с именем `ASSETS_PATH`. Также можно использовать класс FileReference, например, чтобы создать интерфейс, позволяющий просматривать определенный SWF-файл на жестком диске.

После успешной загрузки библиотеки динамических ресурсов проигрыватель Flash Player вызывает метод `runtimeAssetsLoadComplete()`.

```
private function runtimeAssetsLoadComplete(event:Event):void
{
    var rl:* = event.target.content;
    var assetList:Array = rl.getAssets();
    populateDropdown(assetList);
    stage.frameRate = 60;
}
```

В этом методе переменная rl представляет загруженный SWF-файл. Код вызывает метод `getAssets()` загруженного SWF-файла, чтобы получить список доступных ресурсов, и заполняет этим списком компонент ComboBox, вызвав метод `populateDropDown()`. В свою очередь, этот метод сохраняет полный путь к классам каждого ресурса. Нажатие кнопки «Добавить» в пользовательском интерфейсе запускает метод `addAsset()`.

```
private function addAsset():void
{
    var className:String = assetNameCbo.selectedItem.data;
    var AssetClass:Class = getDefinitionByName(className) as Class;
    var mc:MovieClip = new AssetClass();
    ...
}
```

Этот метод получает путь к классам текущего ресурса, выбранного в компоненте ComboBox (`assetNameCbo.selectedItem.data`), и использует функцию `getDefinitionByName()` (из пакета flash.utils) для получения фактической ссылки на класс ресурса, чтобы создать его новый экземпляр.

# Глава 17. Работа с анимацией движения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий, требуется Flash CS3 или более поздних версий**

В разделе «Анимация объектов» на странице 208 описывается внедрение сценариев анимации в код ActionScript.

В этой главе описывается другой прием создания анимации: анимация движения. Этот прием позволяет создавать движение путем его интерактивной настройки в документе с помощью Adobe® Flash® Professional. Затем это движение можно использовать в динамической анимации на базе ActionScript во время выполнения.

Flash Professional автоматически создает код ActionScript, который выполняет анимацию движения и делает ее доступной для копирования и повторного использования.

Для создания анимации движения необходимо иметь лицензию на Flash Professional.

### Дополнительные разделы справки

Пакет fl.motion

## Основные сведения об анимации движения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий, требуется Flash CS3 или более поздних версий**

Анимация движения представляет собой простой способ создания анимации.

Анимация движения изменяет свойства экранного объекта, например позицию или поворот, для каждого кадра. Анимация движения также может менять внешний вид экранного объекта по ходу движения, применяя различные фильтры и другие свойства. Анимацию движения можно создать в интерактивном режиме с помощью инструмента Flash Professional, который генерирует код ActionScript для анимации движения. В инструменте разработки Flash с помощью команды «Копировать движение как код ActionScript 3.0» скопируйте код ActionScript для создания анимации движения. Затем этот код ActionScript можно повторно использовать, чтобы создавать движение в собственной динамической анимации во время выполнения.

Сведения о создании анимации движения см. в разделе «Анимация движения» руководства *Использование Flash Professional*.

### Важные понятия и термины

Ниже приведен важный термин, используемый при работе.

**Анимация движения** Структура, которая генерирует промежуточные кадры экранного объекта в разных состояниях в разные моменты; создает впечатление, что первое состояние плавно переходит во второе.

Служит для перемещения экранного объекта по рабочей области, а также для увеличения или уменьшения его размера, поворота, затемнения или изменения его цвета на протяжении определенного времени.

# Копирование сценариев анимации движения в Flash

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий, требуется Flash CS3 или более поздних версий**

Анимация движения генерирует промежуточные кадры, которые показывают экранный объект в разных состояниях между двумя разными кадрами на временной шкале. Создается впечатление, что изображение в первом кадре плавно переходит в изображение на втором кадре. В анимации движения изменение внешнего вида, как правило, включает изменение положения экранного объекта, в результате чего создается движение. Помимо изменения позиции экранного объекта, анимация движения также может поворачивать, наклонять объект, изменять его размеры или применять к нему фильтры.

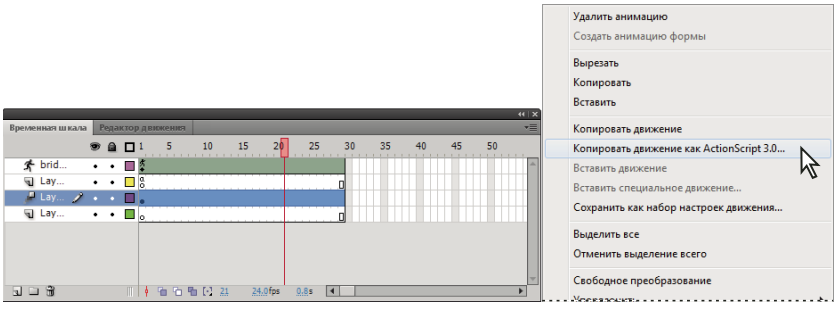
Создайте анимацию движения в инструменте Flash, перемещая экранный объект между ключевыми кадрами вдоль временной шкалы. Flash автоматически генерирует код ActionScript, описывающий анимацию движения, который можно скопировать и сохранить в файл. Сведения о создании анимации движения см. в разделе «Анимация движения» руководства *Использование Flash Professional*.

Команду «Копировать движение как код ActionScript 3.0» в инструменте Flash можно вызвать двумя способами. Во-первых, ее можно вызвать из контекстного меню анимации движения в рабочей области.

1 Выберите анимацию движения в рабочей области.

2 Щелкните по ней правой кнопкой мыши (Windows) или удерживая клавишу «Control» (Macintosh).

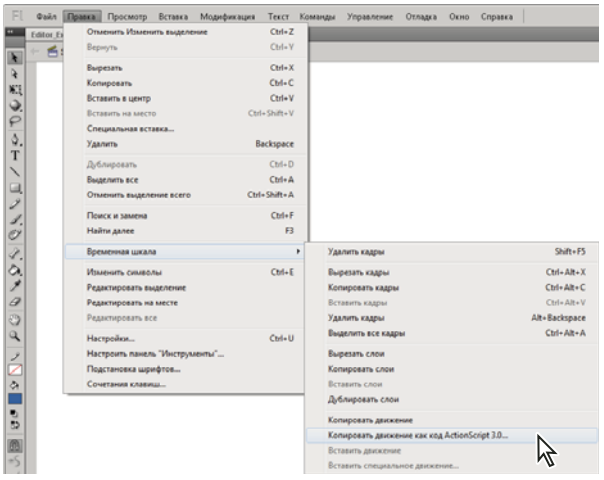
3 Выберите «Копировать движение как код ActionScript 3.0 . . .»




Во-вторых, эту команду можно выбрать непосредственно в меню «Правка» инструмента Flash.

1 Выберите анимацию движения в рабочей области.

**2** Выберите «Правка» > «Временная шкала» > «Копировать движение как код ActionScript 3.0».




Скопированный сценарий вставьте в файл и сохраните.

Создав анимацию движения, а затем скопировав и сохранив сценарий, ее можно повторно использовать как есть или с изменениями в собственной динамической анимации на базе ActionScript.

# Встраивание сценариев анимации движения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий, требуется Flash CS3 или более поздних версий**

Заголовок в коде ActionScript, скопированном из инструмента Flash, включает все модули, необходимые для поддержки анимации движения.

## Классы анимации движения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий, требуется Flash CS3 или более поздних версий**

Основными для анимации движения являются классы AnimatorFactory, MotionBase и Motion из пакета `fl.motion`. Могут понадобиться и дополнительные классы в зависимости от свойств, которыми манипулирует анимация движения. Например, если анимация движения преобразует или поворачивает экранный объект, необходимо импортировать соответствующие классы из пакета `flash.geom`. Если анимация применяет фильтры, необходимо импортировать классы `flash.filter`. В ActionScript анимация движения является экземпляром класса Motion. Класс Motion хранит последовательность анимации ключевых кадров, которая может применяться к визуальным объектам. Данные анимации включают положение, масштаб, наклон, цвет, фильтры и плавность.

Следующий код ActionScript скопирован из движения, созданного в Flash для анимации экранного объекта с именем `Symbol1_2`. Код объявляет переменную для объекта MotionBase с именем `__motion_Symbol1_2`. Класс MotionBase является родителем класса Motion.

```
var __motion_Symbol1_2:MotionBase;
```

Затем сценарий создает объект Motion.

```
__motion_Symbol1_2 = new Motion();
```

## Имена объектов Motion

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий, требуется Flash CS3 или более поздних версий**

В предыдущем примере инструмент Flash автоматически создал имя `__motion_Symbol1_2` для объекта Motion. Он прибавил префикс `__motion_` к имени экранного объекта. Таким образом, автоматически генерируемое имя создается на базе имени экземпляра целевого объекта для анимации движения в Flash. Свойство `duration` объекта Motion указывает общее количество кадров в анимации движения.

```
__motion_Symbol1_2.duration = 200;
```

По умолчанию Flash автоматически присваивает имя экземпляру экранного объекта, движение которого копируется, если у него еще нет имени экземпляра.

При повторном использовании кода ActionScript, созданного при помощи Flash с вашей собственной анимацией, можно принять имя, автоматически сгенерированное Flash для анимации, или заменить его другим именем. Если требуется изменить имя анимации движения, проследите, чтобы были обновлены все вхождения в сценарии.

Также в инструменте Flash можно присвоить собственное имя целевому объекту анимации движения. Затем создайте анимацию движения и скопируйте сценарий. Независимо от выбранного способа наименования проследите, чтобы каждый объект Motion в коде ActionScript имел уникальное имя.

# Описание анимации

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий, требуется Flash CS3 или более поздних версий**

Метод `addPropertyArray()` класса MotionBase добавляет массив значений для описания каждого анимируемого свойства.

Потенциально массив может содержать один элемент для каждого ключевого кадра анимации движения. Часто число элементов в некоторых из таких массивов меньше общего количества ключевых кадров в анимации движения. Такое происходит, когда последнее значение массива не изменяется для оставшихся кадров.

Если длина аргумента массива больше свойства `duration` объекта Motion, метод `addPropertyArray()` соответствующим образом корректирует значение свойства `duration`. Он не прибавляет ключевых кадров для свойств, добавленных ранее. Вновь добавленные ключевые кадры сохраняются для дополнительных кадров анимации.

Свойства `x` и `y` объекта Motion описывают изменяющееся положение объекта в процессе выполнения анимации. При изменении позиции экранного объекта эти координаты меняются практически в каждом ключевом кадре. С помощью метода `addPropertyArray()` можно добавить дополнительные свойства движения. Например, добавьте значения `scaleX` и `scaleY`, если изменяется размер анимируемого объекта. Добавьте значения `scewX` и `skewY`, если объект наклоняется. Добавьте свойство `rotationConcat`, если объект поворачивается.

Используйте метод `addPropertyArray()` для определения следующих свойств анимации:

| | |
|---|---|
| `x` | позиция по вертикали точки преобразования объекта в координатной системе его родителя |
| `y` | позиция по горизонтали точки преобразования объекта в координатной системе его родителя |
| `z` | позиция по оси Z (глубина) точки преобразования объекта в координатной системе его родителя |
| `scaleX` | горизонтальный масштаб объекта в результате трансформации (%) |
| `scaleY` | вертикальный масштаб объекта в результате трансформации (%) |
| `skewX` | угол наклона целевого объекта по горизонтали относительно точки преобразования (в градусах) |
| `skewY` | угол наклона целевого объекта по вертикали относительно точки преобразования (в градусах) |
| `rotationX` | поворот объекта по оси X относительно исходной ориентации |
| `rotationY` | поворот объекта по оси Y относительно исходной ориентации |
| `rotationConcat` | значения поворота (по оси Z) двигающегося объекта относительно предыдущей ориентации и точки преобразования |
| `useRotationConcat` | если задано это свойство, целевой объект поворачивается, когда метод `addPropertyArray()` передает данные для движения |
| `blendMode` | значение класса BlendMode, определяющее смешивание цветов объекта с находящейся под ним графикой |
| `matrix3D` | свойство matrix3D, если оно существует для данного ключевого кадра; используется для трехмерной анимации; если оно применяется, все предыдущие свойства преобразования игнорируются |
| `rotationZ` | поворот объекта по оси Z в градусах относительно исходной ориентации и трехмерного родительского контейнера, используется для трехмерной анимации вместо свойства rotationConcat. |

Свойства, добавляемые в автоматически генерируемый сценарий, зависят от того, какие свойства были назначены анимации движения в инструменте разработки Flash. Некоторые из этих свойств можно добавлять, удалять или изменять при настройке собственной версии сценария.

Следующий код задает значения для следующих свойств анимации движения с именем `__motion_Wheel`. В этом случае анимируемый экранный объект не изменяет положение, а вращается на месте на протяжении 29 кадров анимации. Поворот определяют многочисленные значения, присвоенные массиву `rotationConcat`. Значения других свойств этой анимации движения не изменяются.

```
__motion_Wheel = new Motion();
__motion_Wheel.duration = 29;
__motion_Wheel.addPropertyArray("x", [0]);
__motion_Wheel.addPropertyArray("y", [0]);
__motion_Wheel.addPropertyArray("scaleX", [1.00]);
__motion_Wheel.addPropertyArray("scaleY", [1.00]);
__motion_Wheel.addPropertyArray("skewX", [0]);
__motion_Wheel.addPropertyArray("skewY", [0]);
__motion_Wheel.addPropertyArray("rotationConcat",
    [
        0,-13.2143,-26.4285,-39.6428,-52.8571,-66.0714,-79.2857,-92.4999,-105.714,
        -118.929,-132.143,-145.357,-158.571,-171.786,-185,-198.214,-211.429,-224.643,
        -237.857,-251.071,-264.286,-277.5,-290.714,-303.929,-317.143,-330.357,
        -343.571,-356.786,-370
    ]
);
__motion_Wheel.addPropertyArray("blendMode", ["normal"]);
```

В следующем примере экранный объект с именем `Leaf_1` перемещается по рабочей области. Массивы его свойств `x` и `y` содержат разные значения для каждого из 100 кадров анимации. Кроме того, по мере перемещения по рабочей области объект поворачивается вокруг оси `Z`. Поворот определяют многочисленные элементы массива свойства `rotationZ`.

```
__motion_Leaf_1 = new MotionBase();
__motion_Leaf_1.duration = 100;
__motion_Symbol1_4.addPropertyArray("y",
    [
        0,5.91999,11.84,17.76,23.68,29.6,35.52,41.44,47.36,53.28,59.2,65.12,71.04,
        76.96,82.88,88.8,94.72,100.64,106.56,112.48,118.4,124.32,130.24,136.16,142.08,
        148,150.455,152.909,155.364,157.818,160.273,162.727,165.182,167.636,170.091,
        172.545,175,177.455,179.909,182.364,184.818,187.273,189.727,192.182,194.636,
        197.091,199.545,202,207.433,212.865,218.298,223.73,229.163,234.596,240.028,
        245.461,250.893,256.326,261.759,267.191,272.624,278.057,283.489,
        288.922,294.354,299.787,305.22,310.652,316.085,321.517,326.95,330.475,334,
        337.525,341.05,344.575,348.1,351.625,355.15,358.675,362.2,365.725,369.25,
        372.775,376.3,379.825,383.35,386.875,390.4,393.925,397.45,400.975,404.5,
        407.5,410.5,413.5,416.5,419.5,422.5,425.5
    ]
);
__motion_Symbol1_4.addPropertyArray("scaleX", [1.00]);
__motion_Symbol1_4.addPropertyArray("scaleY", [1.00]);
__motion_Symbol1_4.addPropertyArray("skewX", [0]);
__motion_Symbol1_4.addPropertyArray("skewY", [0]);
__motion_Symbol1_4.addPropertyArray("z",
    [
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ]
);
```

```
__motion_Symbol1_4.addPropertyArray("rotationX", [64.0361]);
__motion_Symbol1_4.addPropertyArray("rotationY", [41.9578]);
__motion_Symbol1_4.addPropertyArray("rotationZ",
    [
        -18.0336,-17.5536,-17.0736,-16.5936,-16.1136,-15.6336,-15.1536,-14.6736,
        -14.1936,-13.7136,-13.2336,-12.7536,-12.2736,-11.7936,-11.3136,-10.8336,
        -10.3536,-9.8736,-9.3936,-8.9136,-8.4336,-7.9536,-7.4736,-6.9936,-6.5136,
        -6.0336,-7.21542,-8.39723,-9.57905,-10.7609,-11.9427,-13.1245,-14.3063,
        -15.4881,-16.67,-17.8518,-19.0336,-20.2154,-21.3972,-22.5791,-23.7609,
        -24.9427,-26.1245,-27.3063,-28.4881,-29.67,-30.8518,-32.0336,-31.0771,
        -30.1206,-29.164,-28.2075,-27.251,-26.2945,-25.338,-24.3814,-23.4249,
        -22.4684,-21.5119,-20.5553,-19.5988,-18.6423,-17.6858,-16.7293,-15.7727
        -14.8162,-13.8597,-12.9032,-11.9466,-10.9901,-10.0336,-10.9427,-11.8518,
        -12.7609,-13.67,-14.5791,-15.4881,-16.3972,-17.3063,-18.2154,-19.1245,
        -20.0336,-20.9427,-21.8518,-22.7609,-23.67,-24.5791,-25.4881,-26.3972,
        -27.3063,-28.2154,-29.1245,-30.0336,-28.3193,-26.605,-24.8907,-23.1765,
        -21.4622,-19.7479,-18.0336
    ]
);
__motion_Symbol1_4.addPropertyArray("blendMode", ["normal"]);
```

# Добавление фильтров

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий, требуется Flash CS3 или более поздних версий**

Если целевой объект анимации движения содержит фильтры, они добавляются с помощью методов `initFilters()` и `addFilterPropertyArray()` класса Motion.

## Инициализация массива фильтров

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий, требуется Flash CS3 или более поздних версий**

Метод `initFilters()` инициализирует фильтры. Его первый аргумент представляет собой массив полностью определенных имен классов всех фильтров, примененных к экранному объекту. Этот массив с именами фильтров генерируется на базе списка фильтров для данной анимации движения в инструменте Flash. Сделав копию сценария, в этом массиве можно удалять или добавлять любые фильтры из пакета `flash.filters`. Следующий код инициализирует список фильтров для целевого экранного объекта. Он применяет фильтры `DropShadowFilter`, `GlowFilter` и `BevelFilter`, а затем копирует список в каждый ключевой кадр объекта Motion.

```
__motion_Box.initFilters(["flash.filters.DropShadowFilter", "flash.filters.GlowFilter",
"flash.filters.BevelFilter"], [0, 0, 0]);
```

## Добавление фильтров

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий, требуется Flash CS3 или более поздних версий**

Метод `addFilterPropertyArray()` описывает свойства инициализированного фильтра, используя следующие аргументы.

1. Первый аргумент определяет фильтр по индексу. Индекс указывает позицию имени фильтра в массиве с именами классов фильтров, переданном в предыдущем вызове метода `initFilters()`.
2. Вторым аргументом является свойство фильтра, которое сохраняется для этого фильтра в каждом ключевом кадре.
3. В качестве третьего аргумента передается значение указанного свойства фильтра.

С учетом предыдущего вызова `initFilters()` последующие вызовы метода `addFilterPropertyArray()` задают значение 5 свойствам `blurX` и `blurY` объекта `DropShadowFilter`. Объект `DropShadowFilter` является первым элементом (с индексом 0) в инициализированном массиве фильтров.

```
__motion_Box.addFilterPropertyArray(0, "blurX", [5]);
__motion_Box.addFilterPropertyArray(0, "blurY", [5]);
```

Следующие три вызова задают значения для свойств качества, альфа-прозрачности и цвета объекта `GlowFilter`, второго элемента (с индексом 1) в инициализированном массиве фильтров.

```
__motion_Box.addFilterPropertyArray(1, "quality", [BitmapFilterQuality.LOW]);
__motion_Box.addFilterPropertyArray(1, "alpha", [1.00]);
__motion_Box.addFilterPropertyArray(1, "color", [0xff0000]);
```

Следующие четыре вызова задают значения свойствам `shadowAlpha`, `shadowColor`, `highlightAlpha` и `highlightColor` объекта `BevelFilter`, третьего элемента (с индексом 2) в инициализированном массиве фильтров.

```
__motion_Box.addFilterPropertyArray(2, "shadowAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "shadowColor", [0x000000]);
__motion_Box.addFilterPropertyArray(2, "highlightAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "highlightColor", [0xffffff]);
```

## Настройка цвета с помощью ColorMatrixFilter

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий, требуется Flash CS3 или более поздних версий**

После инициализации `ColorMatrixFilter` можно задать соответствующие свойства `AdjustColor`, чтобы настроить яркость, контрастность, насыщенность и цветовой тон анимируемого экранного объекта. Как правило, когда во Flash создается анимационный переход, применяется фильтр `AdjustColor`; вы можете выполнить лишь его тонкую настройку в своей собственной копии кода ActionScript. В следующем примере по мере движения преобразуется цветовой тон и насыщенность экранного объекта.

```
__motion_Leaf_1.initFilters(["flash.filters.ColorMatrix"], [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorBrightness", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorContrast", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorSaturation",
    [
        0,-0.589039,1.17808,-1.76712,-2.35616,-2.9452,-3.53424,-4.12328,
        -4.71232,-5.30136,-5.89041, 6.47945,-7.06849,-7.65753,-8.24657,
        -8.83561,-9.42465,-10.0137,-10.6027,-11.1918,11.7808,-12.3699,
        -12.9589,-13.5479,-14.137,-14.726,-15.3151,-15.9041,-16.4931,
        17.0822,-17.6712,-18.2603,-18.8493,-19.4383,-20.0274,-20.6164,
        -21.2055,-21.7945,22.3836,-22.9726,-23.5616,-24.1507,-24.7397,
        -25.3288,-25.9178,-26.5068,-27.0959,27.6849,-28.274,-28.863,-29.452,
        -30.0411,-30.6301,-31.2192,-31.8082,-32.3973,32.9863,-33.5753,
        -34.1644,-34.7534,-35.3425,-35.9315,-36.5205,-37.1096,-37.6986,
        38.2877,-38.8767,-39.4657,-40.0548,-40.6438,-41.2329,-41.8219,
        -42.411,-43
    ],
    -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorHue",
    [
        0,0.677418,1.35484,2.03226,2.70967,3.38709,4.06451,4.74193,5.41935,
        6.09677,6.77419,7.45161,8.12903,8.80645,9.48387,10.1613,10.8387,11.5161,
        12.1935,12.871,13.5484,14.2258,14.9032,15.5806,16.2581,16.9355,17.6129,
        18.2903,18.9677,19.6452,20.3226,21,22.4286,23.8571,25.2857,26.7143,28.1429,
        29.5714,31,32.4286,33.8571,35.2857,36.7143,38.1429,39.5714,41,42.4286,43.8571,
        45.2857,46.7143,48.1429,49.5714,51,54,57,60,63,66,69,72,75,78,81,84,87,
        90,93,96,99,102,105,108,111,114
    ],
    -1, -1);
```

# Связывание анимации движения с ее экранными объектами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий, требуется Flash CS3 или более поздних версий**

Последняя задача заключается в связывании анимации движения с одним или несколькими экранными объектами, которыми она манипулирует.

Класс AnimatorFactory управляет связью между анимацией движения и ее целевыми экранными объектами. В качестве аргумента конструктора AnimatorFactory передается объект Motion.

```
var __animFactory_Wheel:AnimatorFactory = new AnimatorFactory(__motion_Wheel);
```

Используйте метод `addTarget()` класса AnimatorFactory, чтобы связать целевой экранный объект с его анимацией движения. Код ActionScript, скопированный из Flash исключает строку `addTarget()` и не указывает имя экземпляра.

```
// __animFactory_Wheel.addTarget(<instance name goes here>, 0);
```

В копии кода нужно указать экранный объект, который требуется связать с анимацией движения. В следующем примере в качестве целей указываются объекты `greenWheel` и `redWheel`.

```
__animFactory_Wheel.AnimatorFactory.addTarget(greenWheel, 0);
__animFactory_Wheel.AnimationFactory.addTarget(redWheel, 0);
```

С одной анимацией движения можно связать несколько экранных объектов, вызывая для каждого из них метод `addTarget()`.

# Глава 18. Работа с обратной кинематикой

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий, требуется Flash CS4 или более поздних версий**

Обратная кинематика (ОК) — это отличная техника для создания реалистичного движения.

ОК позволяет создавать скоординированные движения в пределах цепи связанных между собой частей, образующих каркас ОК, так что эти части правдоподобно двигаются вместе. Части каркаса называются костями и соединениями. На основе конечной точки каркаса ОК вычисляет углы для соединений, необходимые для достижения этой конечной точки.

Вычислять эти углы вручную довольно проблематично. Прелесть этой функции заключается в том, что каркасы можно создавать в интерактивном режиме с помощью Adobe® Flash® Professional. Затем они анимируются с помощью ActionScript. Модуль ОК, включенный в среду разработки Flash Professional, выполняет вычисления, чтобы описать движение каркаса. С помощью кода ActionScript движение можно ограничить определенными параметрами.

В новой версии модуля обратной кинематики для Flash Professional CS5 присутствует поддержка костной пружины, обычно являющаяся атрибутом профессиональных приложений для создания анимации. В совокупности с механизмом физического моделирования эта функция позволяет добиться максимально реалистичного движения. Результат использования этой функции виден как во время выполнения, так и во время разработки.

Для создания каркасов обратной кинематики необходимо иметь лицензию на Flash Professional.

### Дополнительные разделы справки

Пакет fl.ik

## Основы обратной кинематики

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий, требуется Flash CS4 или более поздних версий**

Обратная кинематика (ОК) позволяет создавать реалистичную анимацию путем соединения частей так, что они правдоподобно передвигаются относительно друг друга.

Например, с помощью ОК можно переместить ногу в определенное положение путем воспроизведения движений составов ноги, необходимых для перехода в нужную позу. ОК использует систему костей, связанных вместе в структуре, которая называется каркасом ОК. С помощью пакета `fl.ik` можно создавать анимацию, напоминающую естественные движения. Он позволяет согласованно анимировать несколько каркасов ОК, не требуя знаний по физике, на основе которой построены алгоритмы ОК.

С помощью Flash Professional создайте каркас ОК с вспомогательными костями и соединениями. Затем вы сможете обращаться к классам ОК, чтобы анимировать их во время выполнения.

Подробные инструкции по созданию каркаса ОК см. в разделе «Использование обратной кинематики» в руководстве *Использование Flash Professional*.

**Важные понятия и термины**

Ниже приводится список важных терминов, относящихся к этой функции.

**Арматура** Кинематическая цепочка, состоящая из костей и соединений, которая используется в компьютерной анимации для симуляции реалистичного движения.

**Кость** Несгибаемый сегмент каркаса, аналогичный кости в скелете животного.

**Обратная кинематика (ОК)** Процесс определения параметров объекта с гибкими соединениями, называемого кинематической цепочкой или каркасом.

**Соединение** Место, в котором соединяются две кости, которое обеспечивает движение костей; аналогично суставу животного.

**Механизм физического моделирования** Пакет алгоритмов, связанных с физикой, используемый для обеспечения правдоподобных движений при анимации.

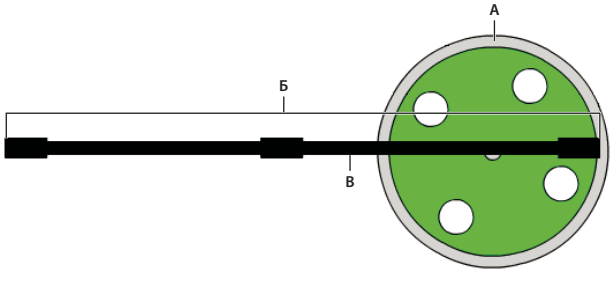
**Пружина** Качество кости, которая перемещается и реагирует на движение родительской кости, а затем это движение со временем затухает.

# Обзор анимации каркасов ОК

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий, требуется Flash CS4 или более поздних версий**

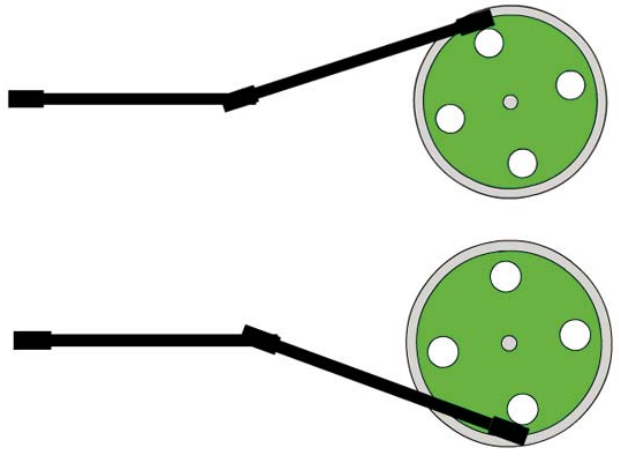
Создав каркас ОК в Flash Professional, используйте классы `fl.ik` для ограничения его движений, отслеживания событий и анимации во время выполнения.

На следующем рисунке показан фрагмент ролика с именем `Wheel`. Валом является экземпляр IKArmature с именем `Axle`. Класс IKMover перемещает каркас синхронно с вращением колеса. Объект IKBone, `ikBone2`, в каркасе присоединен к колесу хвостовым соединением.




***А.** Wheel **Б.** Axle **В.** ikBone2*

Во время выполнения приложения колесо вращается в зависимости от анимационного перехода `__motion_Wheel` (см. раздел «Описание анимации» на странице 365). Объект IKMover инициирует и контролирует движение вала. На следующем рисунке приводится два снимка кадров каркаса вала, присоединенного к колесу, на разных этапах вращения.



Во время выполнения следующий код ActionScript:

- получает информацию о каркасе и его компонентах;
- создает экземпляр класса IKMover;
- перемещает вал в соответствии с вращением колеса.

```
import fl.ik.*

var tree:IKArmature = IKManager.getArmatureByName("Axle");
var bone:IKBone = tree.getBoneByName("ikBone2");
var endEffector:IKJoint = bone.tailJoint;
var pos:Point = endEffector.position;

var ik:IKMover = new IKMover(endEffector, pos);
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;

Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90, 0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

Для передвижения вала используются следующие классы ОК.

- IKArmature: описывает каркас, имеет структуру дерева, состоящую из костей и соединений; необходимо создать с помощью Flash. Professional
- IKManager: класс-контейнер для всех каркасов ОК в данном документе; его необходимо создать с помощью Flash Professional.
- IKBone: сегмент каркаса ОК.
- IKJoint: соединение двух костей ОК
- IKMover: инициирует и контролирует движение ОК каркасов.

Полные и подробные описания этих классов см. в пакете ik.

# Получение сведений о каркасе ОК

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий, требуется Flash CS4 или более поздних версий**

Во-первых, необходимо объявить переменные для каркаса, костей и соединений, которые образуют части, которые должны передвигаться.

В следующем коде используется метод `getArmatureByName()` класса IKManager, чтобы присвоить значение каркаса Axle переменной IKArmature с именем `tree`. Каркас Axle был предварительно создан с помощью Flash Professional.

```
var tree:IKArmature = IKManager.getArmatureByName("Axle");
```

Подобным образом следующий код использует метод `getBoneByName()` класса IKArmature, чтобы присвоить переменной IKBone значение кости `ikBone2`.

```
var bone:IKBone = tree.getBoneByName("ikBone2");
```

Хвостовое соединение кости `ikBone2` является частью каркаса, присоединенного к вращающемуся колесу.

Следующая строка объявляет переменную `endEffector` и назначает ее свойству `tailjoint` кости `ikBone2`.

```
var endEffector:IKJoint = home.tailjoint;
```

Переменная `pos` является точкой, в которой сохранена текущая позиция соединения `endEffector`.

```
var pos:Point = endEffector.position;
```

В данном примере `pos` — это позиция соединения в конце вала, где он прикрепляется к колесу. Исходное значение этой переменной получается из свойства `position` объекта IKJoint.

# Создание экземпляра IK Mover и ограничение его движения

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий, требуется Flash CS4 или более поздних версий**

Экземпляр класса IKMover передвигает вал.

Следующая строка кода создает объект IKMover с именем `ik`, передавая его конструктору элемент, который должен двигаться, и начальную точку движения.

```
var ik:IKMover = new IKMover(endEffector, pos);
```

Свойства класса IKMover позволяют ограничить движение каркаса. Движение можно ограничить по расстоянию, количество повторов и времени движения.

Для установки этих ограничений служат следующие пары свойств. Пара состоит из логического значения Boolean, которое указывает, ограничено ли движение, и целого числа, определяющего степень ограничения.

| Свойство Boolean | Свойство Integer | Ограничение |
|---|---|---|
| `limitByDistance:Boolean` | `distanceLimit:int` | Задает максимальное расстояние в пикселах, на которое перемещается модуль OK в пределах каждого повтора. |
| `limitByIteration:Boolean` | `iterationLimit:int` | Задает максимальное число повторений, выполняемых модулем обратной кинематики для каждого движения. |
| `limitByTime:Boolean` | `timeLimit:int` | Задает максимальное время в миллисекундах, выделенное модулю OK для выполнения движения. |

По умолчанию все значения типа Boolean равны `false`, поэтому движение не ограничено, если вы в явном виде не задали значение типа Boolean равным `true`. Чтобы ввести ограничение, задайте соответствующему свойству значение `true`, после чего укажите значение связанного с ним целочисленного свойства. Если задать для ограничения целочисленное значение, не настроив соответствующего свойства Boolean, то ограничение игнорируется. В этом случае модуль OK продолжает передвигать объект до другого ограничения или до достижения целевой позиции объекта IKMover.

В следующем примере максимальному расстоянию движения каркаса задано значение 0,1 пиксела для каждого повтора. Максимальное количество повторов для каждого движения равно десяти.

```
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;
```

# Перемещение каркаса ОК

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий, требуется Flash CS4 или более поздних версий**

Объект IKMover передвигает вал внутри прослушивателя событий для колеса. При отправке каждого события enterFrame колеса вычисляется новая целевая позиция каркаса. С помощью метода `moveTo()` объект IKMover перемещает хвостовое соединение в целевую позицию или до ограничения, заданного свойствами `limitByDistance`, `limitByIteration` и `limitByTime`.

```
Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90,0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

# Использование пружин

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий, требуется Flash CS5 или более поздних версий**

В Flash Professional CS5 модуль обратной кинематики поддерживает костную пружину. Костную пружину можно настроить во время разработки, а ее атрибуты можно добавлять или изменять прямо во время выполнения. Пружина Spring — это свойство кости и ее суставов. Она имеет два атрибута: `IKJoint.springStrength`, который задает усилие пружины, и `IKJoint.springDamping`, который складывает сопротивление и усилие, что влияет на скорость демпфирования колебаний пружины.

Усилие пружины выражается в процентах от 0 (максимально жесткая) до 100 (максимально мягкая, управляемая физикой). Кости с пружиной реагируют на движения своих суставов. Если не применяется других типов поступательного движения (по оси X или Y, вращение), настройки пружины не действуют.

Демпфирование пружины выражается в процентах от 0 (нет сопротивления) до 100 (максимальное сопротивление). Демпфирование влияет на время, которое проходит после начала движения кости и до ее возвращения в состояние покоя.

Чтобы узнать, включены ли пружины для объекта IKArmature, проверьте свойство `IKArmature.springsEnabled` . Остальные свойства и методы пружины принадлежат отдельным объектам IKJoint. Сустав может поддерживать угловой поворот и поступательное движение вдоль осей X и Y. Задать расположение угла вращательной пружины поворачиваемого сустава можно с помощью свойства `IKJoint.setSpringAngle`, а расположение пружины поступательного сустава — с помощью свойства `IKJoint.setSpringPt`.

В данном примере производится выбор кости по имени и идентификация ее свойства tailJoint. Код проверяет родительский каркас, чтобы узнать, поддерживают ли он пружины, а затем задает свойства пружины для сустава.

```
var arm:IKArmature = IKManager.getArmatureAt(0);
var bone:IKBone = arm.getBoneByName("c");
var joint:IKJoint = bone.tailJoint;
if (arm.springsEnabled) {
    joint.springStrength = 50; //medium spring strength
    joint.springDamping = 10; //light damping resistance
    if (joint.hasSpringAngle) {
        joint.setSpringAngle(30); //set angle for rotational spring
    }
}
```

# Использование событий ОК

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий, требуется Flash CS4 или более поздних версий**

Класс IKEvent позволяет создать объект события, содержащий сведения о событиях ОК. Информация объекта IKEvent описывает движение, которое прекращено в результате достижения указанного ограничения по времени, расстоянию или числу повторов.

В следующем коде регистрируется прослушиватель и обработчик событий для отслеживания событий ограничения по времени. Этот обработчик событий сообщает сведения о времени, расстоянии, количестве повторов и свойствах соединения для события, которое отправляется при превышении ограничения по времени, установленного для объекта IKMover.

```
var ikmover:IKMover = new IKMover(endjoint, pos);
ikMover.limitByTime = true;
ikMover.timeLimit = 1000;

ikmover.addEventListener(IKEvent.TIME_LIMIT, timeLimitFunction);

function timeLimitFunction(evt:IKEvent):void
{
    trace("timeLimit hit");
    trace("time is " + evt.time);
    trace("distance is " + evt.distance);
    trace("iterationCount is " + evt.iterationCount);
    trace("IKJoint is " + evt.joint.name);
}
```

# Глава 19. Работа в трех измерениях (3D)

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Среды выполнения Flash Player и AIR поддерживают 3D-графику двумя способами. В списке отображения Flash можно использовать трехмерные экранные объекты. Этот метод подходит для добавления трехмерных эффектов в содержимое Flash, а также для объектов с небольшим числом многоугольников. Во Flash Player 11 и AIR 3 или более поздних версий можно визуализировать сложные 3D-сцены с использованием API-интерфейса Stage3D.

Видовой экран Stage3D не является экранным объектом. Вместо этого 3D-графика визуализируется в видовом экране, который отображается под списком отображения Flash (и над плоскостями видового экрана StageVideo). Вместо создания сцены с помощью классов Flash DisplayObject используется программируемый 3D-туннель (наподобие OpenGL и Direct3D). Этот туннель принимает данные треугольников и текстуры в качестве ввода и визуализирует сцену с использованием предоставленных шейдерных программ. Аппаратное ускорение используется, когда клиентский компьютер оснащен поддерживаем графическим процессором с поддерживаемыми драйверами.

Stage3D предоставляет API-интерфейс на очень низком уровне. В приложении рекомендуется использовать 3D-структуру, поддерживающую Stage3D. Можно создать собственную структуру или использовать одну из предлагаемых на рынке за плату или находящихся в свободном доступе (с открытым исходным кодом).

Дополнительные сведения о разработке 3D-приложений с использованием Stage3D, а также о доступных 3D-структурах см. на сайте Flash Player Developer Center: Stage 3D.

## Основные сведения о трехмерных экранных объектах

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Основное различие между двухмерным (2D) и трехмерным (3D) объектом, спроецированным на двухмерный экран, заключается в том, что к трехмерному объекту добавляется третье измерение. Третье измерение позволяет объекту перемещаться к точке наблюдения пользователя и от нее.

Когда свойству `z` экранного объекта явно задается числовое значение, объект автоматически создает матрицу трехмерного преобразования. Эту матрицу можно изменить для изменения настроек трехмерного преобразования этого объекта.

Кроме того, поворот трехмерного объекта отличается от поворота двухмерного объекта. В двухмерном измерении ось поворота всегда перпендикулярна плоскости x/y, другими словами, она находится на оси z. В трехмерном измерении ось поворота может совпадать с любой из осей: x, y или z. Настройка свойств поворота и масштабирования экранного объекта позволяет ему перемещаться в трехмерном пространстве.

### Важные понятия и термины

Ниже приводится список важных терминов, встречающихся при программировании трехмерной графики.

**Перспектива** Представление параллельных линий, сходящихся в точке исчезновения для создания иллюзии глубины и расстояния на двумерной плоскости.

**Проецирование** Получение двумерного изображения объекта повышенной размерности; при проецировании трехмерного объекта выполняется наложение трехмерных точек на двумерную плоскость.

**Поворот** Изменение ориентации (и часто позиции) объекта путем перемещения по кругу каждой точки, включенной в объект.

**Преобразование** Изменение трехмерных точек или наборов точек путем перемещения, поворота, масштабирования, наклона или комбинации этих действий.

**Перевод** Изменение позиции объекта путем перемещения каждой включенной в него точки на одинаковое расстояние в одном направлении.

**Точка исчезновения** Точка, в которой сходятся удаляющиеся параллельные линии, представленные в линейной перспективе.

**Вектор** Трехмерный вектор представляет точку или местоположение в трехмерном пространстве с использованием декартовых координат x, y и z.

**Vertex** Вершина угла.

**Текстурированная ячейка** Любая точка, определяющая объект в трехмерном пространстве.

**UV-наложение** Способ нанесения текстуры или растрового изображения на трехмерную поверхность. UV-наложение присваивает значения координатам на изображении в виде процентного соотношения горизонтальной (U) и вертикальной (V) осей.

**значение T** Фактор масштабирования для определения размера трехмерного объекта по мере перемещения к текущей точке обзора или от нее.

**Отбор** Визуализация или отсутствие визуализации поверхностей с определенным искривлением. Применение отбора позволяет скрыть поверхности, невидимые с текущей точки обзора.

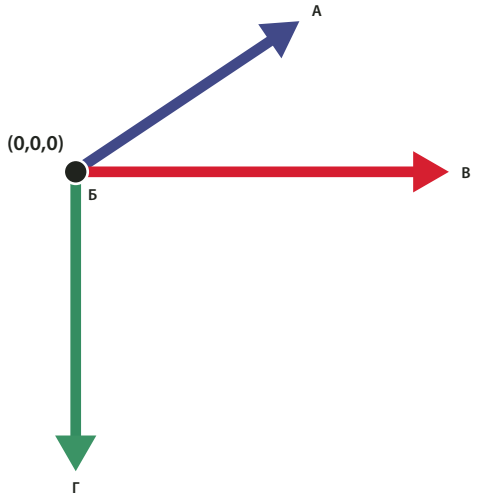
# Трехмерные экранные объекты во Flash Player и среде выполнения AIR

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Во всех версиях Flash Player до Flash Player 10 и Adobe AIR до Adobe AIR 1.5 экранные объекты имели два свойства: `x` и `y`, используемые для их позиционирования в двухмерном пространстве. Начиная с версии Flash Player 10 и Adobe AIR 1.5, каждый экранный объект ActionScript имеет свойство `z`, которое позволяет располагать объект вдоль оси z, обычно используемой для представления глубины или расстояния.

В приложениях Flash Player 10 и Adobe AIR 1.5 представлена поддержка трехмерных эффектов. Тем не менее, экранные объекты по сути являются плоскими. Каждый экранный объект, такой как MovieClip или Sprite, в конечном счете визуализируется в двух измерениях в одной плоскости. Трехмерные функции позволяют размещать, перемещать, поворачивать и преобразовывать каким-либо другим способом эти плоские объекты во всех трех измерениях. Они также позволяют управлять трехмерными точками и преобразовывать их в двухмерные координаты осей x и y для проецирования трехмерных объектов на двухмерную плоскость. С помощью этих функций можно моделировать различные трехмерные эффекты.

Используемая в ActionScript трехмерная система координат отличается от других систем. При использовании двухмерных координат в ActionScript значение координаты x увеличивается при перемещении вправо вдоль оси x, а значение координаты y увеличивается при перемещении вниз по оси y. Трехмерная система координат сохраняет эти условия и добавляет ось z, значение которой увеличивается при удалении от точки обзора.




*Положительные направления осей x, y и z в трехмерной системе координат ActionScript.*
***А.*** *+ ось Z* ***Б.*** *Начало координат* ***В.*** *+ ось X* ***Г.*** *+ ось Y*

***Примечание.*** *Важно помнить, что проигрыватель Flash Player и среда AIR всегда представляют трехмерное пространство в виде слоев. Это означает, что если объект A находится перед объектом B в списке отображения, то Flash Player или AIR всегда выполняет визуализацию A перед B независимо от значений по оси z для обоих объектов. Для разрешения этого конфликта между порядком элементов в списке отображения и порядком по оси z необходимо с помощью метода* `transform.getRelativeMatrix3D()` *сохранить слои трехмерных экранных объектов, а затем изменить их порядок. Дополнительные сведения см. в разделе ««Использование объектов Matrix3D для изменения порядка отображения» на странице 391».*

Следующие классы ActionScript поддерживают новые трехмерные функции:

1. Класс flash.display.DisplayObject содержит свойство `z`, а также новые свойства rotation и scaling для управления экранными объектами в трехмерном пространстве. Метод `DisplayObject.local3DToGlobal()` обеспечивает простой способ проецирования трехмерной геометрии на двухмерную плоскость.
2. Класс flash.geom.Vector3D можно использовать в качестве структуры данных для управления трехмерными точками. Он также поддерживает векторную математику.
3. Класс flash.geom.Matrix3D поддерживает сложные преобразования трехмерной геометрии, например, поворот, масштабирование и перемещение.
4. Класс flash.geom.PerspectiveProjection управляет параметрами для преобразования трехмерной геометрии в двухмерный вид.

В ActionScript имеется два разных подхода к моделированию трехмерных изображений.

1. Организация и анимация плоских объектов в трехмерном пространстве. Этот подход включает анимацию объектов отображения с использованием свойств x, y и z объектов отображения или путем присвоения значений свойствам rotation и scaling с помощью класса DisplayObject. Более сложное движение достигается при помощи объекта DisplayObject.transform.matrix3D. Объект DisplayObject.transform.perspectiveProjection позволяет настроить способ рисования объектов отображения в трехмерной перспективе. Используйте этот подход при анимации трехмерных объектов, имеющих в основном плоские поверхности. В качестве примера этого подхода можно привести галереи трехмерных изображений или двухмерные анимированные объекты, расположенные в трехмерном пространстве.

2 Создание двухмерных треугольников из трехмерных геометрических фигур и визуализация этих треугольников с применением текстур. Для использования этого подхода сначала необходимо определить и упорядочить данные о трехмерных объектах, а затем преобразовать их в двухмерные треугольники для выполнения визуализации. На эти треугольники можно наложить текстуры растровых изображений, а затем с помощью метода `Graphics.drawTriangles()` эти треугольники отображаются в объекте Graphics. Этот подход используется, например, для загрузки данных трехмерной модели из файла и визуализации этой модели на экране или для рисования трехмерной поверхности в виде сетки с треугольными ячейками.

# Создание и перемещение трехмерных экранных объектов

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Для преобразования двухмерного объекта отображения в трехмерный необходимо явно установить свойство `z` для числового значения. При присвоении значения свойству `z` для объекта отображения создается новый объект Transform. При установке значения для свойства `DisplayObject.rotationX` или `DisplayObject.rotationY` также создается новый объект Transform. Объект Transform содержит свойство `Matrix3D`, управляющее представлением экранного объекта в трехмерном пространстве.

Следующий код устанавливает координаты для объекта отображения с именем leaf:

```
leaf.x = 100; leaf.y = 50; leaf.z = -30;
```

Эти значения, а также производные от них свойства можно найти в свойстве `matrix3D` объекта Transform объекта отображения leaf.

```
var leafMatrix:Matrix3D  = leaf.transform.matrix3D;

trace(leafMatrix.position.x);
trace(leafMatrix.position.y);
trace(leafMatrix.position.z);
trace(leafMatrix.position.length);
trace(leafMatrix.position.lengthSquared);
```

Сведения о свойствах объекта Transform см. в описании класса Transform. Сведения о свойствах объекта Matrix3D см. в описании класса Matrix3D.

## перемещение объекта в трехмерном пространстве;

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Можно перемещать объект в трехмерном пространстве путем изменения значений его свойств `x, y, z`. При изменении значения свойства `z` кажется, что объект приближается к наблюдателю или отдаляется от него.

Следующий код перемещает два эллипса вперед и назад по оси z с помощью изменения значения их свойств `z` в ответ на событие. Объект `ellipse2` перемещается быстрее объекта `ellipse1`: его свойство `z` увеличивается в 20 раз при каждом событии Frame, а свойство `z` объекта `ellipse1` увеличивается в 10 раз.

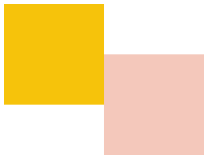
```
var depth:int = 1000;

function ellipse1FrameHandler(e:Event):void
{
    ellipse1Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 10;
}
function ellipse2FrameHandler(e:Event):void
{
    ellipse2Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 20;
}
function setDepth(e:Event, d:int):int
{
    if(e.currentTarget.z > depth)
    {
        e.currentTarget.z = depth;
        d = -1;
    }
    else if (e.currentTarget.z <  0)
    {
        e.currentTarget.z = 0;
        d = 1;
    }
}
```

## поворот объекта в трехмерном пространстве;

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Объект можно поворачивать тремя способами в зависимости от того, как устанавливаются свойства поворота объекта: `rotationX`, `rotationY` и `rotationZ`.

На следующем рисунке изображены два неповернутых квадрата:

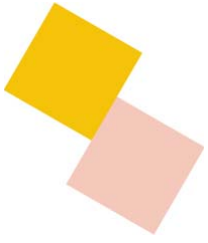


На следующем рисунке представлены два квадрата при увеличении значения свойства их контейнера `rotationY` для поворота квадратов вокруг оси y. Поворот контейнера или вышестоящего объекта отображения двух квадратов выполняет поворот обоих квадратов:

```
container.rotationY += 10;
```

На рисунке представлен результат установки значения для свойства `rotationX` контейнера квадратов. Благодаря этому квадраты поворачиваются вокруг оси x.

На рисунке представлен результат увеличения значения свойства `rotationZ` контейнера квадратов. Благодаря этому фигуры поворачиваются вокруг оси z.



Объект отображения можно одновременно перемещать и поворачивать в трехмерном пространстве.

# Проецирование трехмерных объектов в двухмерное пространство

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Класс PerspectiveProjection в пакете `flash.geom` обеспечивает простой способ применения элементарной перспективы при перемещении объектов отображения в трехмерном пространстве.

Если для трехмерного пространства не создана явно перспективная проекция, трехмерный модуль использует объект PerspectiveProjection по умолчанию, находящийся в корневом объекте и передающийся всем его нижестоящим элементам.

Три свойства, определяющие способ отображения трехмерного пространства объектом PerspectiveProjection:

- `fieldOfView`;
- `projectionCenter`;
- `focalLength`.

При изменении значения свойства `fieldOfView` автоматически изменяется значение свойства `focalLength` и наоборот, так как они являются взаимозависимыми.

Для вычисления значения `focalLength` при указанном значении `fieldOfView` используется следующая формула:

```
focalLength = stageWidth/2 * (cos(fieldOfView/2) / sin(fieldOfView/2)
```

Как правило, значение свойства `fieldOfView` изменяется явно.

## Поле обзора

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Посредством управления свойством `fieldOfView` класса PerspectiveProjection можно сделать приближающийся к пользователю трехмерный экранный объект зрительно больше, а удаляющийся — меньше.

Свойство `fieldOfView` указывает угол от 0 до 180 градусов, определяющий степень перспективной проекции. Чем больше значение, тем больше искажение, применяемое к экранному объекту, который перемещается по оси z. Использование низкого значения `fieldOfView` приводит к низкой степени масштабирования, и создается впечатление, что объекты лишь немного передвинулись назад. При использовании высокого значения `fieldOfView` искажение увеличивается, и создается впечатление, что объект передвинулся на большое расстояние. Использование максимального значения (179.9999 градусов) приводит к эффекту камеры с линзой типа «рыбий глаз». Максимальным значением `fieldOfView` является 179.9999..., а минимальным 0.00001... Только 0 и 180 являются допустимыми значениями.

## Центр проекций

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Свойство `projectionCenter` представляет точку схода в перспективной проекции. Оно применяется как сдвиг относительно точки регистрации по умолчанию (0,0), которая находится в верхнем левом углу рабочей области.

По мере отдаления объекта от пользователя, он смещается к точке схода и в конце концов исчезает. Представьте себе бесконечно длинный зал. Когда смотришь вглубь зала, вдалеке края стен сходятся в точке схода.

Если точка схода находится в центре рабочей области, зал исчезает в направлении точки в центре. По умолчанию свойство `projectionCenter` указывает центральную точку рабочей области. Например, если требуется, чтобы элементы отображались в левой части рабочей области, а трехмерная область – в правой части, требуется установить для свойства `projectionCenter` точку в правой части рабочей области, чтобы сделать ее точкой схода трехмерной области просмотра.

## Фокусное расстояние

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Свойство `focalLength` представляет расстояние между исходной точкой поля зрения (0,0,0) и местоположением экранного объекта на его оси z.

Большое фокусное расстояние создает эффект применения телеобъектива с узким видом и сжатым расстоянием между объектами. Маленькое фокусное расстояние создает эффект применения угловой линзы с широким видом и большим искажением. Среднее фокусное расстояние приблизительно соответствует тому, что видит человеческий глаз.

Как правило, значение `focalLength` вычисляется динамически несколько раз в течение преобразования перспективы при удалении объекта, но его также можно задать явно.

## Значения по умолчанию для перспективной проекции

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Объект PerspectiveProjection по умолчанию, созданный в корневом объекте, имеет следующие значения:

- `fieldOfView: 55`
- `perspectiveCenter: stagewidth/2, stageHeight/2`
- `focalLength: stageWidth/ 2 * ( cos(fieldOfView/2) / sin(fieldOfView/2) )`

Эти значение используются, если не создается собственный объект PerspectiveProjection.

Для самостоятельного изменения свойств `projectionCenter` и `fieldOfView` можно создать экземпляр PerspectiveProjection. В данном случае вновь созданный объект имеет следующие значения по умолчанию, исходя из размера рабочей области по умолчанию 500x500:

- `fieldOfView: 55`
- `perspectiveCenter: 250,250`
- `focalLength: 480.24554443359375`

# Пример: перспективная проекция

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

В примере представлено использование перспективной проекции для создания трехмерного пространства. В нем описывается способ указания другой точки схода или изменение перспективной проекции пространства с помощью свойства `projectionCenter`. Это изменение приводит к повторному вычислению значений `focalLength` и `fieldOfView` и возникновению сопутствующего искажения трехмерного пространства.

В этом примере:

1. Создается спрайт с именем `center`, представляющий круг с крестиком;
2. Координаты спрайта `center` назначаются свойству `projectionCenter` свойства `perspectiveProjection` для свойства `transform` корневого объекта;
3. Добавляются прослушиватели событий мыши, вызывающие обработчики, которые изменяют значение `projectionCenter` таким образом, что центр перспективы перемещается вслед за объектом `center`;
4. Создаются четыре окна в виде аккордеона, образующие стены пространства перспективы.

При тестировании этого примера, ProjectionDragger.swf, требуется перетаскивание круга в разные местоположения. Точка схода будет перемещаться вслед за кругом и оставаться там, где ее разместит пользователь. Обратите внимание, что окна, в которые заключено пространство, растягиваются и искажаются при перемещении центра проекции на большое расстояние от центра рабочей области.

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения ProjectionDragger находятся в папке Samples/ProjectionDragger.

```
package
{
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.geom.Point;
    import flash.events.*;
    public class ProjectionDragger extends Sprite
    {
        private var center : Sprite;
        private var boxPanel:Shape;
        private var inDrag:Boolean = false;

        public function ProjectionDragger():void
        {
            createBoxes();
            createCenter();
        }
        public function createCenter():void
        {
            var  centerRadius:int = 20;

            center = new Sprite();

            // circle
            center.graphics.lineStyle(1, 0x000099);
            center.graphics.beginFill(0xCCCCCC, 0.5);
            center.graphics.drawCircle(0, 0, centerRadius);
            center.graphics.endFill();
            // cross hairs
            center.graphics.moveTo(0, centerRadius);
            center.graphics.lineTo(0, -centerRadius);
            center.graphics.moveTo(centerRadius, 0);
            center.graphics.lineTo(-centerRadius, 0);
            center.x = 175;
            center.y = 175;
            center.z = 0;
            this.addChild(center);

            center.addEventListener(MouseEvent.MOUSE_DOWN, startDragProjectionCenter);
            center.addEventListener(MouseEvent.MOUSE_UP, stopDragProjectionCenter);
            center.addEventListener( MouseEvent.MOUSE_MOVE, doDragProjectionCenter);
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
        }
        public function createBoxes():void
        {
            // createBoxPanel();
            var boxWidth:int = 50;
            var boxHeight:int = 50;
            var numLayers:int = 12;
            var depthPerLayer:int = 50;

            // var boxVec:Vector.<Shape> = new Vector.<Shape>(numLayers);
```

```
            for (var i:int = 0; i < numLayers; i++)
            {
                this.addChild(createBox(150, 50, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xCCCCFF));
                this.addChild(createBox(50, 150, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xFFCCCC));
                this.addChild(createBox(250, 150, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xCCFFCC));
                this.addChild(createBox(150, 250, (numLayers - i) * depthPerLayer, boxWidth,
boxHeight, 0xDDDDDD));
            }
        }

        public function createBox(xPos:int = 0, yPos:int = 0, zPos:int = 100, w:int = 50, h:int
= 50, color:int = 0xDDDDDD):Shape
        {
            var box:Shape = new Shape();
            box.graphics.lineStyle(2, 0x666666);
            box.graphics.beginFill(color, 1.0);
            box.graphics.drawRect(0, 0, w, h);
            box.graphics.endFill();
            box.x = xPos;
            box.y = yPos;
            box.z = zPos;
            return box;
        }
        public function startDragProjectionCenter(e:Event)
        {
            center.startDrag();
            inDrag = true;
        }

        public function doDragProjectionCenter(e:Event)
        {
            if (inDrag)
            {
                root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
            }
        }

        public function stopDragProjectionCenter(e:Event)
        {
            center.stopDrag();
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x,
center.y);
            inDrag = false;
        }
    }
}
```

Для создания более сложной перспективной проекции требуется использовать класс Matrix3D.

# Выполнение сложных трехмерных преобразований

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Класс Matrix3D позволяет преобразовывать трехмерные точки в рамках координатной плоскости или переносить трехмерные точки из одной плоскости координат в другую.

Для использования класса Matrix3D не требуется знание матричной математики. С помощью методов этого класса можно обрабатывать большую часть общих операций преобразования. Нет необходимости явно устанавливать или вычислять значения для каждого элемента в матрице.

После установки свойства `z` объекта отображения на числовое значение можно получить матрицу преобразования этого объекта с помощью свойства Matrix3D его объекта Transform.

```
var leafMatrix:Matrix3D = this.transform.matrix3D;
```

С помощью методов объекта Matrix3D можно выполнять перемещение, поворот, масштабирование и перспективную проекцию объекта отображения.

Для управления трехмерными точками можно использовать класс Vector3D и его свойства `x`, `y` и `z`. Этот класс также может представлять пространственный вектор в физике, имеющий направление и величину. Методы класса Vector3D позволяют выполнять общие вычисления с пространственными векторами, например, сложение, скалярное произведение и векторное произведение.

***Примечание.*** *Класс Vector3D не связан с классом Vector в ActionScript. Класс Vector3D содержит свойства и методы для определения трехмерных точек и управления ими, а класс Vector поддерживает массивы объектов со строгим контролем типов.*

## Создание объектов Matrix3D

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Существует три основных способа создания или получения объектов `Matrix3D`:

- Использование конструктора `Matrix3D()` для создания новой матрицы. Конструктор `Matrix3D()` принимает объект `Vector`, содержащий 16 числовых значений, и помещает каждое из них в ячейку матрицы. Например:

  ```
  var rotateMatrix:Matrix3D = new Matrix3D(1,0,0,1, 0,1,0,1, 0,0,1,1, 0,0,0,1);
  ```

- Указание числового значения для свойства `z` любого объекта отображения. Матрицу преобразования после этого можно получить, запросив свойство `transform.matrix3D` этого объекта.
- Получение объекта Matrix3D, управляющего отображением трехмерных объектов в рабочей области, происходит путем вызова метода `perspectiveProjection.toMatrix3D()` корневого объекта отображения.

## Применение нескольких трехмерных преобразований

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

С помощью одного объекта Matrix3D можно одновременно применить несколько трехмерных преобразований. Например, если требуется повернуть, масштабировать, а затем переместить куб, можно применить три отдельных преобразования для каждой точки куба. Однако более эффективным является выполнение предварительного расчета нескольких преобразований в одном объекте Matrix3D, а затем выполнение одного преобразования матрицы для каждой из этих точек.

***Примечание.*** *Важную роль играет порядок, в котором применяются преобразования матрицы. Вычисления матрицы на являются коммутативными. Например, применение поворота с последующим перемещением дает не тот же результат, что применение такого же перемещения с последующим поворотом.*

Пример демонстрирует два способа выполнения нескольких трехмерных преобразований.

```
package {
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;
    import flash.geom.*;

public class Matrix3DTransformsExample extends Sprite
    {
        private var rect1:Shape;
        private var rect2:Shape;

public function Matrix3DTransformsExample():void
        {
            var pp:PerspectiveProjection = this.transform.perspectiveProjection;
            pp.projectionCenter = new Point(275,200);
            this.transform.perspectiveProjection = pp;

            rect1 = new Shape();
            rect1.x = -70;
            rect1.y = -40;
            rect1.z = 0;
            rect1.graphics.beginFill(0xFF8800);
            rect1.graphics.drawRect(0,0,50,80);
            rect1.graphics.endFill();
            addChild(rect1);

            rect2 = new Shape();
            rect2.x = 20;
            rect2.y = -40;
            rect2.z = 0;
            rect2.graphics.beginFill(0xFF0088);
            rect2.graphics.drawRect(0,0,50,80);
            rect2.graphics.endFill();
            addChild(rect2);
```

```
            doTransforms();
        }

        private function doTransforms():void
        {
            rect1.rotationX = 15;
            rect1.scaleX = 1.2;
            rect1.x += 100;
            rect1.y += 50;
            rect1.rotationZ = 10;

            var matrix:Matrix3D = rect2.transform.matrix3D;
            matrix.appendRotation(15, Vector3D.X_AXIS);
            matrix.appendScale(1.2, 1, 1);
            matrix.appendTranslation(100, 50, 0);
            matrix.appendRotation(10, Vector3D.Z_AXIS);
            rect2.transform.matrix3D = matrix;
        }
    }
}
```

В методе `doTransforms()` первый блок кода использует свойства DisplayObject для изменения поворота, масштабирования и позиции прямоугольной фигуры. Второй блок кода использует методы класса Matrix3D для выполнения тех же преобразований.

Главное преимущество использования методов `Matrix3D` заключается в том, что все вычисления выполняются сначала в матрице. Затем выполняется их применение к объекту отображения только один раз при указании его свойства `transform.matrix3D`. Установка свойств DisplayObject немного облегчает чтение исходного кода. Однако каждый раз при установке свойства поворота или масштабирования выполняются многочисленные расчеты и изменение многих свойств объекта отображения.

При применении одних и тех же сложных преобразований к объектам отображения с помощью кода более одного раза требуется сохранить объект Matrix3D как переменную для повторного его использования.

## Использование объектов Matrix3D для изменения порядка отображения

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Как было указано ранее, порядок слоев объектов отображения в списке отображения определяет порядок слоев отображения независимо от положения объектов по соответствующим осям z. При преобразовании с помощью анимации свойства объектов отображения в порядке, отличном от порядка в списке отображения, пользователь может увидеть неправильное последовательное расположение объектов отображения, не соответствующее расположению слоев по оси z. Таким образом, объект, который должен находиться дальше от пользователя, может быть отображен перед объектом, находящимся ближе к пользователю.

Для соответствия расположения слоев трехмерных объектов отображения соответствующей глубине объектов требуется использовать, например, следующий подход:

1. Используйте метод `getRelativeMatrix3D()` объекта Transform для получения относительных значений `z-axes` нижестоящих трехмерных объектов отображения.
2. Используйте метод `removeChild()` для удаления объектов из списка отображения.
3. Отсортируйте объекты отображения на основе их относительных значений по оси z.
4. Используйте метод `addChild()` для повторного добавления нижестоящих объектов в список отображения в обратном порядке.

Такая перестановка позволяет выстроить объекты отображения в соответствии с их относительными значениями по оси z.

Следующий код обеспечивает правильное отображение шести граней параллелепипеда. Он изменяет порядок граней параллелепипеда после применения поворотов:

```
public var faces:Array; . . .

public function ReorderChildren()
{
    for(var ind:uint = 0; ind < 6; ind++)
    {
        faces[ind].z = faces[ind].child.transform.getRelativeMatrix3D(root).position.z;
        this.removeChild(faces[ind].child);
    }
    faces.sortOn("z", Array.NUMERIC | Array.DESCENDING);
    for (ind = 0; ind < 6; ind++)
    {
        this.addChild(faces[ind].child);
    }
}
```

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения находятся в папке Samples/ReorderByZ.

# Использование треугольников для создания трехмерных эффектов

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

В ActionScript преобразования растрового изображения выполняются с помощью метода `Graphics.drawTriangles()`, так как трехмерные модели представляются в виде набора треугольников в пространстве. (Однако проигрыватель Flash Player и среда AIR не поддерживают буфер глубины, поэтому экранные объекты по существу остаются плоскими, то есть двухмерными. Подробнее см. в разделе ««Трехмерные экранные объекты во Flash Player и среде выполнения AIR» на странице 380».) Метод `Graphics.drawTriangles()` напоминает метод `Graphics.drawPath()`, так как он принимает набор координат для рисования контура треугольника.

Сведения об использовании метода `Graphics.drawPath()` см. в разделе ««Создание контуров» на странице 254».

Метод `Graphics.drawTriangles()` использует Vector.<Number> для указания местоположения точек для контура треугольника:

```
drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null, uvtData:Vector.<Number>
= null, culling:String = "none"):void
```

Первым и единственным обязательным параметром метода `drawTriangles()` является параметр `vertices`. Этот параметр представляет собой вектор чисел, определяющих координаты точек, через которые рисуются треугольники. Каждые три группы координат (шесть чисел) представляют контур треугольника. Если не задан параметр `indices`, длина вектора всегда должна быть кратна шести, так как каждому треугольнику требуется три пары координат (три набора из пары значений x/y). Например:

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([
        10,10,  100,10,  10,100,
        110,10, 110,100, 20,100]));
```

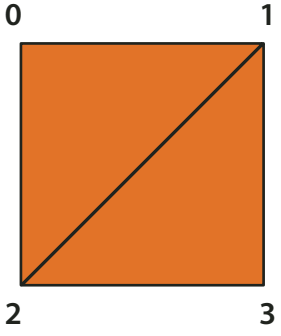
Ни один из этих треугольников не имеет общих точек с остальными, но при наличии таких точек второй параметр метода `drawTriangles()`, `indices`, можно было бы использовать для повторного использования значений в векторе `vertices` для нескольких треугольников.

При использовании параметра `indices` следует помнить, что значения `indices` являются индексами точек, а не элементами массива `vertices`. Другими словами, индекс в векторе `vertices`, определенный параметром `indices`, на самом деле является настоящим индексом, разделенным на 2. Например, для третьей точки вектора `vertices` требуется использовать значение 2 для `indices`, несмотря на то, что первое числовое значение этой точки начинается в индексе вектора 4.

Например, с помощью параметра `indices` требуется объединить два треугольника так, чтобы у них была общая гипотенуза.

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([10,10, 100,10, 10,100, 100,100]),
    Vector.<int>([0,1,2, 1,3,2]));
```

Обратите внимание, что квадрат нарисован с помощью двух треугольников, и были указаны всего четыре точки в векторе `vertices`. При использовании `indices`, две точки, общие для двух треугольников, используются повторно при построении каждого из них. Это сокращает общее число вершин с 6 (12 чисел) до 4 (8 чисел).




*Квадрат, полученный при построении двух треугольников с помощью параметра vertices*

Этот прием полезен при создании больших сеток с треугольными ячейками, где большинство точек являются общими для многих треугольников.

К треугольникам можно применять все возможные заливки. Заливки применяются к полученной сетке с треугольными ячейками так же, как к любой другой фигуре.

## Преобразование растровых изображений

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Преобразования растровых изображений создают иллюзию перспективы или «текстуры» на трехмерном объекте. В частности, можно исказить растровое изображение в направлении точки схода, чтобы изображение зрительно сжималось по мере удаления от точки зрения. Также можно использовать двухмерное растровое изображение для создания поверхности трехмерного объекта, придавая вид текстуры или «оболочки» этому объекту.

*Двухмерная поверхность с использованием точки схода и трехмерный объект, заключенный в оболочку растровым изображением.*

## UV-наложение

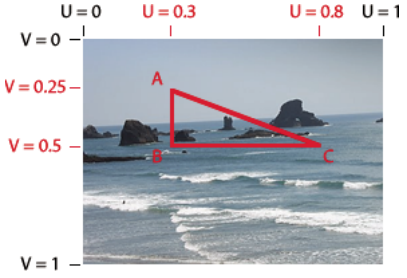
**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

При работе с текстурами необходимо использовать параметр uvtData метода `drawTriangles()`. Этот параметр позволяет настраивать UV-наложение для растровых заливок.

UV-наложение — это метод текстурирования объектов. В нем используются два значения: значение U по горизонтали (x) и значение V по вертикали (y). Эти значения представлены не в пикселах, а в процентах. Значения 0 U и 0 V определяют верхнюю левую точку изображения, а 1 U и 1 V — нижнюю правую.

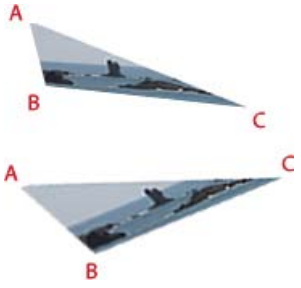
*Точки UV 0 и 1 на растровом изображении*

Можно указать UV-координаты для векторов треугольника для связывания их с соответствующими точками на изображении.



*UV-координаты треугольной области растрового изображения*

UV-значения согласуются с точками треугольника.



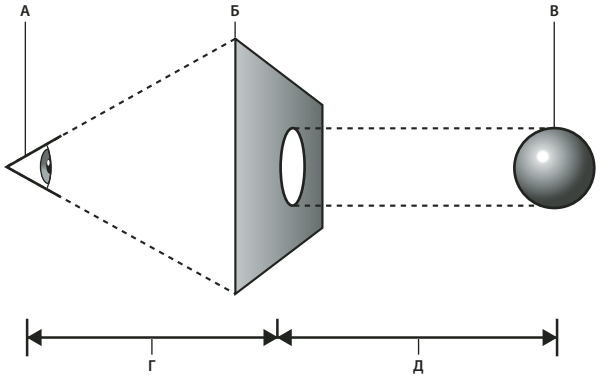
*Вершины треугольника перемещаются, и растровое изображение искажается для сохранения неизменяемых UV-значений для каждой точки*

При применении преобразований ActionScript к треугольнику, связанному с растровым изображением, это изображение накладывается на треугольник на основе UV-значений. Поэтому вместо использования вычислений матрицы трехмерный эффект создается путем указания или изменения UV-значений.

Метод `Graphics.drawTriangles()` также принимает дополнительный элемент информации для трехмерных трансформаций: значение T. Значение T в uvtData представляет трехмерную перспективу, а именно коэффициент масштабирования для связанной с ним вершины. UVT-наложение добавляет поправку перспективы к UV-наложению. Например, при помещении объекта в трехмерном пространстве в отдалении от точки зрения так, что он зрительно уменьшается на 50% по сравнению с исходным размером его значение T будет равно 0,5. Так как треугольники рисуются для представления объектов в трехмерном пространстве, их значение T определяется местоположением по оси z. Ниже приводится уравнение для определения значения T.

```
T = focalLength/(focalLength + z);
```

В этом уравнении focalLength представляет фокусное расстояние или вычисленное местоположение «экрана», определяющее степень перспективы, применяемой к данному виду.




*Фокусное расстояние и значение z*
***А.** точка зрения **Б.** экран **В.** трехмерный объект **Г.** значение focalLength **Д.** значение z*

Значение T используется для масштабирования простых фигур, чтобы они казались дальше. Как правило, это значение используется для преобразование трехмерных точек в двухмерные. В случае UVT-данных это значение также используется для масштабирования растрового изображения между точками в пределах треугольника с перспективой.

При определении UVT-значений значение T следует непосредственно за значениями UV, определенными для вершины. При добавлении T каждые три значения параметра `uvtData` (U, V и T) соответствуют каждым двум значениями в параметре `vertices` (x и y). При использовании только значений UV uvtData.length == vertices.length. При включении значения T uvtData.length = 1.5*vertices.length.

Следующий пример демонстрирует плоскость, которая поворачивается в трехмерном пространстве с использованием данных UVT. В этом примере используется изображение ocean.jpg и вспомогательный класс ImageLoader, предназначенный для загрузки изображения ocean.jpg и его назначения объекту BitmapData.

Ниже приводится исходный код класса ImageLoader (необходимо сохранить его в файле с именем ImageLoader.as).

```
package {
    import flash.display.*
    import flash.events.*;
    import flash.net.URLRequest;
    public class ImageLoader extends Sprite {
        public var url:String;
        public var bitmap:Bitmap;
    public function ImageLoader(loc:String = null) {
        if (loc != null){
            url = loc;
            loadImage();
        }
    }
    public function loadImage():void{
        if (url != null){
            var loader:Loader = new Loader();
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onComplete);
            loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR, onIoError);

                var req:URLRequest = new URLRequest(url);
                loader.load(req);
            }
        }

    private function onComplete(event:Event):void {
            var loader:Loader = Loader(event.target.loader);
            var info:LoaderInfo = LoaderInfo(loader.contentLoaderInfo);
            this.bitmap = info.content as Bitmap;
            this.dispatchEvent(new Event(Event.COMPLETE));
    }

    private function onIoError(event:IOErrorEvent):void {
            trace("onIoError: " + event);
    }
    }
}
```

А этот код ActionScript использует треугольники, UV-наложение и значения T для создания иллюзии, что изображение сжимается в направлении точки схода и поворачивается. Требуется сохранить этот код в файле с именем Spinning3dOcean.as:

```
package {
    import flash.display.*
    import flash.events.*;
    import flash.utils.getTimer;

    public class Spinning3dOcean extends Sprite {
        // plane vertex coordinates (and t values)
        var x1:Number = -100,y1:Number = -100,z1:Number = 0,t1:Number = 0;
        var x2:Number = 100,y2:Number = -100,z2:Number = 0,t2:Number = 0;
        var x3:Number = 100,y3:Number = 100,z3:Number = 0,t3:Number = 0;
        var x4:Number = -100,y4:Number = 100,z4:Number = 0,t4:Number = 0;
        var focalLength:Number = 200;
        // 2 triangles for 1 plane, indices will always be the same
        var indices:Vector.<int>;

        var container:Sprite;

        var bitmapData:BitmapData; // texture
        var imageLoader:ImageLoader;
        public function Spinning3dOcean():void {
            indices =  new Vector.<int>();
            indices.push(0,1,3, 1,2,3);

            container = new Sprite(); // container to draw triangles in
            container.x = 200;
            container.y = 200;
            addChild(container);

            imageLoader = new ImageLoader("ocean.jpg");
            imageLoader.addEventListener(Event.COMPLETE, onImageLoaded);
        }
        function onImageLoaded(event:Event):void {
            bitmapData = imageLoader.bitmap.bitmapData;
            // animate every frame
            addEventListener(Event.ENTER_FRAME, rotatePlane);
        }
        function rotatePlane(event:Event):void {
            // rotate vertices over time
            var ticker = getTimer()/400;
            z2 = z3 = -(z1 = z4 = 100*Math.sin(ticker));
            x2 = x3 = -(x1 = x4 = 100*Math.cos(ticker));

            // calculate t values
```

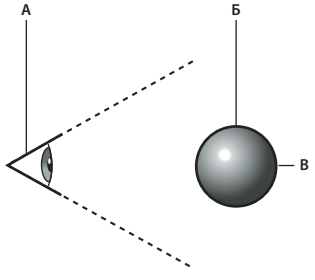
```
            t1 = focalLength/(focalLength + z1);
            t2 = focalLength/(focalLength + z2);
            t3 = focalLength/(focalLength + z3);
            t4 = focalLength/(focalLength + z4);

            // determine triangle vertices based on t values
            var vertices:Vector.<Number> = new Vector.<Number>();
            vertices.push(x1*t1,y1*t1, x2*t2,y2*t2, x3*t3,y3*t3, x4*t4,y4*t4);
            // set T values allowing perspective to change
            // as each vertex moves around in z space
            var uvtData:Vector.<Number> = new Vector.<Number>();
            uvtData.push(0,0,t1, 1,0,t2, 1,1,t3, 0,1,t4);

            // draw
            container.graphics.clear();
            container.graphics.beginBitmapFill(bitmapData);
            container.graphics.drawTriangles(vertices, indices, uvtData);
        }
    }
}
```

Для тестирования этого примера необходимо сохранить эти два файла классов в том же каталоге, что и изображение с именем ocean.jpg. Можно видеть, как исходное растровое изображение преобразуется, как будто оно исчезает в перспективе и поворачивается в трехмерном пространстве.

## Отбор

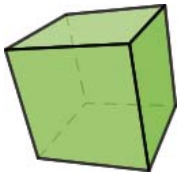
**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Отбором называется процесс, который определяет, какие поверхности трехмерного объекта не должно визуализировать средство визуализации, так как они не видны с текущей точки зрения. В трехмерном пространстве поверхность, расположенная на обратной стороне трехмерного объекта, не видна с текущей точки зрения.




*Обратная сторона трехмерного объекта скрыта для текущей точки обзора.*
***А.** точка зрения **Б.** трехмерный объект **В.** обратная сторона трехмерного объекта*

Все треугольники по своей сути всегда визуализируются независимо от размера, формы и позиции. Отбор обеспечивает правильную визуализацию трехмерного объекта в проигрывателе Flash Player или среде AIR. Кроме того, с целью экономии на циклах визуализации иногда требуется, чтобы средство визуализации пропускало некоторые треугольники. Рассмотрим куб, поворачивающийся в пространстве. В любой момент времени вы видите только три стороны этого куба, так как остальные стороны обращены в другую сторону и не видны с данной точки обзора. Так как эти стороны не будут видны, средству визуализации не нужно их рисовать. Проигрыватель Flash Player или среда AIR визуализирует без отбора как переднюю, так и заднюю сторону.



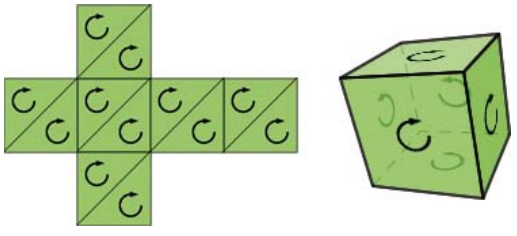
*У куба есть стороны, которые не видны с текущей точки зрения*

Поэтому метод `Graphics.drawTriangles()` имеет четвертый параметр для указания значения отбора:

```
public function drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null,
uvtData:Vector.<Number> = null, culling:String = "none"):void
```

Параметр отбора представляет значение из класса перечисления `TriangleCulling`: `TriangleCulling.NONE`, `TriangleCulling.POSITIVE` и `TriangleCulling.NEGATIVE`. Эти значения зависят от направления контура треугольника, определяющего поверхность объекта. API-интерфейс ActionScript для определения отбора предполагает, что все треугольники трехмерной фигуры, обращенные в сторону точки зрения, рисуются с одинаковым направлением контура. Когда поверхность треугольника поворачивается, направление контура меняется. На этом этапе выполняется отбор треугольника (то есть он исключается из визуализации).

Таким образом, параметр `TriangleCulling` со значением `POSITIVE` убирает треугольники с положительным направлением контура (по часовой стрелке). Параметр `TriangleCulling` со значением `NEGATIVE` убирает треугольники с отрицательным направлением контура (против часовой стрелки). В случае с кубом, когда стороны, обращенные вперед, имеют положительное направление контура, стороны, обращенные назад, имеют отрицательное направление контура.



*Куб без оболочки для демонстрации направления контура. Куб в оболочке; направление контура задней стороны повернуто в обратную сторону*

Для просмотра принципа работы отбора необходимо начать с предыдущего примера из раздела ««UV-наложение» на странице 394», установить для параметра отбора метода `drawTriangles()` значение `TriangleCulling.NEGATIVE`.

```
container.graphics.drawTriangles(vertices, indices, uvtData, TriangleCulling.NEGATIVE);
```

Обратите внимание, что при повороте объекта, «обратная» сторона изображения не визуализируется.

# Глава 20. Основы работы с текстом

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для отображения текста на экране в Adobe® Flash® Player или Adobe® AIR™ можно использовать экземпляр класса TextField или классы Flash Text Engine. Эти классы позволяют создавать, отображать и форматировать текст. В качестве альтернативного варианта можно использовать Text Layout Framework (TLF) - легкий в использовании компонент библиотеки на основе классов Flash Text Engine. На мобильных устройствах для ввода текста можно использовать класс StageText.

Для текстовых полей можно выбрать определенное содержимое или назначить источник текста, а затем настроить внешний вид этого текста. Можно также реагировать на события пользователя по мере того, как он вводит текст или переходит по гиперссылкам.

Как класс TextField, так и классы Flash Text Engine позволяют отображать текст в программах Flash Player и AIR. Класс TextField можно применять, чтобы создавать текстовые объекты для отображения и ввода. Класс TextField является основой для других текстовых компонентов, таких как TextArea и TextInput. Класс TextFormat можно использовать для настройки форматирования символов и абзацев для объектов TextField, а с помощью свойства Textfield.styleSheet и класса StyleSheet можно применять каскадные таблицы стилей (Cascading Style Sheets, CSS). Текстовому полю можно напрямую назначить текст с форматированием HTML, в том числе содержащий встроенные мультимедийные файлы (фрагменты роликов, SWF-файлы, GIF-файлы, PNG-файлы и JPEG-файлы).

Класс Flash TextEngine (FTE), доступный для использования во Flash Player версии 10 и AIR версии 1.5, обеспечивает поддержку низкого уровня для расширенного управления метриками текста, форматированием и двунаправленным текстом. Он также обеспечивает улучшенный поток текста и расширенную поддержку языков. Несмотря на то, что Flash Text Engine можно применять для создания и управления элементами текста, он был разработан, в основном, в качестве платформы для создания компонентов обработки текста и требует большего опыта программирования. Text Layout Framework, включающий компонент обработки текста, который основан на Flash Text Engine, позволяет легче прибегать к дополнительным функциям нового механизма визуализации текста. Text Layout Framework представляет собой обширную библиотеку, полностью встроенную в ActionScript 3.0. Можно использовать как существующий компонент TLF, так и инфраструктуру для создания пользовательского текстового компонента.

Класс StageText, доступный с версии AIR 3, предоставляет собственное поле ввода текста. Поле, предоставляемое операционной системой устройства, позволяет создать наиболее привычную для пользователей среду. Экземпляр StageText не является экранным объектом. Вместо добавления его в список отображения экземпляр следует назначить для рабочей области и области отображения на ней, называемой видовым экраном. Экземпляр StageText отображается перед любыми экранными объектами.

Дополнительные сведения по этим темам см. в следующих разделах.


- «Использование класса TextField» на странице 403
- «Использование Flash Text Engine» на странице 428
- «Использование Text Layout Framework» на странице 458
- Native text input with StageText (собственный ввод текста с помощью StageText)

**Важные понятия и термины**

В следующем справочном списке содержатся важные термины, используемые для описания обработки текста.

**Каскадные таблицы стилей** Стандартный синтаксис для определения стилей и форматирования содержимого, организованный в формате XML (или HTML).

**Шрифт устройства** Шрифт, установленный на компьютере пользователя.

**Динамическое текстовое поле** Текстовое поле, содержимое которого можно изменить с помощью ActionScript, но не с помощью пользовательского ввода.

**Встроенный шрифт** Шрифт, данные контура символов которого сохранены в SWF-файле приложения.

**HTML-текст** Текстовое содержимое, введенное в текстовое поле с помощью ActionScript и включающее теги форматирования наряду с текстом.

**Текстовое поле ввода** Поле, содержимое которого может быть изменено посредством пользовательского ввода или посредством кода ActionScript.

**Kerning** Корректировка расстояния между парами символов для получения более пропорциональных интервалов между символами и улучшения удобочитаемости текста.

**Статическое текстовое поле** Поле, созданное в инструменте разработки, содержимое которого не может изменяться во время выполнения SWF-файла.

**Метрики строки текста** Размеры разных частей текстового содержимого в текстовом поле, таких как базовая линия текста, высота верхнего элемента символов, размер подстрочных элементов (элементов строчных букв, опускающиеся ниже базовой линии) и т. д.

**Трекинг** Корректировка расстояния между группами букв или блоками текста для увеличения или уменьшения плотности и улучшения удобочитаемости текста.

# Глава 21. Использование класса TextField

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Экземпляр класса TextField можно использовать для отображения текста или создания поля для ввода текста на экране в приложении Adobe® Flash® Player или Adobe® AIR™. Класс TextField является основой для других текстовых компонентов, таких как компоненты TextArea или TextInput.

Содержимое текстового поля может быть предварительно определено в SWF-файле, загружено из текстового файла или базы данных либо введено пользователем в ходе действий пользователя в приложении. Текст может отображаться в поле как визуализированное HTML-содержимое, включая изображения, встроенные в HTML. После создания экземпляра текстового поля для управления внешним видом текста можно использовать классы flash.text, например TextFormat и StyleSheet. Пакет flash.text содержит практически все классы, связанные с созданием, форматированием текста и управлением им в ActionScript.

Для изменения формата текста можно определить форматирование с помощью объекта TextField и назначить этот объект текстовому полю. Если текстовое поле содержит HTML-текст, к нему можно применить объект StyleSheet, чтобы назначить стили определенным фрагментам содержимого. Объект TextFormat или StyleSheet содержит свойства, определяющие внешний вид текста, включая цвет, размер и толщину шрифта. Объект TextFormat назначает свойства всему содержимому текстового поля или диапазону текста. Например, в одном и том же текстовом поле одно предложение может быть выделено полужирным красным текстом, а следующее — синим курсивом.

Помимо классов в пакете flash.text можно использовать класс flash.events.TextEvent, чтобы реагировать на действия пользователя, связанные с текстом.

### Дополнительные разделы справки

«Назначение форматов текста» на странице 411

«Отображение HTML-текста» на странице 405

«Применение каскадных таблиц стилей» на странице 412

## Отображение текста

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Хотя в инструментах разработки, таких как Adobe Flash Builder и Flash Professional, предусмотрено несколько способов отображения текста, включая текстовые компоненты и текстовые инструменты, простейшим способом программного отображения текста является использование текстового поля.

## Типы текста

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Тип текста в текстовом поле определяется по его источнику.

- Динамический текст

  Динамический текст включает содержимое, загруженное из внешнего источника, например, текстового файла, XML-файла или даже удаленной web-службы.

- Вводимый текст

  Вводимый текст — это текст, введенный пользователем, или динамический текст, который может быть изменен пользователем. Можно создать таблицу стилей для форматирования вводимого текста или использовать класс flash.text.TextFormat для назначения свойств текстовому полю для вводимого содержимого. Дополнительные сведения см. в разделе ««Захват введенного текста» на странице 409».

- Статический текст

  Статический текст создается только в инструменте Flash Professional. Экземпляр статического текста нельзя создать с помощью ActionScript 3.0. Однако классы ActionScript, такие как StaticText и TextSnapshot, можно использовать для манипуляций с существующим экземпляром статического текста. Дополнительные сведения см. в разделе «Работа со статическим текстом» на странице 417.

## Изменение содержимого текстового поля

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Можно определить динамический текст, назначив строку свойству `flash.text.TextField.text`. Строку можно назначить свойству напрямую, как показано ниже:

```
myTextField.text = "Hello World";
```

Также можно назначить свойству `text` значение из переменной, определенной в сценарии, как показано в следующем примере:

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class TextWithImage extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myText:String = "Hello World";

        public function TextWithImage()
        {
            addChild(myTextBox);
            myTextBox.text = myText;
        }
    }
}
```

Или же свойству `text` можно назначить значение из удаленной переменной. Существует три способа загрузки текстовых значений из удаленных источников:

- Классы flash.net.URLLoader и flash.net.URLRequest загружают переменные для текста из локального или удаленного местоположения.
- В HTML-страницу, содержащую SWF-файл, встраивается атрибут `FlashVars`, который может содержать значения для текстовых переменных.
- Класс flash.net.SharedObject управляет постоянным хранением значений. Дополнительную информацию см. в разделе «Хранение локальных данных» на странице 754.

## Отображение HTML-текста

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс flash.text.TextField имеет свойство `htmlText`, которое можно использовать для определения текстовой строки с HTML-тегами для форматирования содержимого. Как показано в следующем примере, для визуализации текста Flash Player или AIR как HTML необходимо назначить строковое значение свойству `htmlText` (а не свойству `text`).

```
var myText:String = "<p>This is <b>some</b> content to <i>render</i> as <u>HTML</u> text.</p>";
myTextBox.htmlText = myText;
```

Flash Player и AIR поддерживают подгруппу тегов и сущностей HTML для свойства `htmlText`. Дополнительные сведения о поддерживаемых тегах и сущностях HTML см. в описании свойства `flash.text.TextField.htmlText` в справочнике ActionScript® 3.0 для Adobe® Flash® Professional CS5.

Указав содержимое с помощью свойства `htmlText`, можно использовать таблицы стилей или тег `textformat` для управления форматированием содержимого. Дополнительные сведения см. в разделе «Форматирование текста» на странице 411.

## Использование изображений в текстовых полях

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Еще одним преимуществом отображения содержимого в виде HTML-текста является то, что в текстовые поля можно включать изображения. Для ссылки на локальное или удаленное изображение можно использовать тег `img`, который следует добавить в соответствующее текстовое поле.

В следующем примере создается текстовое поле `myTextBox`, в котором в отображаемый текст включено изображение глаза в формате JPG, сохраненное в том же каталоге, что и SWF-файл.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class TextWithImage extends Sprite
    {
        private var myTextBox:TextField;
        private var myText:String = "<p>This is <b>some</b> content to <i>test</i> and
<i>see</i></p><p><img src='eye.jpg' width='20' height='20'></p><p>what can be
rendered.</p><p>You should see an eye image and some <u>HTML</u> text.</p>";

        public function TextWithImage()
        {
            myTextBox.width = 200;
            myTextBox.height = 200;
            myTextBox.multiline = true;
            myTextBox.wordWrap = true;
            myTextBox.border = true;

            addChild(myTextBox);
            myTextBox.htmlText = myText;
        }
    }
}
```

Тег `img` поддерживает теги в формате JPEG, GIF, PNG и SWF.

## Прокрутка текста в текстовом поле

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Во многих случаях длина текста превышает длину текстового поля, в котором отображается текст. Иногда поля ввода позволяют ввести текст, выходящий за пределы области отображения. Свойства класса flash.text.TextField, связанные с прокруткой, можно использовать для управления объемным содержимым с помощью прокрутки по вертикали или по горизонтали.

С прокруткой связаны свойства `TextField.scrollV`, `TextField.scrollH`, `maxScrollV` и `maxScrollH`. Их можно использовать для реагирования на такие события, как щелчок мыши или нажатие клавиши.

В следующем примере создается текстовое поле фиксированного размера, которое содержит текст с длиной, превышающей размер отображаемой области. Когда пользователь щелкает текстовое поле, выполняется вертикальная прокрутка текста.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;
    import flash.events.MouseEvent;

    public class TextScrollExample extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myText:String = "Hello world and welcome to the show. It's really nice to
meet you. Take your coat off and stay a while. OK, show is over. Hope you had fun. You can go
home now. Don't forget to tip your waiter. There are mints in the bowl by the door. Thank you.
Please come again.";

        public function TextScrollExample()
        {
            myTextBox.text = myText;
            myTextBox.width = 200;
            myTextBox.height = 50;
            myTextBox.multiline = true;
            myTextBox.wordWrap = true;
            myTextBox.background = true;
            myTextBox.border = true;

            var format:TextFormat = new TextFormat();
            format.font = "Verdana";
            format.color = 0xFF0000;
            format.size = 10;

            myTextBox.defaultTextFormat = format;
            addChild(myTextBox);
            myTextBox.addEventListener(MouseEvent.MOUSE_DOWN, mouseDownScroll);
        }

        public function mouseDownScroll(event:MouseEvent):void
        {
            myTextBox.scrollV++;
        }
    }
}
```

# Выделение и обработка текста

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Динамический или вводимый текст может быть выделен. Свойства и методы выделения текста в классе TextField используют позиции индекса для выбора нужного диапазона текста, поэтому динамический или вводимый текст можно выделить с помощью кода, даже если содержимое поля неизвестно.

***Примечание.*** *Если в инструменте Flash Professional для статического текстового поля выбран параметр «С возможностью выделения», то текстовое поле, экспортированное и помещенное в список отображения, является обычным динамический текстовым полем.*

## Выделение текста

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Свойство `flash.text.TextField.selectable` по умолчанию имеет значение `true`. Текст можно выделить с помощью кода, используя метод `setSelection()`.

Например, можно ввести определенный текст в поле, который будет выделяться, когда пользователь щелкает текстовое поле:

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL CAPS is selected.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, selectText);

function selectText(event:MouseEvent):void
{
    myTextField.setSelection(49, 65);
}
```

Подобным образом, если требуется, чтобы текст в поле выделялся, то сразу после появления текста на экране следует создать функцию обработчика событий, вызываемую при добавлении текстового поля в список отображения.

## Захват текста, выделенного пользователем

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Свойства `selectionBeginIndex` и `selectionEndIndex` TextField доступны только для чтения, поэтому их нельзя использовать для программного выделения текста. Они служат для захвата текущего фрагмента, выделенного пользователем. Кроме того, поля ввода текста могут использовать свойство `caretIndex`.

Например, следующий код отслеживает значения индекса для текста, выделенного пользователем.

```
var myTextField:TextField = new TextField();
myTextField.text = "Please select the TEXT IN ALL CAPS to see the index values for the first
and last letters.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.MOUSE_UP, selectText);

function selectText(event:MouseEvent):void
{
    trace("First letter index position: " + myTextField.selectionBeginIndex);
    trace("Last letter index position: " + myTextField.selectionEndIndex);
}
```

Для изменения внешнего вида текста к выделенному фрагменту можно применить коллекцию свойств объекта TextFormat. Дополнительные сведения о применении набора свойств TextFormat к выделенному тексту см. в разделе «Форматирование диапазонов текста в текстовом поле» на странице 414.

# Захват введенного текста

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

По умолчанию свойство `type` текстового поля имеет значение `dynamic`. При установке свойства `type` на значение `input` с помощью класса TextFieldType можно получить пользовательский ввод и сохранить это значение для использования в других частях приложения. Текстовые поля ввода можно использовать в формах и приложениях для определения текстовых значений, используемых в другом месте программы.

Например, следующий код создает текстовое поле ввода `myTextBox`. При вводе пользователем текста в поле инициируется событие `textInput`. Обработчик события с именем `textInputCapture` захватывает строку введенного текста и назначает его переменной. Flash Player или AIR отображает новый текст в другом текстовом окне с именем `myOutputBox`.

```
package
{
    import flash.display.Sprite;
    import flash.display.Stage;
    import flash.text.*;
    import flash.events.*;

    public class CaptureUserInput extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myOutputBox:TextField = new TextField();
        private var myText:String = "Type your text here.";

        public function CaptureUserInput()
        {
            captureText();
        }

        public function captureText():void
        {
            myTextBox.type = TextFieldType.INPUT;
            myTextBox.background = true;
            addChild(myTextBox);
```

```
            myTextBox.text = myText;
            myTextBox.addEventListener(TextEvent.TEXT_INPUT, textInputCapture);
        }

        public function textInputCapture(event:TextEvent):void
        {
            var str:String = myTextBox.text;
            createOutputBox(str);
        }

        public function createOutputBox(str:String):void
        {
            myOutputBox.background = true;
            myOutputBox.x = 200;
            addChild(myOutputBox);
            myOutputBox.text = str;
        }

    }
}
```

# Ограничение ввода текста

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Так как текстовые поля ввода часто используются в приложениях для форм и диалоговых окон, в некоторых случаях может потребоваться ограничение типов символов, разрешенных к вводу в том или ином текстовом поле, или даже скрытие текста, например, текста пароля. Класс flash.text.TextField имеет свойства `displayAsPassword` и `restrict`, с помощью которых можно управлять пользовательским вводом.

Свойство `displayAsPassword` просто скрывает текст, вводимый пользователем, отображая его в виде звездочек. Если свойство `displayAsPassword` имеет значение `true`, команды «Вырезать» и «Копировать», а также соответствующие комбинации клавиш не действуют. Как показано в следующем примере, свойство `displayAsPassword` назначается так же, как и другие свойства, например, background и color.

```
myTextBox.type = TextFieldType.INPUT;
myTextBox.background = true;
myTextBox.displayAsPassword = true;
addChild(myTextBox);
```

Свойство `restrict` является более сложным, так как при его использовании необходимо указать, какие символы разрешены для ввода в текстовое поле. Можно разрешить ввод определенных букв, цифр или диапазонов букв, цифр и символов. Следующий код позволяет пользователю вводить в текстовое поле только заглавные буквы (а числа или специальные символы — нет).

```
myTextBox.restrict = "A-Z";
```

В ActionScript 3.0 дефисы используются для определения диапазонов, а символы вставки — для определения исключенных символов. Дополнительные сведения об ограничении ввода в текстовое поле см. в описании свойства `flash.text.TextField.restrict` в справочнике ActionScript® 3.0 для Adobe® Flash® Professional CS5.

***Примечание.*** *Если используется свойство `flash.text.TextField.restrict`, среда выполнения автоматически преобразует неразрешенные буквы в допустимый шрифт. Если используется свойство `fl.text.TLFTextField.restrict` (то есть используется текстовое поле TLF), среда выполнения игнорирует неразрешенные буквы.*

# Форматирование текста

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Существует несколько способов форматирования отображаемого текста с помощью кода. Можно задать свойства непосредственно для экземпляра TextField, например, `TextFIeld.thickness`, `TextField.textColor` и `TextField.textHeight`.Также можно обозначить содержимое текстового поля с помощью свойства `htmlText` и использовать поддерживаемые теги HTML, например, `b`, `i` и `u`. Кроме того, можно применить объекты TextFormat к текстовым полям, содержащим обычный текст, или применить объекты StyleSheet к текстовым полям со свойством `htmlText`. Использование объектов TextFormat и StyleSheet обеспечивает самый полный контроль над внешним видом текста в рамках всего приложения. Можно определить объект TextFormat или StyleSheet и применить его к нескольким или ко всем текстовым полям в приложении.

## Назначение форматов текста

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс TextFormat позволяет определить ряд различных свойств отображения текста и применить их ко всему содержимому объекта TextField или к диапазону текста.

В следующем примере объект TextFormat применяется ко всему объекту TextField, а второй объект TextFormat применяется к диапазону текста в этом же объекте TextField.

```
var tf:TextField = new TextField();
tf.text = "Hello Hello";

var format1:TextFormat = new TextFormat();
format1.color = 0xFF0000;

var format2:TextFormat = new TextFormat();
format2.font = "Courier";

tf.setTextFormat(format1);
var startRange:uint = 6;
tf.setTextFormat(format2, startRange);

addChild(tf);
```

Метод `TextField.setTextFormat()` воздействует только на текст, уже отображаемый в текстовом поле. При изменении содержимого в TextField приложение повторно вызывает метод `TextField.setTextFormat()` для повторного применения форматирования. Также можно задать свойство `defaultTextFormat` объекта TextField, чтобы указать формат для текста, вводимого пользователем.

## Применение каскадных таблиц стилей

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Текстовые поля могут содержать неформатированный текст или текст в формате HTML. Неформатированный текст сохраняется в свойстве `text` экземпляра, а HTML-текст — в свойстве `htmlText`.

С помощью объявлений CSS-стилей можно определить стили текста, применяемые ко многим другим текстовым полям. Объявления CSS-стилей можно создать в коде приложения или загрузить во время выполнения из внешнего CSS-файла.

Класс flash.text.StyleSheet обрабатывает CSS-стили. Класс StyleSheet распознает ограниченный набор свойств CSS. Подробный список свойств стиля, поддерживаемых классом StyleSheet, см. в описании flash.textStylesheet в справочнике ActionScript® 3.0 для Adobe® Flash® Professional CS5.

Как показано в следующем примере, CSS-стили можно создать в коде и применить их к HTML-тексту с помощью объекта StyleSheet.

```
var style:StyleSheet = new StyleSheet();

var styleObj:Object = new Object();
styleObj.fontSize = "bold";
styleObj.color = "#FF0000";
style.setStyle(".darkRed", styleObj);

var tf:TextField = new TextField();
tf.styleSheet = style;
tf.htmlText = "<span class = 'darkRed'>Red</span> apple";

addChild(tf);
```

После создания объекта StyleSheet пример кода создает простой объект для хранения набора свойств, объявляющих стиль. Затем вызывается метод `StyleSheet.setStyle()`, который добавляет в таблицу стилей новый стиль с именем «.darkred». После этого форматирование таблицы стилей применяется путем назначения объекта StyleSheet свойству TextField `styleSheet`.

Чтобы CSS-стили вступили в силу, таблицу стилей необходимо применить к объекту TextField до того, как будет задано свойство `htmlText`.

Текстовое поле с таблицей стилей не подлежит редактированию. Если применить таблицу стилей к существующему текстовому полю ввода, оно отображает свойства таблицы стилей, но не позволяет пользователям вводить новый текст. Кроме того, если текстовому полю присвоена таблица стилей, для него нельзя использовать следующие API-интерфейсы ActionScript:

- Метод `TextField.replaceText()`
- Метод `TextField.replaceSelectedText()`
- Свойство `TextField.defaultTextFormat`
- Метод `TextField.setTextFormat()`

Если текстовому полю присвоена таблица стилей, но впоследствии свойство `TextField.styleSheet` устанавливается на значение `null`, содержимое свойств `TextField.text` и `TextField.htmlText` добавляет для содержимого теги и атрибуты, чтобы включить форматирование из назначенной ранее таблицы стилей. Для сохранения исходного свойства `htmlText` следует сохранить его в переменной перед установкой таблицы стилей на значение `null`.

## Загрузка внешнего CSS-файла

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Форматирование с помощью CSS обеспечивает больше возможностей, если данные CSS загружаются из внешнего файла во время выполнения. Если приложение использует внешние данные CSS, отображаемый стиль текста в приложении можно изменить, не затрагивая исходный код ActionScript 3.0. После развертывания приложения внешний вид приложения можно изменить путем модификации внешнего CSS-файла. Такой метод не требует разворачивать SWF-повторно.

Метод `StyleSheet.parseCSS()` преобразует данные CSS в объявления стилей в объекте StyleSheet. В следующем примере показано, как считывается внешний CSS-файл и как его объявления стилей применяются к объекту TextField.

Ниже приводится содержимое загружаемого CSS-файла с именем example.css.

```
p {
    font-family: Times New Roman, Times, _serif;
    font-size: 14;
}

h1 {
    font-family: Arial, Helvetica, _sans;
    font-size: 20;
    font-weight: bold;
}

.bluetext {
    color: #0000CC;
}
```

Код ActionScript создает класс, который загружает файл example.css и применяет стили к содержимому TextField:

```
package
{
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import flash.text.StyleSheet;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;

    public class CSSFormattingExample extends Sprite
    {
        var loader:URLLoader;
        var field:TextField;
        var exampleText:String = "<h1>This is a headline</h1>" +
            "<p>This is a line of text. <span class='bluetext'>" +
            "This line of text is colored blue.</span></p>";

        public function CSSFormattingExample():void
        {
            field = new TextField();
            field.width = 300;
```

```
            field.autoSize = TextFieldAutoSize.LEFT;
            field.wordWrap = true;
            addChild(field);

            var req:URLRequest = new URLRequest("example.css");

            loader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
            loader.load(req);
        }

        public function onCSSFileLoaded(event:Event):void
        {
            var sheet:StyleSheet = new StyleSheet();
            sheet.parseCSS(loader.data);
            field.styleSheet = sheet;
            field.htmlText = exampleText;
        }
    }
}
```

После загрузки данных CSS выполняется метод `onCSSFileLoaded()`, который вызывает метод `StyleSheet.parseCSS()` для преобразования объявлений стилей в объект StyleSheet.

## Форматирование диапазонов текста в текстовом поле

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Полезным методом класса flash.text.TextField является метод `setTextFormat()`. С помощью `setTextFormat()` можно назначить определенные свойства содержимому части текстового поля, чтобы оно реагировало на пользовательский ввод (например, для вывода напоминания об обязательных полях при заполнении форм) или чтобы изменить стиль начертания фрагмента текста в поле, когда пользователь выделяет части текста.

В следующем примере метод `TextField.setTextFormat()` вызывается для диапазона символов, чтобы изменить внешний вид части содержимого `myTextField`, когда пользователь щелкает мышью в текстовом поле.

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL CAPS changes
format.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, changeText);

var myformat:TextFormat = new TextFormat();
myformat.color = 0xFF0000;
myformat.size = 18;
myformat.underline = true;

function changeText(event:MouseEvent):void
{
    myTextField.setTextFormat(myformat, 49, 65);
}
```

# Расширенная визуализация текста

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

ActionScript 3.0 имеет большой выбор классов в пакете flash.text, которые позволяют контролировать свойства отображаемого текста, включая встроенные шрифты, настройки сглаживания, управление альфа-каналом и другие специальные настройки. Подробное описание этих классов и свойств, включая классы CSMSettings, Font и TextRenderer, можно найти в справочнике ActionScript® 3.0 для Adobe® Flash® Professional CS5.

## Использование встроенных шрифтов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если в приложении указывается определенный шрифт для экземпляра TextField, Flash Player или AIR выполняет поиск одноименного шрифта устройства (шрифта, сохраненного на компьютере пользователя). Если в системе не удается найти требуемый шрифт или если установлена другая версия одноименного шрифта, текст может отображаться не так, как предполагалось. По умолчанию для текста, выводимого на экран, используется шрифт Times Roman.

Чтобы пользователь видел правильный шрифт, его можно встроить в файл приложения. Встроенные шрифты имеют ряд преимуществ.

- Символы встроенных шрифтов сглаживаются, благодаря чему их края выглядят более ровными, особенно в крупном тексте.
- Текст, отображаемый с помощью встроенных шрифтов, можно поворачивать.
- Текст, использующий встроенные шрифты, можно сделать прозрачным или полупрозрачным.
- Для встроенных шрифтов можно использовать CSS-стиль `kerning`.

Самый серьезный недостаток встроенных шрифтов заключается в том, что они увеличивают размер файла или загружаемого файла приложения.

Выбор метода встраивания файла шрифта в SWF-файл приложения зависит от среды разработки.

Встроив шрифт, можно предписать объекту TextField использовать правильный встроенный шрифт.

- Следует установить для свойства `embedFonts` экземпляра TextField значение `true`.
- Следует создать объект TextFormat, назначить его свойству `fontFamily` имя встроенного шрифта и применить объект TextFormat к экземпляру TextField. При указания встроенного шрифта свойство `fontFamily` должно содержать только одно имя. Недопустимо использовать список из нескольких названий шрифтов, разделенных запятыми.
- Если стили CSS используются для настройки шрифтов для TextFields или компонентов, следует установить в качестве значения свойства CSS `font-family` имя встроенного шрифта. Если требуется указать встроенный шрифт, свойство `font-family` должно содержать только одно имя, а не список имен.

### Встраивание шрифтов во Flash

Инструмент Flash Professional позволяет встраивать практически любые шрифты, установленные в системе, включая шрифты TrueType и Type 1 Postscript.

Встраивать шрифты в приложения можно разными способами, включая:

- настройку свойств font и style экземпляра TextField в рабочей области с последующей установкой флажка «Встроить шрифты»;

- создание символа шрифта и ссылку на него;
- создание и использование динамической общей библиотеки, содержащей символы встроенных шрифтов.

Дополнительные сведения о встраивании шрифтов в приложения см. в разделе «Встроенные шрифты для динамических текстовых полей и текстовых полей ввода» в руководстве *Использование Flash*.

**Встраивание шрифтов во Flex**

Во Flex встраивать шрифты в приложения можно разными способами, включая:

- использование в сценарии тега метаданных `[Embed]`;
- использование объявления стиля `@font-face`.
- Создайте класс для шрифта и встройте его с помощью тега `[Embed]`.

Непосредственно встраивать в приложения Flex можно только шрифты TrueType. Шрифты других форматов, например, Type 1 Postscript, сначала можно встроить в SWF-файл с помощью инструмента Flash Professional, после чего этот SWF-файл можно использовать в приложении Flex. Дополнительные сведения об использовании встроенных шрифтов из SWF-файлов в среде Flex см. в разделе «Встраивание шрифтов из SWF-файлов» руководства *Использование Flex 4*.

### Дополнительные разделы справки

Внедрение шрифтов для согласованного внешнего вида текста

Питер Дехаан (Peter deHaan): Внедрение шрифтов

Divillysausages.com: Мастер-класс по внедрению шрифтов AS3

## Настройка резкости, толщины и сглаживания

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Flash Player или AIR по умолчанию определяет настройки для таких элементов управления отображением текста, как резкость, толщина и сглаживание при изменении размера, цвета или фона текста. В некоторых случаях, когда текст очень мелкий или очень крупный, а также когда он отображается на одном из множества уникальных фонов, важно сохранять контроль над этими настройками. Для переопределения настроек Flash Player или AIR можно использовать класс `flash.text.TextRenderer` и связанные с ним классы, например, CSMSettings. Эти классы дают полный контроль над качеством визуализации текста, использующего встроенные шрифты. Дополнительные сведения о встроенных шрифтах см. в разделе «Использование встроенных шрифтов» на странице 415.

***Примечание.*** *Свойство `flash.text.TextField`.antiAliasType должно иметь значение `AntiAliasType.ADVANCED`, чтобы можно было отрегулировать резкость, толщину или свойство gridFitType, а также использовать метод `TextRenderer.setAdvancedAntiAliasingTable()`.*

В следующем примере к отображаемому тексту применяются пользовательские свойства модуляции непрерывной обводки (CSM) и форматирование с помощью встроенного шрифта `myFont`. При щелчке мышью на отображаемом тексте Flash Player или Adobe AIR применяет пользовательские настройки.

```
var format:TextFormat = new TextFormat();
format.color = 0x336699;
format.size = 48;
format.font = "myFont";

var myText:TextField = new TextField();
myText.embedFonts = true;
myText.autoSize = TextFieldAutoSize.LEFT;
myText.antiAliasType = AntiAliasType.ADVANCED;
myText.defaultTextFormat = format;
myText.selectable = false;
myText.mouseEnabled = true;
myText.text = "Hello World";
addChild(myText);
myText.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:Event):void
{
    var myAntiAliasSettings = new CSMSettings(48, 0.8, -0.8);
    var myAliasTable:Array = new Array(myAntiAliasSettings);
    TextRenderer.setAdvancedAntiAliasingTable("myFont", FontStyle.ITALIC,
TextColorType.DARK_COLOR, myAliasTable);
}
```

# Работа со статическим текстом

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Статический текст создается только в инструменте Flash Professional. Создать экземпляр статического текста с помощью программного кода с использованием ActionScript невозможно. Статический текст удобно использовать, если текст короткий и не предназначен для изменения (в отличие от динамического текста). Статический текст похож на графический элемент, такой как круг или квадрат, нарисованный в рабочей области в инструменте Flash Professional. Хотя статический текст имеет больше ограничений, чем динамический, ActionScript 3.0 позволяет считывать значения его свойств с помощью класса StaticText. Также значения статического текста можно считывать с помощью класса TextSnapshot.

## Доступ к статическим текстовым полям с помощью класса StaticText

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Обычно для взаимодействия с экземпляром статического текста, добавленным в рабочую область, используется класс flash.text.StaticText на панели «Действия» инструмента Flash Professional. Кроме того, можно работать в файлах ActionScript, взаимодействующих с SWF-файлом, содержащим статический текст. В любом случае статический текст невозможно создать с помощью программного кода. Статический текст создается в инструменте Flash Professional.

Для создания ссылки на существующее статическое текстовое поле необходимо выполнить повторное прохождение для элементов в списке отображения и назначить переменную. Например:

```
for (var i = 0; i < this.numChildren; i++) {
var displayitem:DisplayObject = this.getChildAt(i);
if (displayitem instanceof StaticText) {
trace("a static text field is item " + i + " on the display list");
        var myFieldLabel:StaticText = StaticText(displayitem);
        trace("and contains the text: " + myFieldLabel.text);
}
}
```

Создав ссылку на статическое текстовое поле, можно использовать его свойства в ActionScript 3.0. Следующий код присоединяется к кадру на временной шкале, а переменная `myFieldLabel` назначается ссылке на статический текст. Динамическое текстовое поле `myField` помещается в точке с координатами `x` и `y`, определенными в `myFieldLabel`, и повторно отображает значение `myFieldLabel`.

```
var myField:TextField = new TextField();
addChild(myField);
myField.x = myFieldLabel.x;
myField.y = myFieldLabel.y + 20;
myField.autoSize = TextFieldAutoSize.LEFT;
myField.text = "and " + myFieldLabel.text
```

## Использование класса TextSnapshot

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если требуется обработать существующий экземпляр текста с помощью программного кода, можно использовать класс flash.text.TextSnapshot для получения свойства `textSnapshot` экземпляра flash.display.DisplayObjectContainer. Другими словами, можно создать экземпляр TextSnapshot из свойства `DisplayObjectContainer.textSnapshot`. Затем, применив методы к этому экземпляру, можно получить значения или выделить фрагменты статического текста.

Например, поместите в рабочую область статическое текстовое поле с текстом «Пример TextSnapshot». Добавьте следующий код ActionScript в кадр 1 на временной шкале:

```
var mySnap:TextSnapshot = this.textSnapshot;
var count:Number = mySnap.charCount;
mySnap.setSelected(0, 4, true);
mySnap.setSelected(1, 2, false);
var myText:String = mySnap.getSelectedText(false);
trace(myText);
```

Класс TextSnapshot полезен для получения текста из статических текстовых полей в загруженном SWF-файле, если этот текст требуется использовать в качестве значения в другой части приложения.

# Пример класса TextField: форматирование текста в газетном стиле

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В примере News Layout текст форматируется в виде газетной статьи. Вводимый текст может содержать заголовок, подзаголовок и тело статьи. Используя заданные ширину и высоту отображения, пример News Layout форматирует заголовок и подзаголовок так, чтобы они занимали всю ширину области отображения. Текст статьи распределяется по двум или более столбцам.

В этом примере демонстрируется использование следующих приемов программирования на ActionScript:

- расширение класса TextField;
- загрузка и применение внешнего CSS-файла;
- преобразование стилей CSS в объекты TextFormat;
- использование класса TextLineMetrics для получения информации о размере области отображения текста.

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения News Layout находятся в папке Samples/NewsLayout. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| NewsLayout.mxml<br>или<br>NewsLayout.fla | Пользовательский интерфейс приложения для Flex (MXML) или Flash (FLA). |
| com/example/programmingas3/newslayout/StoryLayoutComponent.as | Класс UIComponent Flex, который размещает экземпляр StoryLayout. |
| com/example/programmingas3/newslayout/StoryLayout.as | Основной класс ActionScript, упорядочивающий все компоненты статьи новостей, предназначенной для отображения. |
| com/example/programmingas3/newslayout/FormattedTextField.as | Подкласс класса TextField, управляющий собственным объектом TextFormat. |
| com/example/programmingas3/newslayout/HeadlineTextField.as | Подкласс класса FormattedTextField, регулирующий размеры шрифта в соответствии с требуемой шириной. |
| com/example/programmingas3/newslayout/MultiColumnTextField.as | Класс ActionScript, разбивающий текст на два столбца или более. |
| story.css | CSS-файл, определяющий стили текста для макета. |

## Чтение внешнего CSS-файла

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Приложение News Layout сначала выполняет чтение текста статьи из локального XML-файла. Затем считывается внешний CSS-файл, предоставляющий данные форматирования для заголовка, подзаголовка и основного текста.

CSS-файл определяет три стиля: стандартный стиль абзаца для текста статьи и стили h1 и h2 для заголовка и подзаголовка соответственно.

```
p {
    font-family: Georgia, "Times New Roman", Times, _serif;
    font-size: 12;
    leading: 2;
    text-align: justify;
    indent: 24;
}

h1 {
    font-family: Verdana, Arial, Helvetica, _sans;
    font-size: 20;
    font-weight: bold;
    color: #000099;
    text-align: left;
}

h2 {
    font-family: Verdana, Arial, Helvetica, _sans;
    font-size: 16;
    font-weight: normal;
    text-align: left;
}
```

Для чтения внешнего CSS-файла используется прием, описанный в разделе «Загрузка внешнего CSS-файла» на странице 413. После загрузки CSS-файла приложение выполняет метод `onCSSFileLoaded()`, как показано ниже.

```
public function onCSSFileLoaded(event:Event):void
{
    this.sheet = new StyleSheet();
    this.sheet.parseCSS(loader.data);

    h1Format = getTextStyle("h1", this.sheet);
    if (h1Format == null)
    {
        h1Format = getDefaultHeadFormat();
    }
    h2Format = getTextStyle("h2", this.sheet);
    if (h2Format == null)
    {
        h2Format = getDefaultHeadFormat();
        h2Format.size = 16;
    }
    pFormat = getTextStyle("p", this.sheet);
    if (pFormat == null)
    {
        pFormat = getDefaultTextFormat();
        pFormat.size = 12;
    }
    displayText();
}
```

Метод `onCSSFileLoaded()` создает объект StyleSheet, который анализирует введенные данные CSS. Основной текст статьи отображается в объекте MultiColumnTextField, который может напрямую использовать объект StyleSheet. Однако поля заголовков используют класс HeadlineTextField, который реализует форматирование с помощью объекта TextFormat.

Метод `onCSSFileLoaded()` дважды вызывает метод `getTextStyle()`, чтобы преобразовать объявление CSS-стиля в объект TextFormat для использования с двумя объектами HeadlineTextField.

```
public function getTextStyle(styleName:String, ss:StyleSheet):TextFormat
{
    var format:TextFormat = null;

    var style:Object = ss.getStyle(styleName);
    if (style != null)
    {
        var colorStr:String = style.color;
        if (colorStr != null && colorStr.indexOf("#") == 0)
        {
            style.color = colorStr.substr(1);
        }
        format = new TextFormat(style.fontFamily,
                        style.fontSize,
                        style.color,
                        (style.fontWeight == "bold"),
                        (style.fontStyle == "italic"),
                        (style.textDecoration == "underline"),
                        style.url,
                        style.target,
                        style.textAlign,
                        style.marginLeft,
                        style.marginRight,
                        style.indent,
                        style.leading);

        if (style.hasOwnProperty("letterSpacing"))
        {
            format.letterSpacing = style.letterSpacing;
        }
    }
    return format;
}
```

Имена свойств и смысл их значений для объявлений CSS-стилей и объектов TextFormat отличаются друг от друга. Метод `getTextStyle()` преобразует значения свойств CSS в значения, которые может принять объект TextFormat.

## Организация элементов статьи на странице

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс StoryLayout форматирует и размещает поля заголовка, подзаголовка и основного текста в газетном стиле. Сначала метод `displayText()` создает и размещает различные поля.

```
public function displayText():void
{
    headlineTxt = new HeadlineTextField(h1Format);
    headlineTxt.wordWrap = true;
    headlineTxt.x = this.paddingLeft;
    headlineTxt.y = this.paddingTop;
    headlineTxt.width = this.preferredWidth;
    this.addChild(headlineTxt);

    headlineTxt.fitText(this.headline, 1, true);

    subtitleTxt = new HeadlineTextField(h2Format);
    subtitleTxt.wordWrap = true;
    subtitleTxt.x = this.paddingLeft;
    subtitleTxt.y = headlineTxt.y + headlineTxt.height;
    subtitleTxt.width = this.preferredWidth;
    this.addChild(subtitleTxt);

    subtitleTxt.fitText(this.subtitle, 2, false);

    storyTxt = new MultiColumnText(this.numColumns, 20,
                        this.preferredWidth, 400, true, this.pFormat);
    storyTxt.x = this.paddingLeft;
    storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
    this.addChild(storyTxt);

    storyTxt.text = this.content;
...
```

Каждое поле помещается под предыдущим. Для этого свойство `y` определяется как сумма значения `y` предыдущего поля и его высоты. Такое динамическое вычисление позиции требуется потому, что объекты HeadlineTextField и MultiColumnTextField могут изменять высоту в соответствии с размерами содержимого.

## Изменение размера шрифта в соответствии с размером поля

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Используя заданную ширину в пикселах и максимальное число отображаемых строк, объект HeadlineTextField изменяет размер шрифта так, чтобы текст поместился в поле. Если текст короткий, используется большой размер шрифта, в результате чего заголовок приобретает стиль таблоида. Если текст длинный, то размер шрифта становится меньше.

Метод `HeadlineTextField.fitText()`, показанный ниже, изменяет размер шрифта.

```
public function fitText(msg:String, maxLines:uint = 1, toUpper:Boolean = false,
targetWidth:Number = -1):uint
{
    this.text = toUpper ? msg.toUpperCase() : msg;

    if (targetWidth == -1)
    {
        targetWidth = this.width;
    }

    var pixelsPerChar:Number = targetWidth / msg.length;

    var pointSize:Number = Math.min(MAX_POINT_SIZE, Math.round(pixelsPerChar * 1.8 * maxLines));

    if (pointSize < 6)
    {
        // the point size is too small
        return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(--pointSize, maxLines);
    }
    else
    {
        return growText(pointSize, maxLines);
    }
}

public function growText(pointSize:Number, maxLines:uint = 1):Number
{
    if (pointSize >= MAX_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize + 1);

    if (this.numLines > maxLines)
    {
        // set it back to the last size
        this.changeSize(pointSize);
        return pointSize;
    }
    else
    {
        return growText(pointSize + 1, maxLines);
    }
```

```
}

public function shrinkText(pointSize:Number, maxLines:uint=1):Number
{
    if (pointSize <= MIN_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(pointSize - 1, maxLines);
    }
    else
    {
        return pointSize;
    }
}
```

Метод `HeadlineTextField.fitText()` использует простой рекурсивный прием для настройки размера шрифта. Сначала он подсчитывает среднее число пикселов на символ текста, а затем вычисляет начальный размер в пунктах. После этого метод изменяет размер шрифта и проверяет, не превышено ли максимальное количество строк в результате переноса слов. Если строк слишком много, вызывается метод `shrinkText()`, чтобы уменьшить размер шрифта и повторить попытку. Если строк меньше, вызывается метод `growText()`, чтобы увеличить размер шрифта и повторить попытку. Процесс прекращается, когда увеличение размера шрифта на один пункт приводит к превышению максимального числа строк.

## Разбивка текста на несколько столбцов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс MultiColumnTextField распределяет текст по нескольким объектам TextField, в результате чего текст принимает вид газетных столбцов.

Сначала конструктор `MultiColumnTextField()` создает массив объектов TextField, по одному на каждый столбец, как показано ниже.

```
for (var i:int = 0; i < cols; i++)
{
    var field:TextField = new TextField();
    field.multiline = true;
    field.autoSize = TextFieldAutoSize.NONE;
    field.wordWrap = true;
    field.width = this.colWidth;
    field.setTextFormat(this.format);
    this.fieldArray.push(field);
    this.addChild(field);
}
```

Каждый объект TextField добавляется в массив, а затем помещается в список отображения с помощью метода `addChild()`.

Каждый раз, когда изменяется свойство `text` или `styleSheet` StoryLayout, этот класс вызывает метод `layoutColumns()` для повторного отображения текста. Метод `layoutColumns()` вызывает метод `getOptimalHeight()` для вычисления правильной высоты в пикселах, необходимую для отображения всего текста при заданной ширине.

```
public function getOptimalHeight(str:String):int
{
    if (field.text == "" || field.text == null)
    {
        return this.preferredHeight;
    }
    else
    {
        this.linesPerCol = Math.ceil(field.numLines / this.numColumns);

        var metrics:TextLineMetrics = field.getLineMetrics(0);
        this.lineHeight = metrics.height;
        var prefHeight:int = linesPerCol * this.lineHeight;

        return prefHeight + 4;
    }
}
```

Сначала метод `getOptimalHeight()` вычисляет ширину каждого столбца. Затем он устанавливает ширину и свойство `htmlText` для первого объекта TextField в массиве. Метод `getOptimalHeight()` использует первый объект TextField для получения общего числа строк с переносом слов в тексте, в соответствии с которым он затем определяет количество строк в каждом столбце. После этого вызывается метод `TextField.getLineMetrics()` для получения объекта TextLineMetrics, содержащего сведения о размере текста в первой строке. Свойство `TextLineMetrics.height` представляет полную высоту строки текста в пикселах, включая надстрочные и подстрочные выносные элементы и междустрочный пробел. Таким образом, оптимальная высота объекта MultiColumnTextField равняется высоте строки, умноженной на количество строк в столбце, плюс 4 пиксела для рамки вверху и внизу объекта TextField, равной двум пикселам.

Ниже приводится код полного метода `layoutColumns()`.

```
public function layoutColumns():void
{
    if (this._text == "" || this._text == null)
    {
        return;
    }

    var field:TextField = fieldArray[0] as TextField;
    field.text = this._text;
    field.setTextFormat(this.format);

    this.preferredHeight = this.getOptimalHeight(field);

    var remainder:String = this._text;
    var fieldText:String = "";
    var lastLineEndedPara:Boolean = true;

    var indent:Number = this.format.indent as Number;

    for (var i:int = 0; i < fieldArray.length; i++)
```

```
{
    field = this.fieldArray[i] as TextField;

    field.height = this.preferredHeight;
    field.text = remainder;

    field.setTextFormat(this.format);

    var lineLen:int;
    if (indent > 0 && !lastLineEndedPara && field.numLines > 0)
    {
        lineLen = field.getLineLength(0);
        if (lineLen > 0)
        {
            field.setTextFormat(this.firstLineFormat, 0, lineLen);
            }
        }

    field.x = i * (colWidth + gutter);
    field.y = 0;

    remainder = "";
    fieldText = "";

    var linesRemaining:int = field.numLines;
    var linesVisible:int = Math.min(this.linesPerCol, linesRemaining);

    for (var j:int = 0; j < linesRemaining; j++)
    {
        if (j < linesVisible)
        {
            fieldText += field.getLineText(j);
        }
        else
        {
            remainder +=field.getLineText(j);
        }
    }

    field.text = fieldText;

    field.setTextFormat(this.format);

    if (indent > 0 && !lastLineEndedPara)
    {
        lineLen = field.getLineLength(0);
        if (lineLen > 0)
        {
            field.setTextFormat(this.firstLineFormat, 0, lineLen);
        }
    }

    var lastLine:String = field.getLineText(field.numLines - 1);
    var lastCharCode:Number = lastLine.charCodeAt(lastLine.length - 1);
```

```
        if (lastCharCode == 10 || lastCharCode == 13)
        {
        lastLineEndedPara = true;
        }
        else
        {
        lastLineEndedPara = false;
        }

        if ((this.format.align == TextFormatAlign.JUSTIFY) &&
                (i < fieldArray.length - 1))
        {
        if (!lastLineEndedPara)
        {
            justifyLastLine(field, lastLine);
        }
    }
    }
}
```

После определения свойства `preferredHeight` с помощью метода `getOptimalHeight()` метод `layoutColumns()` выполняет повторное прохождение через все объекты TextField, устанавливая значение `preferredHeight` в качестве высоты для каждого из них. Затем метод `layoutColumns()` распределяет каждому текстовому полю достаточное количество строк, чтобы в них не требовалась прокрутка, и текст в каждом последующем столбце начинался с того места, на котором закончился предыдущий. Если стилю выравнивания текста задано значение justify, то вызывается метод `justifyLastLine()` для выравнивания последней строки текста в поле. В противном случае последняя строка обрабатывается как последняя строка абзаца и не выравнивается.

# Глава 22. Использование Flash Text Engine

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Инструмент Adobe® Flash® Text Engine (FTE), доступный, начиная с версий Flash Player 10 и Adobe® AIR™ 1.5, обеспечивает поддержку низкого уровня для расширенного управления метриками текста, форматированием и двунаправленным текстом. Он обеспечивает улучшенный процесс обработки текста и расширенную поддержку языков. Хотя возможно применение FTE для создания и управления простыми текстовыми элементами, в первую очередь FTE служит разработчикам в качестве основы для создания компонентов для обработки текстов. Поэтому для использования Flash Text Engine требуется более серьезный опыт в программировании. Сведения об отображении простых текстовых элементов см. в разделе ««Использование класса TextField» на странице 403».

В среде Text Layout Framework, которая включает компонент обработки текста на основе FTE, существует возможность использования дополнительных функций. Text Layout Framework представляет собой обширную библиотеку, полностью встроенную в ActionScript 3.0. Можно использовать как существующий компонент TLF, так и инфраструктуру для создания пользовательского текстового компонента. Дополнительные сведения см. в разделе ««Использование Text Layout Framework» на странице 458».

### Дополнительные разделы справки

Пакет flash.text.engine

## создание и отображение текста

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Классы, включенные во Flash Text Engine, позволяют создавать, форматировать и контролировать текст. Перечисленные ниже классы являются основными элементами для создания и отображения текста с помощью Flash Text Engine:

- TextElement/GraphicElement/GroupElement включают в себя содержимое экземпляра TextBlock
- ElementFormat указывает атрибуты форматирования для содержимого экземпляра TextBlock;
- TextBlock представляет собой основной фабрику для создания абзаца текста;
- TextLine — это строка текста, созданная из экземпляра TextBlock.

Чтобы отобразить текст, создайте объект TextElement на основе переменной String с использованием объекта ElementFormat для задания характеристик форматирования. Назначьте элемент TextElement свойству `content` объекта TextBlock. Создайте строки текста для отображения посредством вызова метода `TextBlock.createTextLine()`. Метод `createTextLine()` возвращает объект TextLine, содержащий ту часть строки, которая соответствует заданной ширине. Циклически вызывайте метод, пока вся строка не будет отформатирована в строках. После создания всех необходимых строк свойству textLineCreationResult объекта TextBlock присваивается значение `TextLineCreationResult.COMPLETE`. Чтобы показать строки, добавьте их в список отображения (с соответствующими значениями расположения `x` и `y`).

Например, в следующем коде используются эти классы FTE для отображения текста «Hello World! This is Flash Text Engine!» (Здравствуй, мир! Это Flash Text Engine!) с использование значений по умолчанию для формата и шрифта. В этом простом примере создается только одна строка текста.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class HelloWorldExample extends Sprite
    {
        public function HelloWorldExample()
        {
            var str = "Hello World! This is Flash Text Engine!";
            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;

            var textLine1:TextLine = textBlock.createTextLine(null, 300);
            addChild(textLine1);
            textLine1.x = 30;
            textLine1.y = 30;
        }
    }
}
```

Параметры для `createTextLine()` указывают строку, с которой должна начинаться новая строка, а также ширину строки в пикселах. Как правило, строкой, с которой должна начинаться новая строка, является предыдущая строка, но в случае с первой строкой используется значение `null`.

## Добавление объектов GraphicElement и GroupElement

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Объекту TextBlock можно присвоить объект GraphicElement, чтобы отобразить изображение или графический элемент. Создайте экземпляр класса GraphicElement, содержащий графику или изображение, и назначьте его свойству `TextBlock.content`. Создайте текстовую строку с помощью метода `TextBlock.createTextline()` стандартным способом. В следующем примере создаются две строки, одна с объектом GraphicElement, а другая с объектом TextElement.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GraphicElementExample extends Sprite
    {
        public function GraphicElementExample()
        {
            var str:String = "Beware of Dog!";

            var triangle:Shape = new Shape();
            triangle.graphics.beginFill(0xFF0000, 1);
            triangle.graphics.lineStyle(3);
            triangle.graphics.moveTo(30, 0);
            triangle.graphics.lineTo(60, 50);
            triangle.graphics.lineTo(0, 50);
            triangle.graphics.lineTo(30, 0);
            triangle.graphics.endFill();

            var format:ElementFormat = new ElementFormat();
            format.fontSize = 20;

            var graphicElement:GraphicElement = new GraphicElement(triangle, triangle.width,
triangle.height, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = graphicElement;
            var textLine1:TextLine = textBlock.createTextLine(null, triangle.width);
            textLine1.x = 50;
            textLine1.y = 110;
            addChild(textLine1);

            var textElement:TextElement = new TextElement(str, format);
            textBlock.content = textElement;
            var textLine2 = textBlock.createTextLine(null, 300);
            addChild(textLine2);
            textLine2.x = textLine1.x - 30;
            textLine2.y = textLine1.y + 15;
        }
    }
}
```

Можно создать объект GroupElement для создания группы объектов TextElement, GraphicElement и других объектов GroupElement. Объект GroupElement можно назначить свойству `content` объекта TextBlock. Параметром конструктора `GroupElement()` является Vector, который указывает на текстовые, графические или групповые элементы, включенные в группу. В следующем примере группируются два графических элемента и один текстовый, и полученная группа присваивается текстовому блоку.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GroupElementExample extends Sprite
    {
        public function GroupElementExample()
        {
            var str:String = "Beware of Alligators!";

            var triangle1:Shape = new Shape();
            triangle1.graphics.beginFill(0xFF0000, 1);
            triangle1.graphics.lineStyle(3);
            triangle1.graphics.moveTo(30, 0);
            triangle1.graphics.lineTo(60, 50);
            triangle1.graphics.lineTo(0, 50);
            triangle1.graphics.lineTo(30, 0);
            triangle1.graphics.endFill();

            var triangle2:Shape = new Shape();
            triangle2.graphics.beginFill(0xFF0000, 1);
            triangle2.graphics.lineStyle(3);
            triangle2.graphics.moveTo(30, 0);
            triangle2.graphics.lineTo(60, 50);
            triangle2.graphics.lineTo(0, 50);
            triangle2.graphics.lineTo(30, 0);
            triangle2.graphics.endFill();

            var format:ElementFormat = new ElementFormat();
            format.fontSize = 20;
            var graphicElement1:GraphicElement = new GraphicElement(triangle1,
triangle1.width, triangle1.height, format);
            var textElement:TextElement = new TextElement(str, format);
            var graphicElement2:GraphicElement = new GraphicElement(triangle2,
triangle2.width, triangle2.height, format);
            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>();
            groupVector.push(graphicElement1, textElement, graphicElement2);
            var groupElement = new GroupElement(groupVector);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = groupElement;
            var textLine:TextLine = textBlock.createTextLine(null, 800);
            addChild(textLine);
            textLine.x = 100;
            textLine.y = 200;
        }
    }
}
```

## Замена текста

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Чтобы заменить текст в экземпляре TextBlock, нужно вызвать метод `TextElement.replaceText()`, который заменит текст в объекте TextElement, назначенном свойству `TextBlock.content`.

В следующем примере используется метод `repaceText()`, который сначала вставляет текст в начало строки, затем добавляет текст в конец строки и, наконец, заменяет текст в середине строки.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class ReplaceTextExample extends Sprite
    {
        public function ReplaceTextExample()
        {

            var str:String = "Lorem ipsum dolor sit amet";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription);
            format.fontSize = 14;
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;
            createLine(textBlock, 10);
            textElement.replaceText(0, 0, "A text fragment: ");
            createLine(textBlock, 30);
            textElement.replaceText(43, 43, "...");
            createLine(textBlock, 50);
            textElement.replaceText(23, 28, "(ipsum)");
            createLine(textBlock, 70);
        }

        function createLine(textBlock:TextBlock, y:Number):void {
            var textLine:TextLine = textBlock.createTextLine(null, 300);
            textLine.x = 10;
            textLine.y = y;
            addChild(textLine);
        }
    }
}
```

Метод `replaceText()` заменяет текст, заданный параметрами `beginIndex` и `endIndex`, текстом, определенном в параметре `newText`. Если значения `beginIndex` и `endIndex` имеют одинаковое значение, метод `replaceText()` вставляет указанный текст в этом местоположении. В противном случае он заменяет символы в диапазоне от `beginIndex` до `endIndex` новым текстом.

# Обработка событий в FTE

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Прослушиватели событий для экземпляра TextLine добавляются так же, как и для других объектов отображения. Например, можно определить, когда пользователь наводит мышь на текстовую строку или щелкает мышью на строке. В следующем примере отслеживаются оба эти события. При наведении мыши на строку курсор меняется на курсор «кнопка», а при щелчке на строке курсор меняет цвет.

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventHandlerExample extends Sprite
    {
        var textBlock:TextBlock = new TextBlock();

        public function EventHandlerExample():void
        {
            var str:String = "I'll change color if you click me.";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription, 18);
            var textElement = new TextElement(str, format);
            textBlock.content = textElement;
            createLine(textBlock);
        }

        private function createLine(textBlock:TextBlock):void
        {
            var textLine:TextLine = textBlock.createTextLine(null, 500);
            textLine.x = 30;
            textLine.y = 30;
            addChild(textLine);
            textLine.addEventListener("mouseOut", mouseOutHandler);
            textLine.addEventListener("mouseOver", mouseOverHandler);
            textLine.addEventListener("click", clickHandler);
        }

        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
            Mouse.cursor = "arrow";
        }

        function clickHandler(event:MouseEvent):void {
            if(textBlock.firstLine)
                removeChild(textBlock.firstLine);
```

```
            var newFormat:ElementFormat = textBlock.content.elementFormat.clone();
            switch(newFormat.color)
            {
                case 0x000000:
                    newFormat.color = 0xFF0000;
                    break;
                case 0xFF0000:
                    newFormat.color = 0x00FF00;
                    break;
                case 0x00FF00:
                    newFormat.color = 0x0000FF;
                    break;
                case 0x0000FF:
                    newFormat.color = 0x000000;
                    break;
            }
            textBlock.content.elementFormat = newFormat;
            createLine(textBlock);
        }
    }
}
```

## Зеркальное отображение событий

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Также события в текстовом блоке или в его части можно дублировать в диспетчере событий. Сначала создайте экземпляр EventDispatcher и назначьте его свойству `eventMirror` экземпляра TextElement. Если текстовый блок состоит из одного текстового элемента, то текстовый сервер дублирует события для всего текстового блока. Если текстовый блок содержит несколько текстовых элементов, то текстовый сервер дублирует события только для тех экземпляров TextElement, для которых задано свойство `eventMirror`. В следующем примере текст содержит три элемента: слово Click («щелкните»), слово here («здесь») и строку to see me in italic («для использования курсива»). Второму текстовому элементу, слову here, назначается диспетчер событий и добавляется прослушиватель событий, метод `clickHandler()`. Метод `clickHandler()` выделяет текст курсивом. Кроме того, он меняет содержимое третьего текстового элемента на строку Click here to see me in normal font! («Нажмите здесь, чтобы сделать форматирование шрифта стандартным!»).

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventMirrorExample extends Sprite
    {
        var fontDescription:FontDescription = new FontDescription("Helvetica", "bold");
        var format:ElementFormat = new ElementFormat(fontDescription, 18);
        var textElement1 = new TextElement("Click ", format);
        var textElement2 = new TextElement("here ", format);
        var textElement3 = new TextElement("to see me in italic! ", format);
        var textBlock:TextBlock = new TextBlock();

        public function EventMirrorExample()
        {
            var myEvent:EventDispatcher = new EventDispatcher();

            myEvent.addEventListener("click", clickHandler);
            myEvent.addEventListener("mouseOut", mouseOutHandler);
            myEvent.addEventListener("mouseOver", mouseOverHandler);

            textElement2.eventMirror=myEvent;

            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>;
            groupVector.push(textElement1, textElement2, textElement3);
            var groupElement:GroupElement = new GroupElement(groupVector);

            textBlock.content = groupElement;
            createLines(textBlock);
        }

        private function clickHandler(event:MouseEvent):void
        {
            var newFont:FontDescription = new FontDescription();
            newFont.fontWeight = "bold";

            var newFormat:ElementFormat = new ElementFormat();
            newFormat.fontSize = 18;
            if(textElement3.text == "to see me in italic! ") {
                newFont.fontPosture = FontPosture.ITALIC;
                textElement3.replaceText(0,21, "to see me in normal font! ");
            }
            else {
                newFont.fontPosture = FontPosture.NORMAL;
                textElement3.replaceText(0, 26, "to see me in italic! ");
            }
            newFormat.fontDescription = newFont;
            textElement1.elementFormat = newFormat;
            textElement2.elementFormat = newFormat;
            textElement3.elementFormat = newFormat;
            createLines(textBlock);
        }
```

```
        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
                Mouse.cursor = "arrow";
        }

        private function createLines(textBlock:TextBlock):void
        {
            if(textBlock.firstLine)
                removeChild (textBlock.firstLine);
            var textLine:TextLine = textBlock.createTextLine (null, 300);
            textLine.x = 15;
            textLine.y = 20;
            addChild (textLine);
        }
    }
}
```

Функции `mouseOverHandler()` и `mouseOutHandler()` меняют курсор «стрелка» на курсор «кнопка», когда он наведен на слово here и возвращают ему прежний вид, когда он покидает область этого элемента.

# Форматирование текста

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Объект `TextBlock` является основой для создания строк текста. Содержимое объекта `TextBlock` задается с помощью объекта `TextElement`. Объект `ElementFormat` обрабатывает форматирование текста. Класс ElementFormat определяет такие свойства, как выравнивание базовой линии, кернинг, трекинг, поворот текста, а также размер, цвет и регистр шрифта. Он также включает класс `FontDescription`, который подробно описывается в главе «Работа со шрифтами» на странице 441.

## Использование объекта ElementFormat

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Конструктор объекта `ElementFormat` принимает любой из многочисленных дополнительных параметров, включая `FontDescription`. Эти свойства можно задать и за пределами конструктора. В следующем примере демонстрируются отношения различных объектов в ходе определения и отображения простой текстовой строки.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function ElementFormatExample()
        {
            fd.fontName = "Garamond";
            ef = new ElementFormat(fd);
            ef.fontSize = 30;
            ef.color = 0xFF0000;
            str = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

## Цвет и прозрачность (альфа) шрифта

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Свойство `color` объекта `ElementFormat` задает цвет шрифта. Это значение является целым числом, представляющим RGB-компоненты цвета; например, 0xFF0000 для красного цвета, а 0x00FF00 — для зеленого. По умолчанию используется значение «черный» (0x000000).

Свойство `alpha` задает значение альфа-прозрачности для элемента (как для `TextElement`, так и для `GraphicElement`). Диапазон допустимых значений: от 0 (полностью прозрачный) до 1 (полностью непрозрачный, используется по умолчанию). Элементы, у которых свойство `alpha` имеет значение 0, являются невидимыми, но активными. Это значение умножается на унаследованные альфа-значения, в результате чего элемент становится более прозрачным.

```
var ef:ElementFormat = new ElementFormat();
ef.alpha = 0.8;
ef.color = 0x999999;
```

## Выравнивание и смещение базовой линии

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Доминантная базовая линия определяется по шрифту и размеру самого крупного текста в строке. Эти значения можно переопределить, задав свойства `TextBlock.baselineFontDescription` и `TextBlock.baselineFontSize`. Доминантную базовую линию можно выровнять по одной из нескольких базовых линий текста. К таким базовым линиям относятся линии верхних и нижних выносных элементов или идеографические верхняя, центральная и нижняя линии.

***А.** Линия верхних выносных элементов **Б.** Базовая линия **В.** Линия нижних выносных элементов **Г.** Высота x*

В объекте `ElementFormat` базовую линию и характеристики выравнивания определяют три свойства. Свойство `alignmentBaseline` задает основную базовую линию объекта `TextElement` или `GraphicElement`. Это линия, к которой «привязывается» элемент. Именно по этой линии выравнивается доминантная базовая линия всего текста.

Свойство `dominantBaseline` определяет, какая из нескольких базовых линий элемента должна использоваться для определения вертикальной позиции элемента в строке. По умолчанию используется значение `TextBaseline.ROMAN`, но доминантной базовой линией можно также назначить линию `IDEOGRAPHIC_TOP` или `IDEOGRAPHIC_BOTTOM`.

Свойство `baselineShift` перемещает базовую линию на заданное число пикселов по оси Y. В стандартном тексте (без поворота) положительное значение перемещает базовую линию вниз, а отрицательное — вверх.

## Типографский регистр

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Свойство `TypographicCase` объекта `ElementFormat` указывает регистр текста, например верхний, нижний регистр или капитель.

```
var ef_Upper:ElementFormat = new ElementFormat();
ef_Upper.typographicCase = TypographicCase.UPPERCASE;

var ef_SmallCaps:ElementFormat = new ElementFormat();
ef_SmallCaps.typographicCase = TypographicCase.SMALL_CAPS;
```

## Поворот текста

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Блок текста или глифы в сегменте текста можно поворачивать с шагом 90°. Класс TextRotation определяет следующие константы для настройки поворота текстового блока или глифа:

| Константа | Значение | Описание |
|---|---|---|
| AUTO | auto | Задает поворот на 90 градусов против часовой стрелки. Обычно используется с вертикальным восточным текстом для поворота отдельных глифов. |
| ROTATE_0 | rotate_0 | Задает отсутствие поворота. |
| ROTATE_180 | rotate_180 | Задает поворот на 180 градусов. |
| ROTATE_270 | rotate_270 | Задает поворот на 270 градусов. |
| ROTATE_90 | rotate_90 | Задает поворот на 90 градусов по часовой стрелке. |

Для поворота строк в текстовом блоке необходимо установить свойство `TextBlock.lineRotation` перед вызовом метода `TextBlock.createTextLine()` для создания строки текста.

Для поворота глифов в блоке или сегменте текста требуется установить свойство `ElementFormat.textRotation` на требуемое число градусов для поворота глифов. Глиф — это фигура, образующая символ, или часть символа, состоящего из нескольких глифов. Например, глифами являются буква «a» и точка над символом «i».

Поворот глифов применяется в некоторых восточных языках, в которых требуется сделать горизонтальные строки вертикальными, не поворачивая символы в строках. Дополнительные сведения о повороте восточного текста см. в разделе «Выравнивание восточно-азиатского текста» на странице 445.

Ниже приводится пример поворота блока текста и содержащихся в нем глифов для восточных языков. В примере также используется японский шрифт.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class RotationExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function RotationExample()
        {
            fd.fontName = "MS Mincho";
            ef = new ElementFormat(fd);
            ef.textRotation = TextRotation.AUTO;
            str = "This is rotated Japanese text";
            te = new TextElement(str, ef);
            tb.lineRotation = TextRotation.ROTATE_90;
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

## Блокировка и клонирование ElementFormat

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

При назначении объекта `ElementFormat` любому типу `ContentElement` его свойство `locked` автоматически получает значение `true`. При попытке изменить заблокированный объект `ElementFormat` возвращается ошибка `IllegalOperationError`. Лучше всего дать полное определение такому объекту, прежде чем назначать его экземпляру `TextElement`.

Если требуется изменить существующий экземпляр `ElementFormat`, сначала нужно проверить значение свойства `locked`. Если оно имеет значение `true`, используйте метод `clone()` для создания незаблокированной копии объекта. Свойства такого незаблокированного объекта можно изменить, и после этого назначить его экземпляру `TextElement`. Новые строки, созданные из этого объекта, будут иметь новое форматирование. Предыдущие строки, созданные из этого же объекта с использованием старого формата, остаются без изменений.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd:FontDescription = new FontDescription();

        public function ElementFormatCloneExample()
        {
            fd.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            ef1.fontSize = 24;
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontSize = 32;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

# Работа со шрифтами

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Объект `FontDescription` используется вместе с объектом `ElementFormat` для определения начертания шрифта и настройки его характеристик. В число характеристик входит имя шрифта, его толщина, наклон, визуализация и местоположение шрифта (в устройстве или в SWF-файле).

***Примечание.*** *FTE не поддерживает шрифты Type 1 и точечные шрифты, например, Type 3, ATC, sfnt-wrapped CID или Naked CID.*

## Определение характеристик шрифта (объект FontDescription)

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Свойство `fontName` объекта `FontDescription` может содержать одно имя или список имен, разделенных запятыми. Например, в списке Arial, Helvetica, _sans механизм обработки текста сначала ищет Arial, затем Helvetica, а после этого _sans, если не удалось найти первые два шрифта. Набор имен шрифтов включает три имени общих шрифтов устройства: _sans, _serif и _typewriter. Они соотносятся с определенными шрифтами устройства в зависимости от системы воспроизведения. Полезно указывать имена по умолчанию, подобные этим, во всех описаниях шрифтов, использующих шрифты устройства. Если свойство `fontName` не определено, по умолчанию используется _serif.

Свойство `fontPosture` можно установить на значение по умолчанию (`FontPosture.NORMAL`) или на значение «курсивное начертание» (`FontPosture.ITALIC`). Свойство `fontWeight` можно Установить на значение по умолчанию (`FontWeight.NORMAL`) или на значение «полужирное начертание» (`FontWeight.BOLD`).

```
var fd1:FontDescription = new FontDescription();
fd1.fontName = "Arial, Helvetica, _sans";
fd1.fontPosture = FontPosture.NORMAL;
fd1.fontWeight = FontWeight.BOLD;
```

## Встроенные шрифты и шрифты устройства

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Свойство `fontLookup` объекта `FontDescription` указывает, какой шрифт должен искать механизм обработки текста для визуализации текста: шрифт устройства или встроенный шрифт. Если указан шрифт устройства (`FontLookup.DEVICE`), среда выполнения ищет шрифт в системе воспроизведения. Если указан встроенный шрифт (`FontLookup.EMBEDDED_CFF`), среда выполнения ищет встроенный шрифт с указанным именем в SWF-файле. Эта настройка поддерживает только компактные встроенные шрифты CFF (Compact Font Format). Если указанный шрифт не найден, используется шрифт устройства по умолчанию.

Использование шрифтов устройства позволяет уменьшить размер SWF-файла. Встроенные шрифты обеспечивают правильное отображение на разных платформах.

```
var fd1:FontDescription = new FontDescription();
fd1.fontLookup = FontLookup.EMBEDDED_CFF;
fd1.fontName = "Garamond, _serif";
```

## Режим визуализации и доводка до пикселов

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Во Flash Player 10 и Adobe AIR 1.5 добавлена визуализация CFF (компактные встроенные шрифты). При использовании данного типа визуализации экранные шрифты становятся более разборчивыми, а отображение текста, написанного мелким шрифтом, выполняется более качественно. Эта настройка применяется только к встроенным шрифтам. В объекте `FontDescription` это значение (`RenderingMode.CFF`) используется по умолчанию для свойства `renderingMode`. Можно установить свойство на `RenderingMode.NORMAL` для выбора типа визуализации, применяемого во Flash Player 7 и более ранних версиях.

Если выбрана CFF-визуализация, второе свойство, `cffHinting`, управляет привязкой горизонтальных линий шрифта к подпикселной сетке. По умолчанию используется значение `CFFHinting.HORIZONTAL_STEM`, которое включает привязку к пикселам. Если задать этому свойству значение `CFFHinting.NONE`, доводка отключается. Эта настройка подходит для анимации и крупного шрифта.

```
var fd1:FontDescription = new FontDescription();
fd1.renderingMode = RenderingMode.CFF;
fd1.cffHinting = CFFHinting.HORIZONTAL_STEM;
```

## Блокировка и клонирование объектов FontDescription

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Если объект `FontDescription` назначается объекту `ElementFormat`, его свойство `locked` автоматически получает значение `true`. При попытке изменить заблокированный объект`FontDescription` выводится ошибка `IllegalOperationError`. Лучше всего дать полное определение такому объекту, прежде чем назначать его экземпляру `ElementFormat`.

Если требуется изменить существующий экземпляр `FontDescription`, сначала нужно проверить значение свойства `locked`. Если оно имеет значение `true`, используйте метод `clone()` для создания незаблокированной копии объекта. Свойства такого объекта можно изменить, а затем назначить его экземпляру `ElementFormat`. Новые строки, созданные из этого объекта `TextElement`, будут иметь новое форматирование. Предыдущие строки, созданные из этого объекта ранее, останутся без изменений.

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class FontDescriptionCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd1:FontDescription = new FontDescription();
        private var fd2:FontDescription;

        public function FontDescriptionCloneExample()
        {
            fd1.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            fd2 = (fd1.locked) ? fd1.clone() : fd1;
            fd2.fontName = "Arial";
            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontDescription = fd2;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

# Управление текстом

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

FTE предоставляет новые возможности управления форматированием текста для настройки выравнивания и интервала между знаками (кернинга и трекинга). Кроме того, предусмотрены свойства для управления разбивкой строк и для установки позиций табуляции в строках.

## Выравнивание текста

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Выравнивание текста делает длину всех строк абзаца одинаковой путем изменения интервалов между словами и иногда между буквами. Цель заключается в том, чтобы выровнять текст по краям, а расстояние между словами и буквами может быть разным. Нередко выравнивание столбцов текста используют в газетах и журналах.

Свойство `lineJustfication` класса SpaceJustifier позволяет управлять выравниванием строк в блоке текста. Класс LineJustification определяет константы, которые можно использовать при выборе типа выравнивания: `ALL_BUT_LAST` выравнивает все строки текста, кроме последней; `ALL_INCLUDING_LAST` выравнивает все строки текста, включая последнюю; `UNJUSTIFIED` (значение по умолчанию) не применяет выравнивание.

Для выравнивания текста следует установить свойство `lineJustification` на экземпляр класса SpaceJustifier и назначить этот экземпляр свойству `textJustifier` экземпляра TextBlock. В следующем примере создается абзац, в котором выравниваются все строки текста кроме последней.

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class JustifyExample extends Sprite
    {
        public function JustifyExample()
        {
            var str:String = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, " +
            "sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut " +
            "enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut " +
            "aliquip ex ea commodo consequat.";

            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement=new TextElement(str,format);
            var spaceJustifier:SpaceJustifier=new
SpaceJustifier("en",LineJustification.ALL_BUT_LAST);

            var textBlock:TextBlock = new TextBlock();
            textBlock.content=textElement;
            textBlock.textJustifier=spaceJustifier;
            createLines(textBlock);
        }

        private function createLines(textBlock:TextBlock):void {
            var yPos=20;
            var textLine:TextLine=textBlock.createTextLine(null,150);

            while (textLine) {
                addChild(textLine);
                textLine.x=15;
                yPos+=textLine.textHeight+2;
                textLine.y=yPos;
                textLine=textBlock.createTextLine(textLine,150);
            }
        }
    }
}
```

Чтобы варьировать интервалы не только между словами, но и между буквами, следует установить свойство `SpaceJustifier.letterspacing` на значение `true`. Включение настройки межбуквенного интервала может сократить число чрезмерно больших пробелов между словами, которые иногда могут появляться при простом выравнивании.

## Выравнивание восточно-азиатского текста

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

При выравнивании восточно-азиатского текста учитываются дополнительные характеристики. Он может писаться сверху вниз, и некоторые символы, называемые «кинсоку», не могут быть отображены в начале или конце строки. Класс JustificationStyle определяет следующие константы, предназначенные для работы с этими символами. Константа `PRIORITIZE_LEAST_ADJUSTMENT` выполняет выравнивание на основе расширения или сжатия строки в зависимости от того, что дает оптимальный результат. Константа `PUSH_IN_KINSOKU`выполняет выравнивание на основе сжатия кинсоку в конце строки или расширения при отсутствии кинсоку или если недостаточно места.

Константа `PUSH_OUT_ONLY` выполняет выравнивание на основе расширения строки. Для создания вертикального восточно-азиатского текста следует установить свойство `TextBlock.lineRotation` на значение `TextRotation.ROTATE_90`, а свойство `ElementFormat.textRotation` — на значение `TextRotation.AUTO`, используемое по умолчанию. Если свойство `textRotation` имеет значение `AUTO`, при повороте строки глифы в тексте остаются в вертикальном положении. Настройка `AUTO` поворачивает на 90° против часовой стрелки только полноширинные и широкие глифы, что определяется свойствами глифа в кодировке Юникод. В следующем примере отображается вертикальный блок японского текста, который выравнивается с использованием варианта `PUSH_IN_KINSOKU`.

```
package
{
    import flash.text.engine.*;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.system.Capabilities;

    public class EastAsianJustifyExample extends Sprite
    {
        public function EastAsianJustifyExample()
        {
            var Japanese_txt:String = String.fromCharCode(
            0x5185, 0x95A3, 0x5E9C, 0x304C, 0x300C, 0x653F, 0x5E9C, 0x30A4,
            0x30F3, 0x30BF, 0x30FC, 0x30CD, 0x30C3, 0x30C8, 0x30C6, 0x30EC,
            0x30D3, 0x300D, 0x306E, 0x52D5, 0x753B, 0x914D, 0x4FE1, 0x5411,
            0x3051, 0x306B, 0x30A2, 0x30C9, 0x30D3, 0x30B7, 0x30B9, 0x30C6,
            0x30E0, 0x30BA, 0x793E, 0x306E)
            var textBlock:TextBlock = new TextBlock();
            var font:FontDescription = new FontDescription();
            var format:ElementFormat = new ElementFormat();
            format.fontSize = 12;
            format.color = 0xCC0000;
            format.textRotation = TextRotation.AUTO;
            textBlock.baselineZero = TextBaseline.IDEOGRAPHIC_CENTER;
            var eastAsianJustifier:EastAsianJustifier = new EastAsianJustifier("ja",
LineJustification.ALL_BUT_LAST);
            eastAsianJustifier.justificationStyle = JustificationStyle.PUSH_IN_KINSOKU;
            textBlock.textJustifier = eastAsianJustifier;
            textBlock.lineRotation = TextRotation.ROTATE_90;
            var linePosition:Number = this.stage.stageWidth - 75;
            if (Capabilities.os.search("Mac OS") > -1)
                // set fontName: Kozuka Mincho Pro R
```

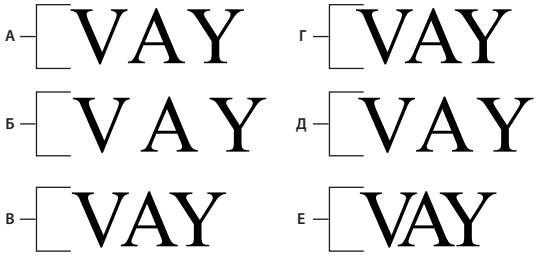
```
            font.fontName = String.fromCharCode(0x5C0F, 0x585A, 0x660E, 0x671D) + " Pro R";
        else
            font.fontName = "Kozuka Mincho Pro R";
        textBlock.content = new TextElement(Japanese_txt, format);
        var previousLine:TextLine = null;

        while (true)
        {
            var textLine:TextLine = textBlock.createTextLine(previousLine, 200);
            if (textLine == null)
                break;
            textLine.y = 20;
            textLine.x = linePosition;
            linePosition -= 25;
            addChild(textLine);
            previousLine = textLine;
        }
    }
  }
}
```

## Кернинг и трекинг

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Кернинг и трекинг влияют на расстояние между двумя соседними символами в блоке текста. Кернинг регулирует совместное написание пары символов, например WA или Va. Кернинг устанавливается в объекте `ElementFormat`. По умолчанию кернинг включен (`Kerning.ON`); он может быть также установлен на OFF или AUTO. В этом случае кернинг применяется, только если ни один из символов не является символом кандзи, хирагана или катакана.

Трекинг увеличивает или уменьшает заданное число пикселов между всеми символами в текстовом блоке. Он также задается в объекте `ElementFormat`. Трекинг применяется как ко встроенным шрифтам, так и к шрифтам устройства. FTE поддерживает два свойства трекинга: `trackingLeft`, которое увеличивает или уменьшает число пикселов слева от символа, и `trackingRight`, которое увеличивает или уменьшает число пикселов справа от символа. Если используется кернинг, значение трекинга прибавляется или отнимается от значений кернинга для данной пары символов.




***А.*** *Kerning.OFF* ***Б.*** *TrackingRight=5, Kerning.OFF* ***В.*** *TrackingRight=-5, Kerning.OFF* ***Г.*** *Kerning.ON* ***Д.*** *TrackingRight=-5, Kerning.ON* ***Е.*** *TrackingRight=-5, Kerning.ON*

```
var ef1:ElementFormat = new ElementFormat();
ef1.kerning = Kerning.OFF;

var ef2:ElementFormat = new ElementFormat();
ef2.kerning = Kerning.ON;
ef2.trackingLeft = 0.8;
ef2.trackingRight = 0.8;

var ef3:ElementFormat = new ElementFormat();
ef3.trackingRight = -0.2;
```

## Разрывы строки для переноса текста

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Свойство `breakOpportunity` объекта `ElementFormat` определяет, какие символы могут использоваться для разрыва строки, когда текст разбивается на несколько строк. По умолчанию используется значение `BreakOpportunity.AUTO`, которое использует стандартные свойства кодировки «Юникод», такие как разрыв строки на стыке слов или после дефиса. Значение `BreakOpportunity.ALL` позволяет использовать любой символ как возможность разрыва строки. Этот вариант подходит для создания таких эффектов, как текст вдоль кривой.

```
var ef:ElementFormat = new ElementFormat();
ef.breakOpportunity = BreakOpportunity.ALL;
```

## Позиции табуляции

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Для установки позиций табуляции в текстовом блоке следует определить их путем создания экземпляров класса TabStop. Параметры конструктора `TabStop()` определяют, как текст выравнивается по позиции табуляции. Эти параметры задают местоположение позиции табуляции, а при десятичном выравнивании они указывают для выравнивания строковое значение. Как правило, это значение представлено десятичной точкой, однако также можно использовать запятую, символ доллара, йены или евро. Следующий код создает позицию табуляции с именем tab1.

```
var tab1:TabStop = new TabStop(TabAlignment.DECIMAL, 50, ".");
```

Создав позиции табуляции для блока текста, их нужно назначить свойству `tabStops` экземпляра TextBlock. Так как свойство `tabStops` требует наличия объекта Vector, сначала необходимо создать Vector и добавить в него позиции табуляции. Vector позволяет назначить набор позиций табуляции блоку текста. В следующем примере создается экземпляр `Vector<TabStop>`, в который затем добавляются объекты TabStop. После этого позиции табуляции назначаются свойству `tabStops` экземпляра TextBlock.

```
var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
tabStops.push(tab1, tab2, tab3, tab4);
textBlock.tabStops = tabStops
```

Дополнительные сведения о классе Vector см. в главе «Работа с массивами» на странице 26.

В следующем примере демонстрируется эффект каждого из вариантов выравнивания TabStop.

```
package {

    import flash.text.engine.*;
    import flash.display.Sprite;

    public class TabStopExample extends Sprite
    {
        public function TabStopExample()
        {
            var format:ElementFormat = new ElementFormat();
            format.fontDescription = new FontDescription("Arial");
            format.fontSize = 16;

            var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
            tabStops.push(
                new TabStop(TabAlignment.START, 20),
                new TabStop(TabAlignment.CENTER, 140),
                new TabStop(TabAlignment.DECIMAL, 260, "."),
                new TabStop(TabAlignment.END, 380));
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = new TextElement(
                "\tt1\tt2\tt3\tt4\n" +
                "\tThis line aligns on 1st tab\n" +
                "\t\t\t\tThis is the end\n" +
                "\tThe following fragment centers on the 2nd tab:\t\t\n" +
                "\t\tit's on me\t\t\n" +
                "\tThe following amounts align on the decimal point:\n" +
                "\t\t\t45.00\t\n" +
                "\t\t\t75,320.00\t\n" +
                "\t\t\t6,950.00\t\n" +
                "\t\t\t7.01\t\n", format);

            textBlock.tabStops = tabStops;
            var yPosition:Number = 60;
            var previousTextLine:TextLine = null;
            var textLine:TextLine;
            var i:int;
            for (i = 0; i < 10; i++) {
                textLine = textBlock.createTextLine(previousTextLine, 1000, 0);
                textLine.x = 20;
                textLine.y = yPosition;
                addChild(textLine);
                yPosition += 25;
                previousTextLine = textLine;
            }
        }
    }
}
```

# Пример Flash Text Engine: макет новостей

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Этот пример программы демонстрирует использование класса Flash Text Engine для верстки простой газетной страницы. Страница содержит большой заголовок, подзаголовок и основой раздел из нескольких колонок.

Во-первых, необходимо создать FLA-файл и присоединить следующий код к кадру 2 слоя по умолчанию:

```
import com.example.programmingas3.newslayout.StoryLayout ;
// frame sc ript - create  a 3-columned arti cle layout
var story:StoryLayout = new StoryLayout(720, 500, 3, 10);
story.x = 20;
story.y = 80;
addChild(story);
stop();
```

StoryLayout.as является управляющим сценарием для данного примера. Он устанавливает содержимое, считывает информацию о стиле из внешней таблицы стилей и назначает эти стили объектам ElementFormat. Затем он создает элемент заголовка, подзаголовка и текста, состоящего из нескольких колонок.

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.StyleSheet;
    import flash.text.engine.*;

    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.net.URLLoader;
    import flash.display.Sprite;
    import flash.display.Graphics;

    public class StoryLayout extends Sprite
    {
        public var headlineTxt:HeadlineTextField;
        public var subtitleTxt:HeadlineTextField;
        public var storyTxt:MultiColumnText;
        public var sheet:StyleSheet;
        public var h1_ElFormat:ElementFormat;
        public var h2_ElFormat:ElementFormat;
        public var p_ElFormat:ElementFormat;

        private var loader:URLLoader;

        public var paddingLeft:Number;
        public var paddingRight:Number;
        public var paddingTop:Number;
        public var paddingBottom:Number;

        public var preferredWidth:Number;
        public var preferredHeight:Number;

        public var numColumns:int;

        public var bgColor:Number = 0xFFFFFF;
```

```
        public var headline:String = "News Layout Example";
        public var subtitle:String = "This example formats text like a newspaper page using the
Flash Text Engine API. ";

        public var rawTestData:String =
        "From the part Mr. Burke took in the American Revolution, it was natural that I should
consider him a friend to mankind; and as our acquaintance commenced on that ground, it would
have been more agreeable to me to have had cause to continue in that opinion than to change it. " +
        "At the time Mr. Burke made his violent speech last winter in the English Parliament
against the French Revolution and the National Assembly, I was in Paris, and had written to him
but a short time before to inform him how prosperously matters were going on. Soon after this I
saw his advertisement of the Pamphlet he intended to publish: As the attack was to be made in a
language but little studied, and less understood in France, and as everything suffers by
translation, I promised some of the friends of the Revolution in that country that whenever Mr.
Burke's Pamphlet came forth, I would answer it. This appeared to me the more necessary to be
done, when I saw the flagrant misrepresentations which Mr. Burke's Pamphlet contains; and that
while it is an outrageous abuse on the French Revolution, and the principles of Liberty, it is
an imposition on the rest of the world. " +
        "I am the more astonished and disappointed at this conduct in Mr. Burke, as (from the
circumstances I am going to mention) I had formed other expectations. " +
        "I had seen enough of the miseries of war, to wish it might never more have existence
in the world, and that some other mode might be found out to settle the differences that should
occasionally arise in the neighbourhood of nations. This certainly might be done if Courts were
disposed to set honesty about it, or if countries were enlightened enough not to be made the
dupes of Courts. The people of America had been bred up in the same prejudices against France,
which at that time characterised the people of England; but experience and an acquaintance with
the French Nation have most effectually shown to the Americans the falsehood of those prejudices;
and I do not believe that a more cordial and confidential intercourse exists between any two
countries than between America and France. ";

        public function StoryLayout(w:int = 400, h:int = 200, cols:int = 3, padding:int =
10):void
        {
            this.preferredWidth = w;
            this.preferredHeight = h;

            this.numColumns = cols;

            this.paddingLeft = padding;
            this.paddingRight = padding;
            this.paddingTop = padding;
            this.paddingBottom = padding;

            var req:URLRequest = new URLRequest("story.css");
            loader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
            loader.load(req);
        }

        public function onCSSFileLoaded(event:Event):void
        {
            this.sheet = new StyleSheet();
            this.sheet.parseCSS(loader.data);

            // convert headline styles to ElementFormat objects
            h1_ElFormat = getElFormat("h1", this.sheet);
```

```
    h1_ElFormat.typographicCase = TypographicCase.UPPERCASE;
    h2_ElFormat = getElFormat("h2", this.sheet);
    p_ElFormat = getElFormat("p", this.sheet);
    displayText();
}

public function drawBackground():void
{
    var h:Number = this.storyTxt.y + this.storyTxt.height +
                    this.paddingTop + this.paddingBottom;
    var g:Graphics = this.graphics;
    g.beginFill(this.bgColor);
    g.drawRect(0, 0, this.width + this.paddingRight + this.paddingLeft, h);
    g.endFill();
}

/**
 * Reads a set of style properties for a named style and then creates
 * a TextFormat object that uses the same properties.
 */
public function getElFormat(styleName:String, ss:StyleSheet):ElementFormat
{
var style:Object = ss.getStyle(styleName);
if (style != null)
{
var colorStr:String = style.color;
if (colorStr != null && colorStr.indexOf("#") == 0)
{
    style.color = colorStr.substr(1);
}
        var fd:FontDescription = new FontDescription(
                              style.fontFamily,
                              style.fontWeight,
                              FontPosture.NORMAL,
                              FontLookup.DEVICE,
                              RenderingMode.NORMAL,
                              CFFHinting.NONE);
var format:ElementFormat = new ElementFormat(fd,
                      style.fontSize,
                      style.color,
                      1,
                      TextRotation.AUTO,
                      TextBaseline.ROMAN,
                      TextBaseline.USE_DOMINANT_BASELINE,
                      0.0,
                      Kerning.ON,
                      0.0,
                      0.0,
                      "en",
                              BreakOpportunity.AUTO,
                              DigitCase.DEFAULT,
                              DigitWidth.DEFAULT,
                      LigatureLevel.NONE,
                              TypographicCase.DEFAULT);

if (style.hasOwnProperty("letterSpacing"))
{
```

```
            format.trackingRight = style.letterSpacing;
        }
        }
        return format;
        }

        public function displayText():void
        {
        headlineTxt = new HeadlineTextField(h1_ElFormat,headline,this.preferredWidth);
        headlineTxt.x = this.paddingLeft;
        headlineTxt.y = 40 + this.paddingTop;
            headlineTxt.fitText(1);
            this.addChild(headlineTxt);

        subtitleTxt = new HeadlineTextField(h2_ElFormat,subtitle,this.preferredWidth);
        subtitleTxt.x = this.paddingLeft;
        subtitleTxt.y = headlineTxt.y + headlineTxt.height;
            subtitleTxt.fitText(2);
        this.addChild(subtitleTxt);

        storyTxt = new MultiColumnText(rawTestData, this.numColumns,
                        20, this.preferredWidth, this.preferredHeight, p_ElFormat);
        storyTxt.x = this.paddingLeft;
        storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
        this.addChild(storyTxt);

            drawBackground();
        }
    }
}
```

FormattedTextBlock.as используется в качестве базового класса для создания блоков текста. Он также включает функции для изменения размера и регистра шрифта.

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class FormattedTextBlock extends Sprite
    {
        public var tb:TextBlock;
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var textWidth:int;
        public var totalTextLines:int;
        public var blockText:String;
        public var leading:Number = 1.25;
        public var preferredWidth:Number = 720;
        public var preferredHeight:Number = 100;

        public function FormattedTextBlock(ef:ElementFormat,txt:String, colW:int = 0)
        {
            this.textWidth = (colW==0) ? preferredWidth : colW;
            blockText = txt;
            ef1 = ef;
            tb = new TextBlock();
```

```
        tb.textJustifier = new SpaceJustifier("en",LineJustification.UNJUSTIFIED,false);
        te = new TextElement(blockText,this.ef1);
        tb.content = te;
        this.breakLines();
    }

    private function breakLines()
    {
        var textLine:TextLine = null;
        var y:Number = 0;
        var lineNum:int = 0;
        while (textLine = tb.createTextLine(textLine,this.textWidth,0,true))
        {
            textLine.x = 0;
            textLine.y = y;
            y += this.leading*textLine.height;
            this.addChild(textLine);
        }
        for (var i:int = 0; i < this.numChildren; i++)
        {
            TextLine(this.getChildAt(i)).validity = TextLineValidity.STATIC;
        }
        this.totalTextLines = this.numChildren;
    }

    private function rebreakLines()
    {
        this.clearLines();
        this.breakLines();
    }

    private function clearLines()
    {
        while(this.numChildren)
        {
            this.removeChildAt(0);
        }
    }
```

```
        public function changeSize(size:uint=12):void
        {
            if (size > 5)
            {
                var ef2:ElementFormat = ef1.clone();
                ef2.fontSize = size;
                te.elementFormat = ef2;
                this.rebreakLines();
            }
        }

        public function changeCase(newCase:String = "default"):void
        {
            var ef2:ElementFormat = ef1.clone();
            ef2.typographicCase = newCase;
            te.elementFormat = ef2;
        }
    }
}
```

HeadlineTextBlock.as расширяет класс FormattedTextBlock и используется для создания заголовков. Он включает функцию для фильтрации текста в пределах определенного пространства на странице.

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    public class HeadlineTextField extends FormattedTextBlock
    {

        public static var MIN_POINT_SIZE:uint = 6;
        public static var MAX_POINT_SIZE:uint = 128;

        public function HeadlineTextField(te:ElementFormat,txt:String,colW:int = 0)
        {
            super(te,txt);
        }

        public function fitText(maxLines:uint = 1, targetWidth:Number = -1):uint
        {
            if (targetWidth == -1)
            {
                targetWidth = this.width;
            }

            var pixelsPerChar:Number = targetWidth / this.blockText.length;
            var pointSize:Number = Math.min(MAX_POINT_SIZE,
                            Math.round(pixelsPerChar * 1.8 * maxLines));

            if (pointSize < 6)
            {
                // the point size is too small
                return pointSize;
            }

            this.changeSize(pointSize);
            if (this.totalTextLines > maxLines)
```

```
            {
                return shrinkText(--pointSize, maxLines);
            }
            else
            {
                return growText(pointSize, maxLines);
            }
        }

        public function growText(pointSize:Number, maxLines:uint = 1):Number
        {
            if (pointSize >= MAX_POINT_SIZE)
            {
                return pointSize;
            }

            this.changeSize(pointSize + 1);
            if (this.totalTextLines > maxLines)
            {
                // set it back to the last size
                this.changeSize(pointSize);
                return pointSize;
            }
            else
            {
                return growText(pointSize + 1, maxLines);
            }
        }

        public function shrinkText(pointSize:Number, maxLines:uint=1):Number
        {
            if (pointSize <= MIN_POINT_SIZE)
            {
                return pointSize;
            }
            this.changeSize(pointSize);

            if (this.totalTextLines > maxLines)
            {
                return shrinkText(pointSize - 1, maxLines);
            }
            else
            {
                return pointSize;
            }
        }
    }
}
```

MultiColumnText.as обрабатывает форматирование текста в представлении, состоящем из нескольких колонок. Он демонстрирует гибкое использование объекта TextBlock в качестве фабрики для создания, форматирования и размещения строк текста.

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.engine.*;

    public class MultiColumnText extends Sprite
    {
        private var tb:TextBlock;
        private var te:TextElement;
        private var numColumns:uint = 2;
        private var gutter:uint = 10;
        private var leading:Number = 1.25;
        private var preferredWidth:Number = 400;
        private var preferredHeight:Number = 100;
        private var colWidth:int = 200;

        public function MultiColumnText(txt:String = "",cols:uint = 2,
            gutter:uint = 10, w:Number = 400, h:Number = 100,
            ef:ElementFormat = null):void
        {
            this.numColumns = Math.max(1, cols);
            this.gutter = Math.max(1, gutter);

            this.preferredWidth = w;
            this.preferredHeight = h;

            this.setColumnWidth();

            var field:FormattedTextBlock = new FormattedTextBlock(ef,txt,this.colWidth);
            var totLines:int = field.totalTextLines;
            field = null;
            var linesPerCol:int = Math.ceil(totLines/cols);

            tb = new TextBlock();
            te = new TextElement(txt,ef);
            tb.content = te;
            var textLine:TextLine = null;
            var x:Number = 0;
            var y:Number = 0;
            var i:int = 0;
            var j:int = 0;
            while (textLine = tb.createTextLine(textLine,this.colWidth,0,true))
            {
                textLine.x = Math.floor(i/(linesPerCol+1))*(this.colWidth+this.gutter);
                textLine.y = y;
                y += this.leading*textLine.height;
```

```
                j++;
                if(j>linesPerCol)
                {
                    y = 0;
                    j = 0;
                }
                i++;

                this.addChild(textLine);
            }
        }

        private function setColumnWidth():void
        {
        this.colWidth = Math.floor( (this.preferredWidth -
            ((this.numColumns - 1) * this.gutter)) / this.numColumns);
        }

    }
}
```

# Глава 23. Использование Text Layout Framework

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

## Обзор Text Layout Framework

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Text Layout Framework (TLF) — это расширяемая библиотека ActionScript. Среда TLF создана на базе механизма обработки текста, реализованного в Adobe® Flash® Player 10 и Adobe® AIR® 1.5. TLF предоставляет расширенные оформления и размещения текста для разработки инновационных веб-решений. Среда может быть использована в Adobe® Flex® или Adobe® Flash® Professional. Разработчики могут пользоваться существующими компонентами или расширять их, а также создавать собственные текстовые компоненты в среде.

TLF предоставляет следующие возможности:

- Двунаправленный текст, текст по вертикали и более 30 систем письма, включая арабскую, иврит, китайскую, японскую, корейскую, тайскую, лаосскую, вьетнамскую и другие.
- Выделение, редактирование и расположение текста в нескольких столбцах и связанных контейнерах.
- Текст по вертикали, элементы Tate-Chu-Yoko (текст по горизонтали и вертикали) и выравнивание для восточноазиатского оформления.
- Широкий набор функций управления оформлением, включая кернинг, лигатуры, типографский регистр, цифровой регистр, цифровая ширина и возможные переносы.
- Вырезание, копирование, вставка, отмена и стандартные жесты мышью и комбинации клавиш для редактирования.
- Многофункциональный API-интерфейс разработчика для манипуляции текстовым содержимым, расположением и разметкой, а также для создания текстовых компонентов.
- Поддержка списков, включая пользовательские маркеры и форматы нумерации
- Встроенные изображения и правила позиционирования

TLF — это библиотека ActionScript 3.0, созданная на основе механизма обработки текста Flash Text Engine (FTE), представленного в инструменте Flash Player 10. Можно получить доступ к FTE с помощью пакета `flash.text.engine`, являющегося компонентом прикладного программного интерфейса (Application Programming Interface, API) Flash Player 10.

Однако доступ к этому механизму обработки текста, предоставляемый API-интерфейсом Flash Player, осуществляется на низком уровне. Это означает, что для выполнения некоторых задач требуется написание большого кода. Среда TLF добавляет код нижнего уровня в более простые API-интерфейсы. TLF также предоставляет концептуальную архитектуру, которая организует базовые структурные блоки, определенные в FTE, в более простую для использования систему.

В отличие от FTE, компонент TLF не встроен в приложение Flash Player. Он представляет собой независимую библиотеку компонентов, полностью созданную на основе кода ActionScript 3.0. Поскольку среда может расширяться, его можно настроить для определенных сред. Инструменты Flash Professional и Flex SDK включают компоненты, созданные на основе среды TLF.

### Дополнительные разделы справки

Приложение разметки TLF «потока»

## Поддержка сложного письма

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

TLF предоставляет поддержку сложного письма. Поддержка сложного письма включает возможность отображения и редактирования текста с направлением письма справа налево. TLF также позволяет отображать и редактировать текст с комбинированным направлением письма слева направо и справа налево, например арабский и иврит. Среда не только поддерживает вертикальное размещение текста для китайского, японского и корейского языков, но также поддерживает элементы tate-chu-yoko (TCY). Элементы TCY являются блоками горизонтального текста, встроенного в вертикальные строки текста. Поддерживаются следующие системы письма и языки:

- латинский (английский, испанский, французский, вьетнамский и так далее);
- греческий, кириллица, армянский, грузинский и эфиопский;
- арабский и иврит;
- иероглифы и кана (китайский, японский и корейский языки), хангыль, джохаб (корейский язык);
- тайский, лаосский и кхмерский;
- девангари, бенгальский, гурмухи, малаялам, телугу, тамильский, гуджарати, ория, каннада и тибетский;
- тифинаг, ий, чероки, канадское слоговое письмо, дезерет, шавиан, ваи, тагальский, хануну, бухид и тагбанва.

## Использование Text Layout Framework в среде Flash Professional и Flex

С помощью классов TLF можно создавать пользовательские компоненты непосредственно в среде Flash. Кроме того, в Flash Professional CS5 представлен новый класс fl.text.TLFTextField, добавляющий функциональность TLF. Класс TLFTextField служит для создания текстовых полей в ActionScript, которые используют расширенные возможности отображения TLF. Создайте объект TLFTextField так же, как и текстовое поле класса TextField. Затем с помощью свойства `textFlow` назначьте расширенное форматирование из классов TLF.

Также можно использовать Flash Professional, чтобы создать экземпляр the TLFTextField в рабочей области с помощью текстового инструмента. Затем ActionScript можно использовать для управления форматированием и размещением содержимого текстовых полей с помощью классов TLF. Дополнительные сведения см. в описании класса TLFTextField в справочнике ActionScript 3.0 для платформы Adobe Flash Platform.

Если вы работаете в среде Flex, используйте классы TLF. Дополнительные сведения см. в разделе ««Использование Text Layout Framework» на странице 460».

# Использование Text Layout Framework

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Если вы работаете в среде Flex или создаете пользовательские текстовые компоненты, используйте классы TLF. TLF — это библиотека ActionScript 3.0, включенная в библиотеку textLayout.swc. Библиотека TLF включает около 100 классов и интерфейсов ActionScript 3.0, сгруппированных в десять пакетов. Эти пакеты являются подпакетами пакета flashx.textLayout.

## Классы Text Layout Framework

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Классы TLF могут быть сгруппированы в три категории:

- Структуры данных и классы форматирования
- классы визуализации;
- классы взаимодействия с пользователем.

### Структуры данных и классы форматирования

Структуры данных и классы форматирования TLF включены в следующие пакеты:

- flashx.textLayout.elements
- flashx.textLayout.formats
- flashx.textLayout.conversion

Основная структура данных TLF — это иерархия потока текста, которая определена в пакете элементов. Внутри данной структуры можно присваивать стили и атрибуты последовательностям текста с пакетом форматов. С помощью пакета преобразования можно также управлять импортом текста в структуру данных и экспортом из нее.

### Классы визуализации

Классы визуализации TLF включены в следующие пакеты:

- flashx.textLayout.factory
- flashx.textLayout.container
- flashx.textLayout.compose

Классы из этих пакетов упрощают визуализацию текста при его отображении в проигрывателе Flash Player. Пакет фабрики предоставляет простой способ отображения статического текста. Пакет контейнера включает классы и интерфейсы, определяющие отображение для динамического текста. Пакет компоновки определяет способы размещения и отображения динамического текста в контейнерах.

### Классы взаимодействия с пользователем

Классы TLF для взаимодействия с пользователем включены в следующие пакеты:

- flashx.textLayout.edit
- flashx.textLayout.operations
- flashx.textLayout.events

Пакеты редактирования и операций определяют классы, которые можно использовать, чтобы разрешить редактирование текстов, сохраненных в структуры данных. Пакет событий содержит обрабатывающие события классы.

## Общая процедура создания текста с помощью Text Layout Framework

Ниже предоставлено описание общей процедуры создания текста с использованием Text Layout Format.

1. Импортируйте отформатированный текст в структуры данных TLF. Дополнительные сведения см. в разделах ««Структурирование текста с использованием TLF» на странице 465» и ««Форматирование текста с использованием TLF» на странице 469».
2. Создайте один или несколько связанных контейнеров экранных объектов для текста. Дополнительные сведения см. в разделе ««Управление контейнерами текста с использованием TLF» на странице 471».
3. Свяжите текст в структурах данных с контейнерами и настройте параметры редактирования и прокрутки. Дополнительные сведения см. в разделе ««Включение выбора текста, редактирования и отмены с использованием TLF» на странице 472».
4. Создайте обработчик события для перекомпоновки текста в ответ на события resize (или другие). Дополнительные сведения см. в разделе ««Обработка событий с использованием TLF» на странице 472».

## Пример Text Layout Framework: макет газетной страницы

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

В следующем примере показано использование TLF для построения макета простой газетной страницы. Страница содержит большой заголовок, подзаголовок и основой раздел из нескольких колонок:

```
package
{
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.Event;
    import flash.geom.Rectangle;

    import flashx.textLayout.compose.StandardFlowComposer;
    import flashx.textLayout.container.ContainerController;
    import flashx.textLayout.container.ScrollPolicy;
    import flashx.textLayout.conversion.TextConverter;
    import flashx.textLayout.elements.TextFlow;
    import flashx.textLayout.formats.TextLayoutFormat;

    public class TLFNewsLayout extends Sprite
    {
        private var hTextFlow:TextFlow;
        private var headContainer:Sprite;
        private var headlineController:ContainerController;
        private var hContainerFormat:TextLayoutFormat;

        private var bTextFlow:TextFlow;
        private var bodyTextContainer:Sprite;
        private var bodyController:ContainerController;
        private var bodyTextContainerFormat:TextLayoutFormat;
```

```
        private const headlineMarkup:String = "<flow:TextFlow
xmlns:flow='http://ns.adobe.com/textLayout/2008'><flow:p textAlign='center'><flow:span
fontFamily='Helvetica' fontSize='18'>TLF News Layout Example</flow:span><flow:br/><flow:span
fontFamily='Helvetica' fontSize='14'>This example formats text like a newspaper page with a
headline, a subtitle, and multiple columns</flow:span></flow:p></flow:TextFlow>";

        private const bodyMarkup:String = "<flow:TextFlow
xmlns:flow='http://ns.adobe.com/textLayout/2008' fontSize='12' textIndent='10' marginBottom='15'
paddingTop='4' paddingLeft='4'><flow:p marginBottom='inherit'><flow:span>There are many
</flow:span><flow:span fontStyle='italic'>such</flow:span><flow:span> lime-kilns in that tract
of country, for the purpose of burning the white marble which composes a large part of the
substance of the hills. Some of them, built years ago, and long deserted, with weeds growing in
the vacant round of the interior, which is open to the sky, and grass and wild-flowers rooting
themselves into the chinks of the stones, look already like relics of antiquity, and may yet be
overspread with the lichens of centuries to come. Others, where the lime-burner still feeds his
daily and nightlong fire, afford points of interest to the wanderer among the hills, who seats
himself on a log of wood or a fragment of marble, to hold a chat with the solitary man. It is a
lonesome, and, when the character is inclined to thought, may be an intensely thoughtful
occupation; as it proved in the case of Ethan Brand, who had mused to such strange purpose, in
days gone by, while the fire in this very kiln was burning.</flow:span></flow:p><flow:p
marginBottom='inherit'><flow:span>The man who now watched the fire was of a different order, and
troubled himself with no thoughts save the very few that were requisite to his business. At
frequent intervals, he flung back the clashing weight of the iron door, and, turning his face
from the insufferable glare, thrust in huge logs of oak, or stirred the immense brands with a
long pole. Within the furnace were seen the curling and riotous flames, and the burning marble,
almost molten with the intensity of heat; while without, the reflection of the fire quivered on
the dark intricacy of the surrounding forest, and showed in the foreground a bright and ruddy
little picture of the hut, the spring beside its door, the athletic and coal-begrimed figure of
the lime-burner, and the half-frightened child, shrinking into the protection of his father's
shadow. And when again the iron door was closed, then reappeared the tender light of the half-
full moon, which vainly strove to trace out the indistinct shapes of the neighboring mountains;
and, in the upper sky, there was a flitting congregation of clouds, still faintly tinged with the
rosy sunset, though thus far down into the valley the sunshine had vanished long and long
ago.</flow:span></flow:p></flow:TextFlow>";

        public function TLFNewsLayout()
        {
            //wait for stage to exist
            addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
        }

        private function onAddedToStage(evtObj:Event):void
        {
            removeEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
            stage.scaleMode = StageScaleMode.NO_SCALE;
            stage.align = StageAlign.TOP_LEFT;

            // Headline text flow and flow composer
            hTextFlow = TextConverter.importToFlow(headlineMarkup,
TextConverter.TEXT_LAYOUT_FORMAT);

            // initialize the headline container and controller objects
            headContainer = new Sprite();
            headlineController = new ContainerController(headContainer);
            headlineController.verticalScrollPolicy = ScrollPolicy.OFF;
            hContainerFormat = new TextLayoutFormat();
            hContainerFormat.paddingTop = 4;
```

```
            hContainerFormat.paddingRight = 4;
            hContainerFormat.paddingBottom = 4;
            hContainerFormat.paddingLeft = 4;

            headlineController.format = hContainerFormat;
            hTextFlow.flowComposer.addController(headlineController);
            addChild(headContainer);
            stage.addEventListener(flash.events.Event.RESIZE, resizeHandler);

            // Body text TextFlow and flow composer
            bTextFlow = TextConverter.importToFlow(bodyMarkup,
TextConverter.TEXT_LAYOUT_FORMAT);

            // The body text container is below, and has three columns
            bodyTextContainer = new Sprite();
            bodyController = new ContainerController(bodyTextContainer);
            bodyTextContainerFormat = new TextLayoutFormat();
            bodyTextContainerFormat.columnCount = 3;
            bodyTextContainerFormat.columnGap = 30;

            bodyController.format = bodyTextContainerFormat;
            bTextFlow.flowComposer.addController(bodyController);
            addChild(bodyTextContainer);
            resizeHandler(null);
        }

        private function resizeHandler(event:Event):void
        {
            const verticalGap:Number = 25;
            const stagePadding:Number = 16;
            var stageWidth:Number = stage.stageWidth - stagePadding;
            var stageHeight:Number = stage.stageHeight - stagePadding;
            var headlineWidth:Number = stageWidth;
            var headlineContainerHeight:Number = stageHeight;

            // Initial compose to get height of headline after resize
            headlineController.setCompositionSize(headlineWidth,
headlineContainerHeight);
            hTextFlow.flowComposer.compose();
            var rect:Rectangle = headlineController.getContentBounds();
            headlineContainerHeight = rect.height;
```

```
            // Resize and place headline text container
            // Call setCompositionSize() again with updated headline height
            headlineController.setCompositionSize(headlineWidth, headlineContainerHeight );
            headlineController.container.x = stagePadding / 2;
            headlineController.container.y = stagePadding / 2;
            hTextFlow.flowComposer.updateAllControllers();

            // Resize and place body text container
            var bodyContainerHeight:Number = (stageHeight - verticalGap -
headlineContainerHeight);
            bodyController.format = bodyTextContainerFormat;
            bodyController.setCompositionSize(stageWidth, bodyContainerHeight );
            bodyController.container.x = (stagePadding/2);
            bodyController.container.y = (stagePadding/2) + headlineContainerHeight +
verticalGap;
            bTextFlow.flowComposer.updateAllControllers();
        }
    }
}
```

Класс TLFNewsLayout использует два контейнера текста. Один контейнер отображает заголовок и подзаголовок, а другой используется для отображения трех колонок основного текста. В целях упрощения текст жестко запрограммирован в примере в качестве текста разметки TLF. Переменная `headlineMarkup` содержит как заголовок, так и подзаголовок, а переменная `bodyMarkup` содержит основной раздел. Дополнительные сведения по TLF Markup см. разделе ««Структурирование текста с использованием TLF» на странице 465».

После инициализации функция `onAddedToStage()` импортирует текст заголовка в объект TextFlow, который является основной структурой данных TLF:

```
hTextFlow = TextConverter.importToFlow(headlineMarkup, TextConverter.TEXT_LAYOUT_FORMAT);
```

Затем создается объект Sprite для контейнера, а также создается контроллер, который связывается с контейнером:

```
headContainer = new Sprite();
headlineController = new ContainerController(headContainer);
```

Выполняется инициализация контроллера для настройки форматирования, прокрутки и других параметров. Контроллер содержит геометрию, определяющую границы контейнера, в которой размещается текст. Объект TextLayoutFormat содержит параметры форматирования:

```
hContainerFormat = new TextLayoutFormat();
```

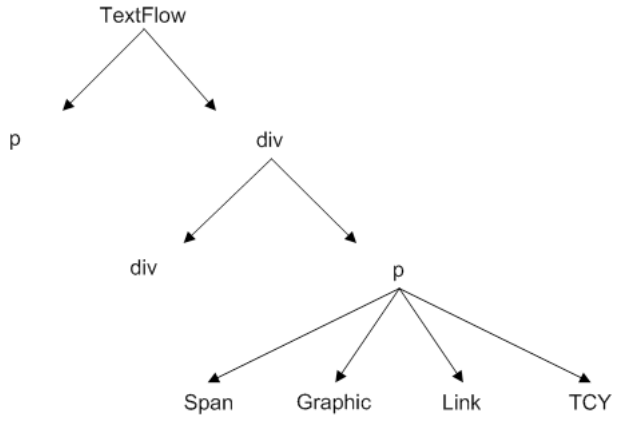
Контроллер назначается для компоновщика потока, и функция добавляет контейнер в список отображения. Реальная компоновка и отображение контейнеров задерживаются методом `resizeHandler()`. Аналогичная процедура выполняется для инициализации объекта TextFlow основного текста.

Метод `resizeHandler()` измеряет как доступное пространство для визуализации контейнеров, так и размеры контейнеров соответственно. Первоначальный вызов метода `compose()` позволяет вычислить правильную высоту контейнера заголовка. Метод `resizeHandler()` может затем разместить и отобразить контейнер заголовка с помощью метода `updateAllControllers()`. Наконец, в методе `resizeHandler()` размер контейнера заголовка используется для определения расположения контейнера основного раздела.

# Структурирование текста с использованием TLF

Для представления текста TLF использует иерархическое дерево. Каждый узел в дереве является экземпляром класса, определенного в пакете элементов. Например, корневой узел дерева всегда валяется экземпляром класса TextFlow. Класс TextFlow представляет полную историю текста. Материал — это коллекция текстов и других элементов, которые обрабатываются как единое целое или как поток. Единичный материал может потребовать для отображения более одной колонки или контейнера текста.

Оставшиеся элементы отдельно от корневого узла являются свободными элементами на базе элементов XHTML. На следующей диаграмме показана иерархия среды:




*Иерархия TextFlow*

## Text Layout Framework Markup

Понимание структуры TLF также необходимо при работе с разметкой TLF Markup. Разметка TLF Markup является представлением XML текста, сохраненного в компоненте TLF. Несмотря на то, что среда поддерживает также другие форматы XML, уникальность разметки TLF Markup заключается в том, что она особым образом основана на структуре иерархии TextFlow. При экспорте XML из TextFlow с помощью этого формата разметки XML экспортируется с помощью данной иерархии в неизменном виде.

TLF Markup обеспечивает максимально точное представление текста в иерархии TextFlow. Язык разметки предоставляет теги для всех основных элементов иерархии TextFlow, а также предоставляет атрибуты для всех свойств форматирования, доступных в классе TextLayoutFormat.

В следующей таблице содержатся теги, которые можно использовать в разметке TLF.

| Элемент | Описание | Дочерний элемент | Класс |
|---|---|---|---|
| textflow | Корневой элемент разметки. | div, p | TextFlow |
| div | Деление в объекте TextFlow. Может содержать группу абзацев. | div, list, p | DivElement |
| p | Абзац. | a, tcy, span, img, tab, br, g | ParagraphElement |
| a | Ссылка. | tcy, span, img, tab, br, g | LinkElement |
| tcy | Строка горизонтального текста (используется в вертикальном объекте TextFlow). | a, span, img, tab, br, g | TCYElement |

| Элемент | Описание | Дочерний элемент | Класс |
|---|---|---|---|
| span | Строка текста в абзаце. | | SpanElement |
| img | Изображение в абзаце. | | InlineGraphicElement |
| tab | Символ табуляции. | | TabElement |
| br | Символ разрыва. Используется в конце строки в абзаце; текст следует, начиная со следующей строки, однако останется в том же абзаце. | | BreakElement |
| linkNormalFormat | Определяет атрибуты форматирования для ссылок в нормальном состоянии. | TextLayoutFormat | TextLayoutFormat |
| linkActiveFormat | Определяет атрибуты форматирования, используемые для ссылок в активном состоянии, то есть при нажатии кнопки мыши. | TextLayoutFormat | TextLayoutFormat |
| linkHoverFormat | Определяет атрибуты форматирования, используемые для ссылок в состоянии наведения, когда мыши находится в пределах ссылки. | TextLayoutFormat | TextLayoutFormat |
| li | Элемент списка. Должен быть внутри элемента списка. | div, li, list, p | ListItemElement |
| list | Список. Списки могут быть вложенными или размещться рядом друг с другом. К элементам списка можно применять другую маркировку или схемы нумерации. | div, li, list, p | ListElement |
| g | Элемент группы. Используется для группировки элементов в абзаце. Позволяет группировать элементы в абзаце. | a, tcy, span, img, tab, br, g | SubParagraphGroupElement |

### Дополнительные разделы справки

Разметка списков TLF 2.0

Объекты SubParagraphGroupElements и typeName TLF 2.0

## Использование нумерованных и маркированных списков

Классы ListElement и ListItemElement можно использовать для добавления маркированных списков в текстовые элементы управления. Маркированные списки могут быть вложенными, и в них можно использовать различные маркеры и автоматическую нумерацию, а также нумерацию в стиле абриса.

Для создания списков в текстовых потоках используйте тег`<list>`. Затем для каждого элемента в списке `<list>` используйте тег `<li>`. Внешний вид маркеров можно настроить с помощью класса ListMarkerFormat.

В примере ниже создаются простые списки.

```
<flow:list paddingRight="24" paddingLeft="24">
    <flow:li>Item 1</flow:li>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

Списки можно вкладывать в другие списки, как показано в следующем примере:

```
<flow:list paddingRight="24" paddingLeft="24">
    <flow:li>Item 1</flow:li>
    <flow:list paddingRight="24" paddingLeft="24">
        <flow:li>Item 1a</flow:li>
        <flow:li>Item 1b</flow:li>
        <flow:li>Item 1c</flow:li>
    </flow:list>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

Для указания типа маркера в списке используйте свойство `listStyleType` класса ListElement. Это свойство может принимать любое значение, определяемое классом ListStyleType (например `check`, `circle`, `decimal` и `box`). В примере ниже создаются списки с различными типами маркера и настраиваемым приращением счётчика:

```
<flow:list paddingRight="24" paddingLeft="24" listStyleType="upperAlpha">
<flow:li>upperAlpha item</flow:li> <flow:li>another</flow:li> </flow:list> <flow:list
paddingRight="24" paddingLeft="24" listStyleType="lowerAlpha"> <flow:li>lowerAlpha
item</flow:li> <flow:li>another</flow:li> </flow:list> <flow:list paddingRight="24"
paddingLeft="24" listStyleType="upperRoman"> <flow:li>upperRoman item</flow:li>
<flow:li>another</flow:li> </flow:list> <flow:list paddingRight="24" paddingLeft="24"
listStyleType="lowerRoman"> <flow:listMarkerFormat> <!-- Increments the list by 2s rather than
1s. --> <flow:ListMarkerFormat counterIncrement="ordered 2"/> </flow:listMarkerFormat>
<flow:li>lowerRoman item</flow:li> <flow:li>another</flow:li> </flow:list>
```

Класс ListMarkerFormat используется для определения счётчика. В дополнение к указанию приращения счётчик можно сбросить с помощью свойства `counterReset`.

Внешний вид маркеров в списках можно дополнительно настроить с помощью свойств `afterContent` и `beforeContent` класса ListMarkerFormat. Эти свойства применяются к контенту, который отображается до и после маркера.

В примере ниже перед маркером добавляется строка «XX», после маркера — строка «YY».

```
<flow:list listStyleType="upperRoman" paddingLeft="36" paddingRight="24">
    <flow:listMarkerFormat>
        <flow:ListMarkerFormat fontSize="16"
            beforeContent="XX"
            afterContent="YY"
            counterIncrement="ordered -1"/>
        </flow:listMarkerFormat>
    <flow:li>Item 1</flow:li>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

В свойстве `content` можно определить дополнительные параметры для формата маркера. В примере ниже отображается маркер в виде римской цифры в верхнем регистре.

```
<flow:list listStyleType="disc"  paddingLeft="96" paddingRight="24">
    <flow:listMarkerFormat>
        <flow:ListMarkerFormat fontSize="16"
            beforeContent="Section "
            content="counters(ordered,&quot;*&quot;,upperRoman)"
            afterContent=": "/>
    </flow:listMarkerFormat>
    <flow:li>Item 1</li>
    <flow:li>Item 2</li>
    <flow:li>Item 3</li>
</flow:list>
```

Как показано в примере выше, через свойство `content` можно также добавить суффикс — текстовую строку, которая отображается после маркера, но перед `afterContent`. Чтобы вставить эту строку при передаче XML-контента в поток, заключите её в HTML-кавычки `&quote;` вместо простых кавычек (`"<  >"`).

### Дополнительные разделы справки

Разметка списков TLF 2.0

## Использование отступов в TLF

Каждый FlowElement имеет свойства отступов, в помощью которых можно управлять положением каждой области контента элемента и пространством между областями контента.

Общая ширина элемента складывается из ширины его контента и значений свойств `paddingLeft` и `paddingRight`. Общая высота элемента складывается из высоты его контента и значений свойств `paddingTop` и `paddingBottom`.

Отступ — это пространство между границей и контентом. Свойства отступов: `paddingBottom`, `paddingTop`, `paddingLeft` и `paddingRight`. Заполнение может применяться к объекту TextFlow, а также к следующим дочерним элементам:

- div
- img
- li
- list
- p

Свойства отступов нельзя применять к элементам «span».

В примере ниже устанавливаются свойства отступов для объекта TextFlow.

```
<flow:TextFlow version="2.0.0" xmlns:flow="http://ns.adobe.com/textLayout/2008" fontSize="14"
textIndent="15" paddingTop="4" paddingLeft="4" fontFamily="Times New Roman">
```

Допустимые значения для свойств отступов: число (в пикселах), процент, auto или inherit. Значение по умолчанию — «auto», которое означает, что отступ рассчитывается автоматически и устанавливается равным 0 для всех элементов, кроме ListElement. Для ListElements «auto» означает 0 за исключением стороны начала списка, где используется значение свойства `listAutoPadding`. Значение по умолчанию для `listAutoPadding` — 40, которое задаёт для списка отступ по умолчанию.

Свойства отступов не наследуются по умолчанию. Значения «auto» и «inherit» являются константами, определенными в классе FormatValue.

Свойства отступов могут содержать отрицательные значения.

**Дополнительные разделы справки**

Изменения в отступах в TLF 2.0

# Форматирование текста с использованием TLF

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Пакет flashx.textLayout.formats содержит интерфейсы и классы, с помощью которых можно назначать форматы любым объектам FlowElement в дереве иерархии потока текста. Существует два способа применения форматирования. С помощью специального объекта форматирования можно назначить определенный формат или одновременно назначить группу форматов.

Интерфейс ITextLayoutFormat содержит все форматы, которые можно применить к объекту FlowElement. Некоторые форматы применяются ко всему контейнеру или абзацу текста, но не применяются логически к отдельным символам. Например, форматирование, такое как выравнивание и расположение символов табуляции, применяется ко всему абзацу, но не применяется к отдельным символам.

## Назначение форматов объекту FlowElement с помощью свойств

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Можно задать форматы для любого объекта FlowElement путем назначения свойств. Класс FlowElement реализует интерфейс ITextLayoutFormat, поэтому любой подкласс класса FlowElement должен также реализовывать этот интерфейс.

Например, следующий код показывает принцип назначения отдельных форматов экземпляру ParagraphElement:

```
var p:ParagraphElement = new ParagraphElement();
p.fontSize = 18;
p.fontFamily = "Arial";
```

## Назначение форматов объекту FlowElement с помощью класса TextLayoutFormat

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Форматы можно применить к объекту FlowElement с помощью класса TextLayoutFormat. Этот класс используется для создания специального объекта форматирования, содержащего все необходимые значения форматирования. Затем можно назначить этот объект свойству `format` объекта FlowElement. Классы TextLayoutFormat и FlowElement реализуют интерфейс ITextLayoutFormat. Такое упорядочение обеспечивает то, что оба класса содержат одинаковые свойства формата.

Дополнительные сведения см. в описании класса TextLayoutFormat в справочнике ActionScript 3.0 для платформы Adobe Flash Platform.

## Наследование форматов

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Форматы наследуются через иерархию потока текста. При назначении экземпляра TextLayoutFormat экземпляру FlowElement с дочерними элементами инфраструктура начинает процесс, называемый *каскадом*. В каскадном списке среда рекурсивно исследует каждый узел в иерархии, который наследуется от объекта FlowElement. Затем она определяет необходимость назначения унаследованных значений каждому свойству форматирования. В процессе каскада применяются следующие правила:

1. Значения свойств наследуются только из непосредственного родительского элемента (иногда называемого родителем).
2. Значения свойств наследуются только в том случае, если для свойства еще не задано значение (т. е. значением является `undefined`).
3. Некоторые атрибуты не наследуют значения при статусе undefined, если для значения атрибута не задан параметр наследования или константа `flashx.textLayout.formats.FormatValue.INHERIT`.

Например, при установке значения `fontSize` на уровне TextFlow настройка применяется ко всем элементам в TextFlow. Другими словами, значения располагаются каскадом в иерархии потока текста. Однако можно переопределить значение в указанном элементе путем прямого назначения нового значения элементу. В качестве контрпримера: если задано значение `backgroundColor` на уровне TextFlow, дочерний элемент TextFlow не наследует это значение. Свойство `backgroundColor` является свойством, которое не наследуется от родительского элемента в каскадном представлении. Это поведение можно переопределить, установив для свойства `backgroundColor` каждого дочернего элемента значение `flashx.textLayout.formats.FormatValue.INHERIT`.

Дополнительные сведения см. в описании класса TextLayoutFormat в справочнике ActionScript 3.0 для платформы Adobe Flash Platform.

# Импорт и экспорт текста с использованием TLF

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Класс TextConverter в flashx.textLayout.conversion.* пакет позволяет импортировать и экспортировать текст в TLF. Воспользуйтесь этим классом, если планируете загрузить текст в среду выполнения вместо того, чтобы скомпилировать текст в SWF-файле. Этот класс можно также использовать для экспорта текста, хранящегося в экземпляре TextFlow, в объект String или XML.

Импорт и экспорт являются простыми процедурами. Можно вызвать метод `export()` или метод `importToFlow()`. Эти методы являются частью класса TextConverter. Оба метода являются статическими. Это означает, что предпочтительнее вызывать методы в классе TextConverter, а не в экземпляре класса TextConverter.

Классы в пакете flashx.textLayout.conversion позволяют свободно выбирать, где будет храниться текст. Например, при сохранении текста в базе данных можно импортировать текст в инфраструктуру для отображения. Затем с помощью классов в пакете flashx.textLayout.edit можно изменить текст и экспортировать измененный текст в базу данных.

Дополнительные сведения см. в описании класса flashx.textLayout.conversion в справочнике ActionScript 3.0 для платформы Adobe Flash Platform.

# Управление контейнерами текста с использованием TLF

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

После того как текст будет сохранен в структуре данных TLF, проигрыватель Flash Player сможет отобразить его. Текст, сохраненный в иерархи потока, следует преобразовать в формат, отображаемый Flash Player. Среда TLF предоставляет два способа создания экранных объектов из потока. Первый более простой подход подходит для отображения статического текста. Второй более сложный подход позволяет создавать динамический текст, который можно выделять и редактировать. В обоих случаях в итоге текст преобразуется в экземпляры класса TextLine, являющегося компонентом пакета flash.text.engine.* в приложении Flash Player 10.

## Создание статического текста

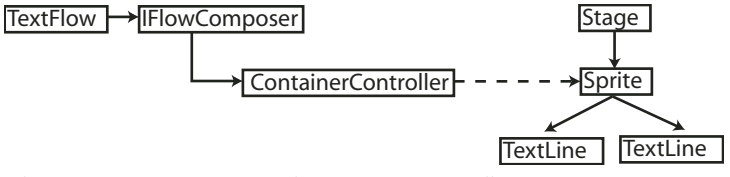
Для простого способа используется класс TextFlowTextLineFactory, который находится в пакете flashx.textLayout.factory. Преимущество этого способа по сравнению со способом FlowComposer, кроме его простоты, заключается также в меньшем влиянии на память. Этот способ рекомендуется использовать для статического текста, который не должен редактироваться, выбираться или прокручиваться пользователем.

Дополнительные сведения см. в описании класса TextFlowTextLineFactory в справочнике ActionScript 3.0 для платформы Adobe Flash Platform.

## Создание динамического текста и контейнеров

Воспользуйтесь компоновщиком потока, если хотите контролировать отображение текста больше, чем с помощью класса TextFlowTextLineFactory. Например, с помощью компоновщика потока пользователи смогут выделять и редактировать текст. Дополнительные сведения см. в разделе ««Включение выбора текста, редактирования и отмены с использованием TLF» на странице 472».

Компоновщик потока — это экземпляр класса StandardFlowComposer в пакете flashx.textLayout.compose. Компоновщик пакета управляет преобразованием экземпляров TextFlow в TextLine, а также размещением этих экземпляров TextLine в одном или нескольких контейнерах.




*IFlowComposer имеет ноль или более ContainerControllers*

Каждый экземпляр TextFlow имеет соответствующий объект, реализующий интерфейс IFlowComposer. Доступ к этому объекту IFlowComposer осуществляется с помощью свойства `TextFlow.flowComposer`. Через это свойство можно вызывать методы, определенные интерфейсом IFlowComposer. Данные методы позволяют связывать текст с одним или несколькими контейнерами и подготавливать текст к отображению в контейнере.

Контейнер является экземпляром класса Sprite, который представляет собой подкласс класса DisplayObjectContainer. Оба эти класса являются компонентами API списка отображения Flash Player. Контейнер является более эффективной формой ограничивающего прямоугольника, используемого в классе TextLineFactory. Подобно ограничивающему прямоугольнику, контейнер задает область, в которой отображаются экземпляры TextLine. В отличие от ограничивающего прямоугольника для контейнера имеется соответствующий объект-контроллер. Контроллер управляет прокруткой, компоновкой, связыванием, форматированием и обработкой событий для контейнера или набора контейнеров. Каждый контейнер имеет соответствующий объект-контроллер, который является экземпляром класса ContainerController в пакете flashx.textLayout.container.

Чтобы отобразить текст, создайте объект-контроллер для управления контейнером и свяжите его с компоновщиком потока. После связывания контейнера скомпонуйте текст для его отображения. Таким образом, контейнеры имеют два состояния: компоновка и отображение. Компоновка — это процесс преобразования текста из иерархии потока текста в экземпляры TextLine и вычисления того, подходят ли эти экземпляры для размещения в контейнере. Отображение — это процесс обновления списка отображения Flash Player.

Дополнительные сведения см. в описании классов IFlowComposer, StandardFlowComposer и ContainerController в справочнике ActionScript 3.0 для платформы Adobe Flash Platform.

# Включение выбора текста, редактирования и отмены с использованием TLF

**Flash Player 9.0 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Управление возможностью выбора или редактирования текста осуществляется на уровне потока текста. Каждый экземпляр класса TextFlow имеет соответствующий координатор взаимодействия. Доступ к координатору взаимодействия объекта TextFlow можно получить через свойство `TextFlow.interactionManager` объекта. Для включения выбора текста следует назначить экземпляр класса SelectionManager свойству `interactionManager`. Для включения и выбора, и редактирования текста следует назначить экземпляр класса EditManager вместо экземпляра класса SelectionManager. Для включения операций отмены создайте экземпляр класса UndoManager и добавьте его в качестве аргумента в вызов конструктора объекта EditManager. Класс UndoManager сохраняет историю последних действий пользователя по редактированию, а также позволяет отменить или выполнить повторно определенные действия редактирования. Все эти три класса являются компонентами пакета редактирования.

Дополнительные сведения см. в описании классов SelectionManager, EditManager и UndoManager в справочнике ActionScript 3.0 для платформы Adobe Flash Platform.

# Обработка событий с использованием TLF

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Объекты TextFlow отправляют события в различных ситуациях, включая:

- изменение текста или макета;
- перед началом операции и после выполнения операции;

- при изменении состояния объекта FlowElement;
- после завершения операции compose.

Дополнительные сведения см. в описании класса flashx.textLayout.events в справочнике ActionScript 3.0 для платформы Adobe Flash Platform.

### Дополнительные разделы справки

Объект FlowElement и события LinkElement и EventMirrors TLF

# Размещение изображений в тексте

Чтобы разместить объект InlineGraphicElement внутри текста, используйте следующие свойства:

- Свойство `float` класса InlineGraphicElement
- Свойство `clearFloats` класса FlowElement

Свойство `float` определяет расположение рисунка и текста вокруг него. Свойство `clearFloats` определяет расположение элементов абзаца относительно `float`.

Для управления расположением изображения внутри текстовом элементе используйте свойство `float`. В примере ниже изображение добавляется в абзац и выравнивается по левому краю, чтобы текст обтекал его справа.

```
<flow:p paragraphSpaceAfter="15" >Images in a flow are a good thing. For example, here is a
float. It should show on the left: <flow:img float="left" height="50" width="19"
source="../assets/bulldog.jpg"></flow:img> Don't you agree? Another sentence here. Another
sentence here. Another sentence here. Another sentence here. Another sentence here. Another
sentence here. Another sentence here. Another sentence here.</flow:p>
```

Допустимые значения для свойства `float`: «left», «right», «start», «end» и «none» (без кавычек). Эти константы определяются классом Float. Значение по умолчанию — «none».

Свойство `clearFloats` полезно в тех случаях, когда нужно подкорректировать начальное положение следующих абзацев, которые должны обтекать изображение. Предположим, например, что есть изображение, которое крупнее первого абзаца. Чтобы второй абзац гарантированно начинался *после* изображения, нужно установить свойство `clearFloats`.

В примере ниже используется изображение, которое выше текста первого абзаца. Чтобы второй абзац начинался после изображения в текстовом блоке, в этом примере для свойства `clearFloats` во втором абзаце установлено значение «end».

```
<flow:p paragraphSpaceAfter="15" >Here is another float, it should show up on the right:
<flow:img float="right" height="50" elementHeight="200" width="19"
source="../assets/bulldog.jpg"></flow:img>We'll add another paragraph that should clear past
it.</flow:p><flow:p clearFloats="end" >This should appear after the previous float on the
right.</flow:p>
```

Допустимые значения для свойства `clearFloats`: «left», «right», «end», «start», «none» и «both». Эти константы определяются классом ClearFloats. Для свойства `clearFloats` можно также установить значение «inherit», которое является константой, определённой в классе FormatValue. Значение по умолчанию — «none».

### Дополнительные разделы справки

Плавающие элементы TLF

# Глава 24. Работа со звуком

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Язык ActionScript разработан для создания комплексных интерактивных приложений. И звук является немаловажным, но часто недооцениваемым элементом таких мощных приложений. Звуковые эффекты можно добавлять в видеоигры, использовать как звуковой отклик для пользовательского интерфейса приложения, можно даже создать программу, анализирующую mp3-файлы, загружаемые через Интернет, основным элементом которой будет звук.

Можно загружать внешние аудиофайлы и работать с аудиоданными, встроенными в SWF-файл. Вы также можете управлять звуком, создавать зрительное представление аудиоданных и захватывать звук с микрофона пользователя.

### Дополнительные разделы справки

Пакет flash.media

flash.events.SampleDataEvent

## Основные сведения о работе со звуком

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Компьютеры могут захватывать и кодировать цифровой звук (компьютерное представление звуковых данных), а также сохранять его и извлекать для воспроизведения через динамики. Воспроизводить звук можно с помощью проигрывателя Adobe® Flash® Player или среды Adobe® AIR™ и кода ActionScript.

Когда аудиоданные преобразуются в цифровую форму, они имеют различные характеристики, такие как громкость, стерео- или монофоничность. При воспроизведении звука с помощью ActionScript эти характеристики можно регулировать: делать звук громче, или выбрать тот или иной канал, чтобы он исходил из определенной точки.

Прежде чем приступить к управлению звуком с помощью ActionScript, необходимо загрузить аудиоданные в проигрыватель Flash Player или среду AIR. Существует пять способов загрузки звуковых данных в Flash Player или AIR, чтобы с ними можно было работать, используя ActionScript.

- Загрузка в SWF внешнего звукового файла (например, файла mp3).
- Встраивание звуковой информации в SFW-файл непосредственно при его создании.
- Захват звука с микрофона, подключенного к компьютеру пользователя.
- Потоковая передача звука с сервера.
- Динамическая генерация и воспроизведение звука.

Когда аудиоданные загружаются из внешнего аудиофайла, можно запустить воспроизведение начала этого файла, пока будут загружаться остальные данные.

Хотя для кодирования цифрового аудио существует много разных форматов, ActionScript 3.0, Flash Player и AIR поддерживают аудиофайлы, сохраненные в формате mp3. Они не могут загружать и воспроизводить аудиофайлы в других форматах, таких как WAV или AIFF.

Для работы со звуком в ActionScript требуются несколько классов из пакета flash.media. Класс Sound служит для получения доступа к звуковой информации: сначала загружается аудиофайл или назначается функция событию, которое сэмплирует аудиоданные, а затем начинается воспроизведение. После запуска воспроизведения звука проигрыватель Flash Player или среда AIR предоставляет доступ к объекту SoundChannel. Так как загруженный аудиофайл может являться одним из нескольких звуков, воспроизводимых на компьютере пользователя, каждый из этих звуков использует собственный объект SoundChannel. Через динамики компьютера на самом деле воспроизводится объединенный вывод всех объектов SoundChannel. Данный экземпляр SoundChannel можно использовать для управления свойствами звука и остановки его воспроизведения. В завершение, если требуется управлять объединенным звуком, это можно сделать с помощью класса SoundMixer.

Для выполнения более конкретных задач при работе со звуком в ActionScript можно также использовать несколько других классов. Дополнительные сведения о классах, связанных со звуком, см. раздел «Обзор архитектуры звука» на странице 475.

### Важные понятия и термины

Ниже приводится список важных терминов, которые могут встретиться.

**Амплитуда** Расстояние между точкой на звуковой волне и нулевой (равновесной) линией.

**Скорость потока** Объем данных, который кодируется или передается для каждой секунды звукового файла. Для mp3-файлов скорость потока обычно указывается в тысячах бит в секунду (кб/с). Чем выше скорость потока, тем выше качество звуковой волны.

**Буферизация** Получение и сохранение аудиоданных перед началом воспроизведения.

**mp3** MPEG-1 Audio Layer 3 или mp3 — это популярный формат сжатия звука.

**Панорамирование** Распределение аудиосигнала между левым и правым каналами в стереофоническом звуковом поле.

**Пик** Самая высокая точка звуковой волны.

**Частота дискретизации** Определяет количество сэмплов, взятых за секунду из аналогового аудиосигнала для получения цифрового сигнала. Частота дискретизации стандартного аудиодиска составляет 44,1 кГц или 44100 сэмплов в секунду.

**Потоковая передача** Процесс воспроизведения более ранних частей звукового или видеофайла, пока более поздние части этого файла еще загружаются с сервера.

**Громкость** Громкость звука.

**Звуковая волна** Форма графика, отображающая изменяющиеся амплитуды звукового сигнала на протяжении времени.

# Обзор архитектуры звука

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Существует пять основных источников, из которых приложения могут загружать звук, а именно:

- внешние аудиофайлы, загружаемые во время выполнения;
- аудиоресурсы, встроенные в SWF-файл приложения;
- аудиоданные, полученные с микрофона, подключенного к системе пользователя;

- потоковые аудиоданные, получаемые с удаленного сервера мультимедийного содержимого, такого как Flash Media Server;
- аудиоданные, создаваемые динамически с помощью обработчика событий `sampleData`.

Воспроизводить аудиоданные можно либо после полной загрузки, либо в режиме потоковой передачи, то есть в процессе загрузки.

Звуковые классы ActionScript 3.0 поддерживают аудиофайлы, сохраненные в формате mp3. Они не могут загружать и воспроизводить аудиофайлы в других форматах, таких как WAV или AIFF. Однако начиная с проигрывателя Flash Player 9.0.115.0 появилась возможность загрузки и воспроизведения аудиофайлов AAC с помощью класса NetStream. Этот же прием используется для загрузки и воспроизведения видеосодержимого. Дополнительные сведения об этом приеме см. в главе «Работа с видео» на странице 508.

С помощью Adobe Flash Professional WAV- и AIFF-файлы можно импортировать, а затем встраивать в SWF-файл приложения в формате mp3. Инструмент разработки Flash также позволяет сжимать встроенные аудиофайлы, чтобы уменьшить их размер, однако при этом ухудшается качество звука. Дополнительные сведения см. в разделе «Импорт звуков» в руководстве *Использование Flash*.

В архитектуре звука ActionScript 3.0 используются следующие классы пакета flash.media.

| **Класс** | **Описание** |
|---|---|
| flash.media.Sound | Класс Sound обрабатывает загрузку звука, управляет его основными свойствами и запускает воспроизведение. |
| flash.media.SoundChannel | Когда приложение воспроизводит объект Sound, для управления воспроизведением создается новый объект SoundChannel. Объект SoundChannel контролирует громкость левого и правого каналов воспроизведения звука. Для каждого воспроизводимого звука создается отдельный объект SoundChannel. |
| flash.media.SoundLoaderContext | Класс SoundLoaderContext определяет, сколько секунд буферизации должно использоваться при загрузке звука, а также должен ли проигрыватель Flash Player или AIR проверять файл политики, загружая файл с сервера. Объект SoundLoaderContext используется в качестве параметра метода `Sound.load()`. |
| flash.media.SoundMixer | Класс SoundMixer управляет свойствами воспроизведения и безопасности, которые относятся ко всем звукам в приложении. В действительности, несколько аудиоканалов микшируются с помощью одного объекта SoundMixer, поэтому значения свойств объекта SoundMixer применяются ко всем объектам SoundChannel, воспроизводимым в данный момент. |
| flash.media.SoundTransform | Класс SoundTransform содержит значения, которые управляют громкостью и панорамированием звука. Объект SoundTransform можно применять, помимо прочего, к отдельному объекту SoundChannel, к глобальному объекту SoundMixer или к объекту Microphone. |
| flash.media.ID3Info | Объект ID3Info содержит свойства, представляющие метаданные ID3, которые часто хранятся в аудиофайлах формата mp3. |
| flash.media.Microphone | Класс Microphone представляет микрофон или другое устройство для ввода звука, подключенное к компьютеру пользователя. Аудиоввод с микрофона можно вывести на локальные динамики или отправить на удаленный сервер. Объект Microphone контролирует усиление, частоту амплитудно-импульской модуляции и другие характеристики собственного аудиопотока. |
| flash.media.AudioPlaybackMode | Класс AudioPlaybackMode определяет константы для свойства `audioPlaybackMode` класса SoundMixer. |

Каждый загружаемый и воспроизводимый звук должен иметь собственный экземпляр класса Sound и класса SoundChannel. Затем вывод нескольких экземпляров SoundChannel микшируется глобальным экземпляром класса SoundMixer во время воспроизведения.

Классы Sound, SoundChannel и SoundMixer не используются для аудиоданных, получаемых с микрофона или с сервера потокового мультимедийного содержимого, такого как Flash Media Server.

# Загрузка внешних звуковых файлов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Каждый экземпляр класса Sound предназначен для загрузки и запуска воспроизведения определенного аудиоресурса. Приложение не может повторно использовать объект Sound для загрузки другого звука. Если приложению требуется загрузить новый ресурс, то нужно создать новый объект Sound.

Если загружается небольшой звуковой файл, например звук щелчка для кнопки, приложение может создать новый объект Sound, чтобы он автоматически загрузил аудиофайл, как показано ниже.

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
```

Конструктор `Sound()` принимает в качестве первого параметра объект URLRequest. Когда передается значение для параметра URLRequest, новый объект Sound автоматически начинает загрузку указанного аудиоресурса.

Во всех случаях, кроме самых простых, приложение должно контролировать прогресс загрузки и отслеживать ошибки во время ее выполнения. Например, если звук щелчка имеет достаточно большой размер, он может не загрузиться полностью к тому моменту, как пользователь нажмет кнопку, запускающую воспроизведение звука. При попытке воспроизвести незагруженный звук может возникнуть ошибка выполнения. Во избежание таких ошибок следует дождаться завершения загрузки звука, прежде чем разрешать пользователям выполнять действия, в результате которых может начаться воспроизведение звуков.

Объект Sound отправляет несколько разных событий в процессе загрузки звука. Приложение может прослушивать эти события, чтобы отслеживать прогресс загрузки и разрешать воспроизведение звука только после полной загрузки. В следующей таблице перечислены события, которые может отправлять объект Sound.

| **Событие** | **Описание** |
|---|---|
| open (`Event.OPEN`) | Отправляется непосредственно перед началом загрузки звука. |
| progress (`ProgressEvent.PROGRESS`) | Отправляется периодически в ходе загрузки звука по мере получения данных из файла или потока. |
| id3 (`Event.ID3`) | Отправляется, когда для звука в формате mp3 доступны данные ID3. |
| complete (`Event.COMPLETE`) | Отправляется после завершения загрузки всех данных аудиоресурса. |
| ioError (`IOErrorEvent.IO_ERROR`) | Отправляется, когда не удается найти аудиофайл или когда процесс загрузки прерывается до получения всех аудиоданных. |

Следующий код демонстрирует воспроизведение звука после завершения его загрузки.

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(Event.COMPLETE, onSoundLoaded);
var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onSoundLoaded(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
```

Сначала код создает новый объект Sound, не передавая исходного значения для параметра URLRequest. Затем прослушивается событие `Event.COMPLETE`, которое отправляется объектом Sound и запускает выполнение метода `onSoundLoaded()` после завершения загрузки всех данных звука. Далее вызывается метод `Sound.load()` с новым значением параметра URLRequest для аудиофайла.

Метод `onSoundLoaded()` выполняется после завершения загрузки звука. Свойство `target` объекта Event является ссылкой на объект Sound. После этого вызывается метод `play()` объекта Sound, который запускает воспроизведение.

## Отслеживание процесса загрузки звука

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Звуковые файлы могут быть очень большими и требовать много времени на загрузку. Хотя проигрыватели Flash Player и AIR позволяют начинать воспроизведение звуков еще до завершения загрузки, возможно, пользователю нужно сообщить, сколько аудиоданных уже загружено и сколько из них уже воспроизведено.

Класс Sound отправляет два события, с помощью которых можно относительно просто отображать прогресс загрузки звука: `ProgressEvent.PROGRESS` и `Event.COMPLETE`. В следующем примере демонстрируется использование этих событий для отображения сведений о прогрессе загрузки звука.

```
import flash.events.Event;
import flash.events.ProgressEvent;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
s.addEventListener(Event.COMPLETE, onLoadComplete);
s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);

var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onLoadProgress(event:ProgressEvent):void
{
    var loadedPct:uint = Math.round(100 * (event.bytesLoaded / event.bytesTotal));
    trace("The sound is " + loadedPct + "% loaded.");
}

function onLoadComplete(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
function onIOError(event:IOErrorEvent)
{
    trace("The sound could not be loaded: " + event.text);
}
```

Этот код сначала создает объект Sound, а затем добавляет прослушиватели для его событий `ProgressEvent.PROGRESS` и `Event.COMPLETE`. После вызова метода `Sound.load()` и получения первых данных аудиофайла отправляется событие `ProgressEvent.PROGRESS`, которое запускает метод `onSoundLoadProgress()`.

Процент загруженных звуковых данных равен значению свойства `bytesLoaded` объекта ProgressEvent, деленному на значение свойства `bytesTotal`. Те же свойства `bytesLoaded` и `bytesTotal` есть и у объекта Sound. Приведенный выше пример наглядно показывает сообщения о ходе загрузки звука, но значения свойств `bytesLoaded` и `bytesTotal` можно легко использовать для обновления компонентов хода выполнения, таких как компоненты, доступные в среде Adobe Flex или инструменте разработки Adobe Flash.

Этот пример также демонстрирует, как приложение может распознавать ошибки при загрузке аудиофайлов и реагировать на них. Например, если не удается найти звуковой файл с заданным именем, объект Sound отправляет событие `Event.IO_ERROR`. В предыдущем коде выполняется метод `onIOError()`, который отображает краткое сообщение в случае возникновения ошибки.

# Работа с встроенными звуками

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Использование встроенных звуков вместо загрузки звуков из внешнего файла больше всего подходит для небольших файлов, которые используются в качестве индикаторов в пользовательском интерфейсе приложения, например звуки, воспроизводимые при нажатии кнопок.

Когда аудиофайл встраивается в приложение, размер полученного SWF-файла увеличивается на размер встроенного файла. Другими словами, встраивание больших звуковых файлов в приложение может привести к нежелательному увеличению размера SWF-файла.

Выбор метода встраивания аудиофайла в SWF-файл приложения зависит от среды разработки.

## Использование встроенных аудиофайлов в Flash

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Инструмент разработки Flash позволяет импортировать звуки в разных форматах и сохранять их в виде символов в библиотеке. После этого звуки можно добавлять в кадры на временной шкале или в кадры состояния кнопки, использовать их для поведений или непосредственно в коде ActionScript. В этом разделе описывается, как использовать звуки, встроенные в код ActionScript с помощью инструмента разработки Flash. Сведения о других способах использования встроенных звуков Flash см. в разделе «Импорт звуков» руководства *Использование Flash*.

**Встраивание аудиофайла с помощью инструмента разработки Flash**

1. Выберите команды «Файл» > «Импорт» > «Импортировать в библиотеку», а затем укажите аудиофайл и импортируйте его.
2. Щелкните правой кнопкой мыши имя импортированного файла на панели «Библиотека» и выберите «Свойства». Установите флажок «Экспорт для ActionScript».
3. В поле «Класс» введите имя, которое будет использоваться при ссылке на данный встроенный звук в ActionScript. По умолчанию будет использоваться имя аудиофайла, указанное в этом поле. Если в имени файла есть точка, как в имени «DrumSound.mp3», его необходимо изменить, например, «DrumSound»; код ActionScript не допускает использования точек в имени класса. В поле «Базовый класс» должен отображаться класс flash.media.Sound.
4. Нажмите кнопку «OK». Возможно, появится сообщение о том, что этот класс не найден в пути к классам. Нажмите кнопку «OK», чтобы продолжить. Если вы ввели имя класса, не соответствующее ни одному из классов в пути к классам приложения, автоматически создается новый класс, наследующий методы и свойства класса flash.media.Sound.
5. Чтобы использовать встроенный звук, укажите его имя класса в ActionScript. Например, следующий код в первую очередь создает новый экземпляр автоматически созданного класса DrumSound.

   ```
   var drum:DrumSound = new DrumSound();
   var channel:SoundChannel = drum.play();
   ```

   Класс DrumSound является подклассом класса flash.media.Sound и поэтому наследует его методы и свойства, включая метод `play()`, как показано выше.

## Использование встроенных аудиофайлов в Flex

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Существует много способов встраивания звуковых ресурсов в приложение Flex, включая следующие:

- использование в сценарии тега метаданных `[Embed]`;
- использование инструкции `@Embed` в MXML для назначения встроенного ресурса в качестве свойства такого компонента, как Button или SoundEffect;
- использование инструкции `@Embed` в CSS-файле.

В этом разделе описывается первый вариант, то есть встраивание звуков в код ActionScript приложения Flex с помощью тега метаданных `[Embed]`.

Чтобы встроить ресурс в код ActionScript, используйте тег метаданных `[Embed]`.

Поместите звуковой файл в основной исходной папке или в другой папке, включенной в путь сборки проекта. Когда компилятор обнаружит тег метаданных Embed, он создаст класс встроенных ресурсов. Обратиться к классу можно через переменную типа данных Class, объявленную сразу после тега метаданных `[Embed]`.

Следующий код встраивает звук с именем smallSound.mp3 и использует переменную `soundClass` для хранения ссылки на класс встроенных ресурсов, связанный с этим звуком. Затем код создает экземпляр класса встроенных ресурсов, передает его в качестве экземпляра класса Sound и вызывает для него метод `play()`.

```
package
{
    import flash.display.Sprite;
    import flash.media.Sound;
    import flash.media.SoundChannel;

    public class EmbeddedSoundExample extends Sprite
    {
        [Embed(source="smallSound.mp3")]
        public var soundClass:Class;

        public function EmbeddedSoundExample()
        {
            var smallSound:Sound = new soundClass() as Sound;
            smallSound.play();
        }
    }
}
```

Чтобы встроенный звук можно было использовать в качестве значения свойства компонента Flex, он должен быть передан в качестве экземпляра класса mx.core.SoundAsset, а не класса Sound. Подобный пример использования класса SoundAsset см. в разделе «Классы встроенных ресурсов» в документе «Изучение ActionScript 3.0».

# Работа с потоковыми аудиофайлами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Когда звуковой или видеофайл воспроизводится в процессе загрузки его данных, он называется *потоковым*. Внешние аудиофайлы, загружаемые с удаленного сервера, часто загружаются методом потоковой передачи, благодаря которой пользователю не нужно ждать загрузки всех аудиоданных, чтобы услышать звук.

Свойство `SoundMixer.bufferTime` представляет количество миллисекунд аудиоданных, которое проигрыватель Flash Player или AIR должен загрузить перед началом воспроизведения звука. Другими словами, если свойство `bufferTime` имеет значение 5000, проигрыватель Flash Player или AIR загружает фрагмент файла продолжительностью не менее 5000 миллисекунд, прежде чем начнется воспроизведение звука. По умолчанию свойство `SoundMixer.bufferTime` имеет значение 1000.

Приложение может переопределить глобальное значение `SoundMixer.bufferTime` для отдельного звука, задав новое значение `bufferTime` при загрузке звука. Чтобы переопределить время буферизации по умолчанию, сначала нужно создать новый экземпляр класса SoundLoaderContext, задать его свойство `bufferTime`, а затем передать его в качестве параметра методу `Sound.load()`, как показано ниже.

```
import flash.media.Sound;
import flash.media.SoundLoaderContext;
import flash.net.URLRequest;

var s:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
var context:SoundLoaderContext = new SoundLoaderContext(8000, true);
s.load(req, context);
s.play();
```

По мере воспроизведения проигрыватель Flash Player или AIR пытается поддерживать буфер звука того же или большего размера. Если скорость загрузки аудиоданных выше скорости воспроизведения, то звук будет воспроизводиться без прерываний. Однако, если скорость загрузки данных снижается из-за ограничений сети, точка воспроизведения может дойти до конца буфера аудиоданных. Когда это происходит, воспроизведение останавливается и автоматически возобновляется после загрузки новой порции данных.

Если воспроизведение приостановлено из-за того, что проигрыватель Flash Player или AIR ожидает данных для загрузки, об этом можно узнать с помощью свойства `Sound.isBuffering`.

# Работа с динамически создаваемым звуком

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

***Примечание.*** *Возможность динамического создания аудио доступна для использования, начиная с Flash Player версии 10 и Adobe AIR версии 1.5.*

Вместо загрузки или потоковой передачи существующего звука можно использовать аудиоданные, создаваемые в динамическом режиме. Можно создавать аудиоданные, когда регистрируется прослушиватель события `sampleData` объекта Sound. (Событие `sampleData` определено в классе SampleDataEvent в пакете flash.events.) В такой среде объект Sound не загружает аудиоданные из файла. Вместо этого он выполняет функцию сокета для потока звуковых данных, направляемых ему при помощи функции, назначенной для этого события.

Когда для объекта Sound добавляется прослушиватель события `sampleData`, объект периодически запрашивает данные для добавления в аудиобуфер. Этот буфер содержит данные, которые должен воспроизводить объект Sound. Когда вызывается метод `play()` объекта Sound, он отправляет событие `sampleData`, запрашивая новые аудиоданные. (Это происходит, только если объект Sound еще не загрузил данных mp3 из файла.)

Объект SampleDataEvent включает свойство`data`. В прослушивателе событий в этот объект `data` записываются объекты ByteArray. Записываемые для этого объекта массивы байтов добавляются в буфер к аудиоданным, которые воспроизводит объект Sound. Массив байтов в буфере является потоком, содержащим значения с плавающей запятой от -1 до 1. Каждое значение с плавающей запятой представляет амплитуду одного канала (левого или правого) звукового сэмпла. Частота дискретизации (сэмплирования) звука составляет 44100 сэмплов в секунду. Каждый сэмпл включает левый и правый канал, которые чередуются как данные с плавающей запятой в массиве байтов.

В функции обработчика используется метод `ByteArray.writeFloat()` для записи свойства `data` события `sampleData`. Например, следующий код создает синусоидальную волну.

```
var mySound:Sound = new Sound();
mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, sineWaveGenerator);
mySound.play();
function sineWaveGenerator(event:SampleDataEvent):void
{
    for (var i:int = 0; i < 8192; i++)
    {
        var n:Number = Math.sin((i + event.position) / Math.PI / 4);
        event.data.writeFloat(n);
        event.data.writeFloat(n);
    }
}
```

При вызове метода `Sound.play()`, приложение начинает вызывать обработчик события, запрашивая данные звуковых сэмплов. Проигрыватель продолжает отправлять события в течение воспроизведения звука до прекращения поступления данных или до вызова метода `SoundChannel.stop()`.

Время ожидания события варьируется от платформы к платформе и может измениться в будущих версиях Flash Player или AIR. Рекомендуется вычислять время ожидания, а не полагаться на указанную величину. Чтобы вычислить время ожидания, используйте следующую формулу:

```
(SampleDataEvent.position / 44.1) - SoundChannelObject.position
```

Передайте от 2048 до 8192 сэмплов свойству `data` объекта SampleDataEvent (для каждого вызова прослушивателя событий). Для лучшей производительности укажите как можно больше сэмплов (до 8192). Чем меньше сэмплов вы укажете, тем больше вероятность возникновения звуковых помех во время воспроизведения. Однако этот принцип действует по-разному на разных платформах, и помехи могут возникнуть в различных ситуациях, например при изменении размера обозревателя. Код, который на одной платформе работает всего при 2048 сэмплах, может плохо работать на другой платформе. Если вам требуется как можно меньший период ожидания, предоставьте пользователям возможность выбирать объем данных.

Если передавать меньше 2048 сэмплов (для одного вызова прослушивателя события `sampleData`), приложение останавливается после воспроизведения оставшихся сэмплов Затем объект SoundChannel отправляет событие SoundComplete.

## Изменение звука, полученного из данных mp3

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Для извлечение данных из объекта Sound служит метод `Sound.extract()`. Полученные данные можно использовать (или изменять) для записи в динамический поток другого объекта Sound с целью воспроизведения. Например, следующий код использует байты загруженного MP3-файла и передает их через функцию фильтра, `upOctave()`.

```
var mySound:Sound = new Sound();
var sourceSnd:Sound = new Sound();
var urlReq:URLRequest = new URLRequest("test.mp3");
sourceSnd.load(urlReq);
sourceSnd.addEventListener(Event.COMPLETE, loaded);
function loaded(event:Event):void
{
    mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, processSound);
    mySound.play();
}
function processSound(event:SampleDataEvent):void
{
        var bytes:ByteArray = new ByteArray();
        sourceSnd.extract(bytes, 8192);
        event.data.writeBytes(upOctave(bytes));
}
function upOctave(bytes:ByteArray):ByteArray
{
    var returnBytes:ByteArray = new ByteArray();
    bytes.position = 0;
    while(bytes.bytesAvailable > 0)
    {
        returnBytes.writeFloat(bytes.readFloat());
        returnBytes.writeFloat(bytes.readFloat());
        if (bytes.bytesAvailable > 0)
        {
            bytes.position += 8;
        }
    }
    return returnBytes;
}
```

## Ограничения для создаваемых звуков

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

При использовании прослушивателя события `sampleData` с объектом Sound единственными другими доступными методами Sound являются `Sound.extract()` и `Sound.play()`. Вызов любых других методов или свойств приводит к возникновению исключения. Все методы и свойства объекта SoundChannel остаются доступными.

# Воспроизведение звуков

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы воспроизвести загруженный звук, достаточно вызвать метод `Sound.play()` для объекта Sound, как показано ниже.

```
var snd:Sound = new Sound(new URLRequest("smallSound.mp3"));
snd.play();
```

Когда звук воспроизводится с помощью ActionScript 3.0, можно выполнять следующие операции:

- воспроизводить звук начиная с определенного места;

- приостановить звук и позднее возобновить воспроизведение с того же места;
- узнать точное время завершения воспроизведения звука;
- отслеживать прогресс воспроизведения звука;
- изменить громкость или панорамирование в процессе воспроизведения звука.

Для выполнения этих операций в ходе воспроизведения можно использовать классы SoundChannel, SoundMixer и SoundTransform.

Класс SoundChannel управляет воспроизведением одного звука. Свойство `SoundChannel.position` можно представить как точку воспроизведения, которая указывает текущую позицию в воспроизводимых аудиоданных.

Когда приложение вызывает метод `Sound.play()`, создается новый экземпляр класса SoundChannel для управления воспроизведением.

Приложение может воспроизводить звук, начиная с определенного места. Для этого нужно указать это место в миллисекундах в параметре `startTime` метода `Sound.play()`. Также можно указать фиксированное число повторов звука в непрерывной последовательности, передав числовое значение в параметре `loops` метода `Sound.play()`.

Когда метод `Sound.play()` вызывается с использованием обоих параметров `startTime` и `loops`, звук воспроизводится несколько раз, начиная с одного и того же места, как в следующем коде.

```
var snd:Sound = new Sound(new URLRequest("repeatingSound.mp3"));
snd.play(1000, 3);
```

В этом примере звук воспроизводится с точки, которая находится через одну секунду после начала звука, и повторяется три раза подряд.

## Приостановка и возобновление звука

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если приложение воспроизводит звуки большого размера, например песни или подкасты, вероятно, следует предоставить пользователю возможность приостанавливать и возобновлять воспроизведение таких аудиофайлов. В ActionScript воспроизводимый звук нельзя приостановить в буквальном смысле, его можно только остановить полностью. Однако звук можно воспроизводить с любой точки. Можно записать позицию звука в момент остановки, а затем продолжить его воспроизведение, начиная с этого же места.

Например, код загружает и воспроизводит аудиофайл, как показано ниже.

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var channel:SoundChannel = snd.play();
```

Во время воспроизведения звука свойство `SoundChannel.position` указывает точку файла, которая воспроизводится в данный момент. Приложение может сохранить значение позиции, в которой звук находился перед остановкой, как показано ниже.

```
var pausePosition:int = channel.position;
channel.stop();
```

Чтобы возобновить звук, передайте сохраненное ранее значение позиции, чтобы начать воспроизведение с того же места, где оно было остановлено.

```
channel = snd.play(pausePosition);
```

## Отслеживание воспроизведения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Возможно, приложению потребуется информация о том, когда заканчивается воспроизведение одного звука, чтобы запустить другой или удалить ресурсы, использованные при предыдущем воспроизведении. Класс SoundChannel отправляет событие `Event.SOUND_COMPLETE`, когда заканчивается воспроизведение его звука. Приложение может прослушивать это событие и, получив его, предпринять соответствующее действие, как показано ниже.

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("smallSound.mp3");
snd.load(req);

var channel:SoundChannel = snd.play();
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

public function onPlaybackComplete(event:Event)
{
    trace("The sound has finished playing.");
}
```

Класс SoundChannel не отправляет события progress во время воспроизведения. Чтобы регистрировать прогресс воспроизведения, приложение может создать собственный механизм контроля времени и отслеживать позицию точки воспроизведения.

Чтобы вычислить, какой процент звука уже воспроизведен, можно разделить значение свойства `SoundChannel.position` на продолжительность аудиоданных.

```
var playbackPercent:uint = 100 * (channel.position / snd.length);
```

Однако этот код сообщает точные проценты воспроизведения, только если оно началось после полной загрузки аудиоданных. Свойство `Sound.length` показывает размер загруженных на данный момент аудиоданных, а не конечный размер всего аудиофайла. Чтобы отслеживать прогресс воспроизведения потокового звука, загрузка которого еще не завершилась, приложение должно подсчитать конечный размер полного аудиофайла и использовать это значение в вычислениях. Подсчитать конечную продолжительность аудиоданных можно с помощью свойств `bytesLoaded` и `bytesTotal` объекта Sound, как показано ниже.

```
var estimatedLength:int =
    Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
var playbackPercent:uint = 100 * (channel.position / estimatedLength);
```

Следующий код загружает более объемный звуковой файл и использует событие `Event.ENTER_FRAME` в качестве механизма определения времени для отображения прогресса воспроизведения. Код периодически сообщает процент воспроизведенного звука, то есть текущее значение позиции, деленное на общую продолжительность аудиоданных.

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new
    URLRequest("http://av.adobe.com/podcast/csbu_dev_podcast_epi_2.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

function onEnterFrame(event:Event):void
{
    var estimatedLength:int =
        Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
    var playbackPercent:uint =
        Math.round(100 * (channel.position / estimatedLength));
    trace("Sound playback is " + playbackPercent + "% complete.");
}

function onPlaybackComplete(event:Event)
{
    trace("The sound has finished playing.");
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

После начала загрузки аудиоданных код вызывает метод `snd.play()` и сохраняет созданный в результате этого объект SoundChannel в переменной `channel`. Затем для главного приложения добавляется прослушиватель события `Event.ENTER_FRAME`, а для объекта SoundChannel — прослушиватель события `Event.SOUND_COMPLETE`, которое отправляется после завершения воспроизведения.

Каждый раз, когда приложение достигает следующего кадра анимации, вызывается метод `onEnterFrame()`. Метод `onEnterFrame()` рассчитывает общую продолжительность аудиофайла на основе уже загруженного объема данных, а затем вычисляет и отображает текущий процент воспроизведения.

По завершении воспроизведения всего звука выполняется метод `onPlaybackComplete()`, который удаляет прослушиватель события `Event.ENTER_FRAME`, чтобы предотвратить попытки отображения обновлений прогресса после окончания воспроизведения.

Событие `Event.ENTER_FRAME` может отправляться много раз в секунду. Иногда отображать прогресс воспроизведения с такой частотой не требуется. В таких случаях приложение может создать собственный механизм определения времени с помощью класса flash.util.Timer. Сведения см. в разделе «Работа с датами и временем» на странице 1.

### Остановка потоковых звуков

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Процесс воспроизведения потоковых звуков имеет одну особенность: воспроизведение звука начинается до завершения его загрузки. Когда приложение вызывает метод `SoundChannel.stop()` для экземпляра SoundChannel, который воспроизводит потоковый звук, воспроизведение останавливается для одного кадра, а при переходе к следующему кадру оно возобнавляется с начала звука. Это происходит потому, что процесс загрузки звука еще не завершен. Чтобы одновременно остановить и загрузку и воспроизведение потокового звука, нужно вызвать метод `Sound.close()`.

## Безопасность при загрузке и воспроизведении звуков

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Способность приложения получать аудиоданные может быть ограничена в соответствии с моделью безопасности проигрывателя Flash Player или AIR. На каждый звук распространяются ограничения двух разных изолированных программных сред: среды самого содержимого («изолированная программная среда содержимого») и среды приложения или объекта, который загружает и воспроизводит звук («изолированная программная среда владельца»). Если содержимое приложения AIR находится в изолированной программной среде приложения, для него доступны все звуки, включая загруженные из других доменов. Однако к содержимому из других изолированных программных сред применяются те же правила, что и к содержимому в Flash Player. Дополнительные сведения о модели безопасности проигрывателя Flash Player в целом и определения изолированных программных сред см. в разделе ««Безопасность» на странице 1121».

Изолированная программная среда содержимого контролирует возможность извлечения подробных данных звука с помощью свойства `id3` или метода `SoundMixer.computeSpectrum()`. Она не накладывает ограничений на загрузку или воспроизведение звука.

Исходный домен аудиофайла определяет ограничения по безопасности для изолированной программной среды содержимого. Как правило, если аудиофайл размещен в том же домене или папке, что и SWF-файл загружающего приложения или объекта, то приложение или объект будет иметь полный доступ к этому звуку. Если звук размещен в другом домене, то его можно импортировать в изолированную программную среду содержимого с помощью файла политики.

Приложение может передать объект SoundLoaderContext со свойством `checkPolicyFile` в качестве параметра метода `Sound.load()`. Если задать свойству `checkPolicyFile` значение `true`, проигрыватель Flash Player или AIR будет искать файл политики на сервере, с которого загружается звук. Если файл политики существует и предоставляет доступ домену загружающего SWF-файла, то этот SWF-файл сможет загрузить свойство `id3` объекта Sound и вызвать метод `SoundMixer.computeSpectrum()` для загруженных звуков.

Изолированная программная среда владельца контролирует локальное воспроизведение звуков. Изолированная программная среда владельца определяется в зависимости от приложения или объекта, который запускает воспроизведение звука.

Метод `SoundMixer.stopAll()` останавливает все звуки во всех объектах SoundChannel, которые воспроизводятся в данный момент, если они соответствуют следующим критериям:

- воспроизведение звуков было запущено объектами из той же изолированной программной среды владельца;

- звуки размещены на исходном сервере, на котором есть файл политики, предоставляющий доступ домену приложения или объекта, который вызывает метод `SoundMixer.stopAll()`.

Однако в AIR эти ограничения по безопасности не распространяются на содержимое в изолированной программной среде приложения (содержимое, установленное с приложением AIR).

Чтобы выяснить, сможет ли метод `SoundMixer.stopAll()` остановить воспроизведение всех звуков, приложение должно вызвать метод `SoundMixer.areSoundsInaccessible()`. Если этот метод возвращает значение `true`, некоторые воспроизводимые звуки находятся за пределами текущей изолированной программной среды владельца и не могут быть остановлены методом `SoundMixer.stopAll()`.

Метод `SoundMixer.stopAll()` также останавливает точку воспроизведения для всех звуков, загруженных из внешних файлов. Однако звуки, встроенные в FLA-файл и прикрепленные к кадрам временной шкалы с помощью инструмента разработки Flash, могут возобновиться, если анимация перейдет к следующему кадру.

# Управление громкостью и панорамированием звука

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Индивидуальный объект SoundChannel управляет обоими стереоканалами звука, левым и правым. Если mp3-файл имеет монофоническое звучание, то левый и правый стереоканалы объекта SoundChannel будут содержать идентичные звуковые волны.

Амплитуду каждого стереоканала звука можно узнать с помощью свойств `leftPeak` и `rightPeak` объекта SoundChannel. Эти свойства показывают максимальную амплитуду самой звуковой волны. Они не представляют фактическую громкость воспроизведения. Фактическая громкость воспроизведения является функцией значений амплитуды звуковой волны и громкости, заданных в объекте SoundChannel и классе SoundMixer.

Свойство pan объекта SoundChannel можно использовать для установки разных уровней громкости для левого и правого каналов во время воспроизведения. Свойство pan может иметь значение от -1 до 1, где при значении -1 левому каналу задается максимальная громкость, а правый заглушается, а при значении 1 правому каналу назначается максимальная громкость, а левый заглушается. Числовые значения от -1 до 1 устанавливаются пропорционально для левого и правого канала, а значение 0 указывает на то, что оба канала сбалансированы и имеют среднюю громкость.

Следующий код создает объект SoundTransform с громкостью на уровне 0,6 и панорамой -1 (максимальная громкость левого канала и заглушенный правый канал). Объект SoundTransform передается в качестве параметра для метода `play()`, который применяет его к новому объекту SoundChannel, созданному для управления воспроизведением.

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var trans:SoundTransform = new SoundTransform(0.6, -1);
var channel:SoundChannel = snd.play(0, 1, trans);
```

Громкость и панорамирование звука можно изменять во время воспроизведения, задавая требуемые значения свойства `pan` или `volume` объекта SoundTransform и затем применяя этот объект в качестве свойства `soundTransform` объекта SoundChannel.

Также можно задать глобальные значения громкости и панорамы для всех звуков сразу с помощью свойства `soundTransform` класса SoundMixer, как в следующем примере.

```
SoundMixer.soundTransform = new SoundTransform(1, -1);
```

Кроме того, объект SoundTransform можно использовать для настройки значений громкости и панорамы для объекта Microphone (см. раздел «Захват вводимого звука» на странице 495), а также для объектов Sprite и SimpleButton.

В следующем примере панорамирования звук переходит с левого канала на правый и обратно во время воспроизведения.

```
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var panCounter:Number = 0;

var trans:SoundTransform;
trans = new SoundTransform(1, 0);
var channel:SoundChannel = snd.play(0, 1, trans);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

addEventListener(Event.ENTER_FRAME, onEnterFrame);

function onEnterFrame(event:Event):void
{
    trans.pan = Math.sin(panCounter);
    channel.soundTransform = trans; // or SoundMixer.soundTransform = trans;
    panCounter += 0.05;
}

function onPlaybackComplete(event:Event):void
{
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

Этот код сначала загружает аудиофайл, а затем создает новый объект SoundTransform с громкостью 1 (полная громкость) и панорамой 0 (одинаковая громкость левого и правого каналов). Затем вызывается метод `snd.play()`, которому в качестве параметра передается объект SoundTransform.

В процессе воспроизведения звука многократно выполняется метод `onEnterFrame()`. Метод `onEnterFrame()` использует функцию `Math.sin()`, чтобы генерировать значение от -1 до 1. Этот диапазон соответствует допустимым значениям для свойства `SoundTransform.pan`. Свойство `pan` объекта SoundTransform получает новое значение, и после этого измененный объект назначается свойству `soundTransform` канала.

Чтобы протестировать этот пример, замените имя файла bigSound.mp3 на имя локального mp3-файла. Затем запустите пример. Вы услышите, как левый канал становится громче, а правый тише и наоборот.

В этом примере того же эффекта можно достичь, определив свойство `soundTransform` класса SoundMixer. Однако это затронуло бы панорамирование всех воспроизводимых в данный момент звуков, а не только звук, воспроизводимый данным объектом SoundChannel.

# Работа с метаданными звука

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Аудиофайлы в формате mp3 могут содержать дополнительные данные о звуке в виде тегов ID3.

Не каждый mp3-файл содержит метаданные ID3. Когда объект Sound загружает аудиофайл в формате mp3, он отправляет событие `Event.ID3`, если файл содержит метаданные ID3. Чтобы избежать ошибок выполнения, приложение должно дождаться получения события `Event.ID3`, прежде чем обращаться к свойству `Sound.id3` загруженного звука.

Следующий код демонстрирует, как можно определить, загружены ли метаданные ID3 для аудиофайла.

```
import flash.events.Event;
import flash.media.ID3Info;
import flash.media.Sound;

var s:Sound = new Sound();
s.addEventListener(Event.ID3, onID3InfoReceived);
s.load("mySound.mp3");

function onID3InfoReceived(event:Event)
{
    var id3:ID3Info = event.target.id3;

    trace("Received ID3 Info:");
    for (var propName:String in id3)
    {
        trace(propName + " = " + id3[propName]);
    }
}
```

Сначала код создает объект Sound и регистрирует для него прослушиватель события `Event.ID3`. После завершения загрузки метаданных ID3 аудиофайла вызывается метод `onID3InfoReceived()`. Целью объекта Event, переданного методу `onID3InfoReceived()` является исходный объект Sound, поэтому метод получает его свойство `id3`, а затем проходит через все названные свойства, чтобы отследить их значения.

# Получение необработанных аудиоданных

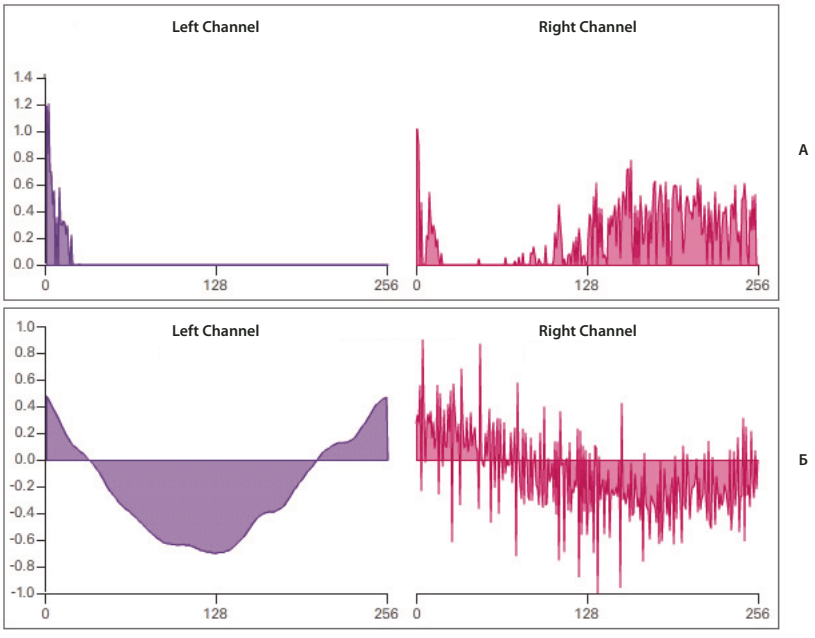
**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `SoundMixer.computeSpectrum()` позволяет приложению считывать необработанные аудиоданные для отображения воспроизводимой в данный момент звуковой волны. Если одновременно воспроизводится несколько объектов SoundChannel, метод `SoundMixer.computeSpectrum()` показывает комбинированные аудиоданные для всех смикшированных вместе объектов SoundChannel.

Аудиоданные возвращаются в виде объекта ByteArray, содержащего 512 байтов данных, каждый из которых содержит значение с плавающей запятой в диапазоне от -1 до 1. Эти значения представляют амплитуду точек воспроизводимой звуковой волны. Значения передаются двумя группами, по 256 в каждой: первая группа предназначена для левого стереоканала, а вторая — для правого.

Если параметр `FFTMode` имеет значение `true`, метод `SoundMixer.computeSpectrum()` возвращает данные спектра частот, а не формы волны. Спектр частот показывает амплитуду по частоте звука: от самой низкой до самой высокой. Алгоритм быстрого преобразования Фурье (FFT) используется для преобразования данных звуковой волны в данные спектра частот. Полученные значения спектра частот находятся в диапазоне от 0 приблизительно до 1,414 (квадратный корень из 2).

На примере следующих графиков сравниваются данные, возвращенные методом `computeSpectrum()` с параметром `FFTMode` в значении `true` и в значении `false`. Звук, данные которого представлены на графиках, содержит громкий бас в левом канале и удар барабана в правом канале.




*Значения, возвращенные методом SoundMixer.computeSpectrum()*
***А.** fftMode=true **Б.** fftMode=false*

Метод `computeSpectrum()` может также возвращать данные, преобразованные с использованием более низкой скорости передачи битов. Как правило, это приводит к сглаживанию данных звуковой волны или частоты за счет их прореживания. Параметр `stretchFactor` управляет частотой дискретизации, используемой для данных метода `computeSpectrum()`. Когда параметр `stretchFactor` имеет значение 0 (по умолчанию), для звука используется частота дискретизации 44,1 кГц. Эта частота уменьшается вдвое для каждого последующего значения параметра stretchFactor, поэтому значению 1 соответствует частота 22,05 кГц, а значению 2 — 11,025 кГц и т.д. Метод `computeSpectrum()` все равно возвращает по 256 байтов для каждого стереоканала, когда используются более высокие значения для параметра`stretchFactor`.

Метод `SoundMixer.computeSpectrum()` имеет некоторые ограничения.

- Так как аудиоданные с микрофона или с RTMP-сервера не проходят через глобальный объект SoundMixer, метод `SoundMixer.computeSpectrum()` не возвращает данных из этих источников.
- Если один или несколько из воспроизводимых звуков находятся за пределами текущей изолированной программной среды содержимого, метод `SoundMixer.computeSpectrum()` выдаст ошибку из-за ограничений по безопасности. Дополнительные сведения об ограничениях по безопасности для метода `SoundMixer.computeSpectrum()` см. в разделах ««Безопасность при загрузке и воспроизведении звуков» на странице 488» и ««Получение данных загруженного мультимедийного содержимого» на странице 1149».

Однако в AIR эти ограничения по безопасности не распространяются на содержимое в изолированной программной среде приложения (содержимое, установленное с приложением AIR).

## Создание простого визуализатора звука

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

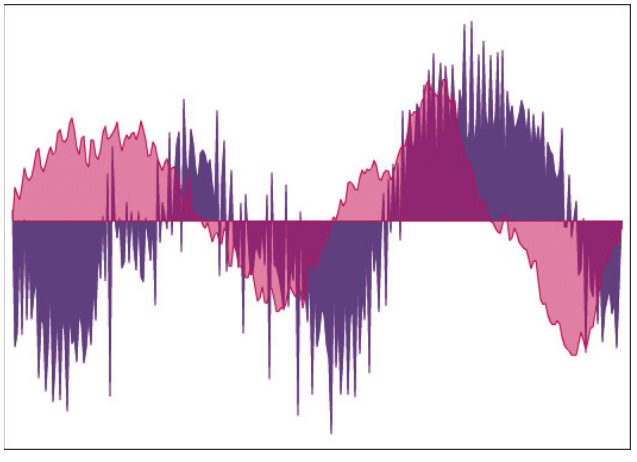
В следующем примере используется метод `SoundMixer.computeSpectrum()` для отображения зрительного образа звука с анимацией каждого кадра.

```
import flash.display.Graphics;
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

const PLOT_HEIGHT:int = 200;
const CHANNEL_LENGTH:int = 256;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
snd.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

var bytes:ByteArray = new ByteArray();

function onEnterFrame(event:Event):void
{
    SoundMixer.computeSpectrum(bytes, false, 0);

    var g:Graphics = this.graphics;

    g.clear();
    g.lineStyle(0, 0x6600CC);
    g.beginFill(0x6600CC);
    g.moveTo(0, PLOT_HEIGHT);

    var n:Number = 0;

    // left channel
    for (var i:int = 0; i < CHANNEL_LENGTH; i++)
    {
        n = (bytes.readFloat() * PLOT_HEIGHT);
        g.lineTo(i * 2, PLOT_HEIGHT - n);
    }
    g.lineTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);
```

```
    g.endFill();

    // right channel
    g.lineStyle(0, 0xCC0066);
    g.beginFill(0xCC0066, 0.5);
    g.moveTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);

    for (i = CHANNEL_LENGTH; i > 0; i--)
    {
        n = (bytes.readFloat() * PLOT_HEIGHT);
        g.lineTo(i * 2, PLOT_HEIGHT - n);
    }
    g.lineTo(0, PLOT_HEIGHT);
    g.endFill();
}

function onPlaybackComplete(event:Event)
{
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

Сначала код загружает аудиофайл и запускает его воспроизведение, а затем прослушивает событие `Event.ENTER_FRAME`, которое запускает метод `onEnterFrame()` во время воспроизведения звука. Метод `onEnterFrame()` вызывает метод `SoundMixer.computeSpectrum()`, который сохраняет данные звуковой волны в объекте ByteArray с именем `bytes`.

Зрительный образ звука отображается с использованием API-интерфейса векторного рисования. Первый цикл `for` считывает первые 256 значений (левый стереоканал) и рисует линию от одной точки к другой с помощью метода `Graphics.lineTo()`. Второй цикл `for` считывает следующие 256 значений и рисует линию, но на этот раз в обратном порядке, справа налево. Полученные зрительные образы звука могут дать интересный эффект зеркального отражения, как показано на следующем рисунке.



# Захват вводимого звука

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс Microphone позволяет приложению подключаться к микрофону или другому устройству ввода звука, установленному в пользовательской системе, и затем транслировать вводимый звук на системные динамики или отправлять его на удаленный сервер, такой как Flash Media Server. Можно получить доступ к необработанной звуковой информации с микрофона для выполнения записи или обработки. Звук также можно направлять напрямую на системные динамики или передавать сжатую звуковую информацию на удаленный сервер. Для данных, отправленных на удаленный сервер, можно использовать кодек Speex или Nellymoser. (Кодек Speex доступен для использования, начиная с Flash Player версии 10 и Adobe AIR версии 1.5.)

### Дополнительные разделы справки

Микаэль Шез (Michael Chaize): AIR, Android и микрофон

Кристоф Конрац (Christophe Coenraets): Голосовые заметки для платформы Android

## Получение доступа к микрофону

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс Microphone не имеет собственного конструктора. Вместо него используется статический метод `Microphone.getMicrophone()`, который позволяет получить новый экземпляр Microphone, как показано ниже.

```
var mic:Microphone = Microphone.getMicrophone();
```

Вызов метода `Microphone.getMicrophone()` без параметра возвращает первое устройство ввода звука, обнаруженное в системе пользователя.

В системе может быть установлено несколько устройств ввода звука. Приложение может использовать свойство `Microphone.names`, чтобы получить массив имен всех доступных устройств ввода звука. Затем оно может вызвать метод `Microphone.getMicrophone()` с параметром `index`, значение которого соответствует позиции индекса нужного устройства в массиве.

К системе может быть не подключен ни микрофон, ни другое устройство ввода звука. С помощью свойства `Microphone.names` или метода `Microphone.getMicrophone()` можно проверить, установлено ли устройство ввода звука в системе пользователя. Если пользователь не установил устройство ввода звука, массив `names` будет иметь нулевую длину, а метод `getMicrophone()` возвращает значение `null`.

Когда приложение вызывает метод `Microphone.getMicrophone()`, проигрыватель Flash Player выводит окно «Параметры Flash Player», в котором пользователь может разрешить или запретить проигрывателю Flash Player доступ к камере или микрофону в системе. После того, как пользователь нажмет кнопку «Разрешить» или «Запретить» в этом окне, отправляется событие StatusEvent. Свойство `code` этого экземпляра StatusEvent указывает, разрешен или запрещен доступ к микрофону, как показано в следующем примере.

```
import flash.media.Microphone;

var mic:Microphone = Microphone.getMicrophone();
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

function onMicStatus(event:StatusEvent):void
{
    if (event.code == "Microphone.Unmuted")
    {
        trace("Microphone access was allowed.");
    }
    else if (event.code == "Microphone.Muted")
    {
        trace("Microphone access was denied.");
    }
}
```

Свойство `StatusEvent.code` будет содержать значение «Microphone.Unmuted», если доступ разрешен, или «Microphone.Muted», если доступ запрещен.

Свойство `Microphone.muted` получает значение `true` или `false`, когда пользователь соответственно разрешает или запрещает доступ к микрофону. Однако свойство `muted` задается для экземпляра Microphone только после отправки события StatusEvent, поэтому приложение также должно дождаться отправки события `StatusEvent.STATUS`, прежде чем проверять свойство `Microphone.muted`.

Для того чтобы в проигрывателе Flash Player было показано диалоговое окно настройки, окно приложения должно быть достаточно большим для его отображения (не менее 215 x 138 пикселов). В противном случае доступ автоматически запрещается.

Содержимому, выполняемому в изолированной программной среде приложения AIR, не требуется разрешение пользователя для получения доступа к микрофону. Таким образом, события status при отключении и включении звука микрофона никогда не отправляются. Содержимому, которое выполняется в AIR за пределами изолированной программной среды приложения, требуется разрешение пользователя, поэтому отправка событий status возможна.

## Трансляция звука с микрофона на локальные динамики

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Звук, вводимый с микрофона, можно транслировать на динамики локальной системы с помощью метода `Microphone.setLoopback()`, параметр которого имеет значение `true`.

Когда звук с локального микрофона транслируется на локальные динамики, существует риск создания эффекта обратной связи, который может вызвать громкие свистящие звуки и даже повредить звуковое оборудование. Вызов метода `Microphone.setUseEchoSuppression()` со значением параметра `true` уменьшает, но не устраняет полностью риск возникновения эффекта обратной связи. Компания Adobe рекомендует всегда вызывать метод `Microphone.setUseEchoSuppression(true)` перед вызовом `Microphone.setLoopback(true)`, если нет уверенности в том, что пользователь воспроизводит звук не через динамики, а с помощью наушников или другого устройства.

Следующий код демонстрирует трансляцию звука с локального микрофона на динамики локальной системы.

```
var mic:Microphone = Microphone.getMicrophone();
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
```

## Изменение звука, захватываемого с микрофона

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Приложение может изменять аудиоданные, поступающие с микрофона, двумя способами. Во-первых, оно может изменять уровень усиления вводимого звука, умножая вводимые значения на заданный коэффициент, чтобы сделать звук громче или тише. Свойство `Microphone.gain` принимает числовые значения от 0 до 100 включительно. Значение 50 соответствует множителю 1 и обозначает нормальную громкость. Значение 0 соответствует множителю 0 и заглушает вводимый звук. Значения выше 50 делают звук громче.

Приложение также может изменять частоту дискретизации вводимого звука. Чем выше частота дискретизации, тем выше качество звука, но при этом также уплотняются потоки данных, которые требуют больше ресурсов для передачи и хранения. Свойство `Microphone.rate` представляет частоту дискретизации звука в килогерцах (кГц). Частота дискретизации по умолчанию составляет 8 кГц. Если микрофон поддерживает более высокую частоту, можно задать для свойства `Microphone.rate` значение больше 8 кГц. Например, если задать свойству `Microphone.rate` значение 11, то частота дискретизации будет составлять 11 кГц, а при значении 22 частота будет равна 22 кГц. Доступная частота дискретизации зависит от выбранного кодека. При использовании кодека Nellymoser можно выбрать частоту дискретизации, равную 5, 8, 11, 16, 22 или 44 кГц. При использовании кодека Speex (доступен, начиная с Flash Player версии 10 и Adobe AIR версии 1.5) вы можете выбрать только частоту 16 кГц.

## Определение активности микрофона

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

С целью экономии пропускной способности и ресурсов обработки проигрыватель Flash Player пытается определить, когда микрофон не передает звука. Если уровень активности микрофона остается ниже порога тишины в течение определенного времени, проигрыватель Flash Player перестает передавать вводимый звук и отправляет простое событие ActivityEvent. При использовании кодека Speex (доступен, начиная с Flash Player версии 10 и Adobe AIR версии 1.5) установите уровень тишины на нулевое значение, чтобы приложение осуществляло непрерывную передачу звука. Система обнаружения речевой активности Speex автоматически снижает потребление пропускной способности.

***Примечание.*** *Объект Microphone отправляет события activity, только когда приложение отслеживает микрофон. Таким образом, если функция `setLoopBack( true )` не вызывается, добавьте прослушиватель для событий звуковых данных сэмплов или подключите микрофон к объекту NetStream. В этом случае события activity не отправляются.*

Для отслеживания и контроля активности в классе Microphone предусмотрено три свойства.

- Свойство `activityLevel`, предназначенное только для чтения, указывает количество звука, определяемое микрофоном, по шкале от 0 до 100.
- Свойство `silenceLevel` указывает количество звука, необходимое для активации микрофона и отправки события `ActivityEvent.ACTIVITY`. Свойство `silenceLevel` также использует шкалу от 0 до 100, по умолчанию используется значение 10.
- Свойство `silenceTimeout` указывает количество миллисекунд, в течение которых уровень активности должен находиться ниже порога тишины, прежде чем будет отправлено событие `ActivityEvent.ACTIVITY`, сообщающее о том, что микрофон заглушен. По умолчанию свойство `silenceTimeout` имеет значение 2000.

И свойство `Microphone.silenceLevel`, и свойство `Microphone.silenceTimeout` предназначены только для чтения, но их значения можно изменить с помощью метода `Microphone.setSilenceLevel()`.

В некоторых случаях процесс активации микрофона при обнаружении новой активности может вызывать небольшую задержку. Постоянное поддержание микрофона в активном состоянии предотвращает такие задержки при активации. Приложение может вызвать метод `Microphone.setSilenceLevel()` с параметром `silenceLevel` в значении 0, чтобы проигрыватель Flash Player поддерживал микрофон в активном состоянии и продолжал собирать аудиоданные, даже когда звук не определяется. И наоборот, если задать параметру `silenceLevel` значение 100, микрофон вообще не будет активироваться.

В следующем примере показана информация о микрофоне, а также сообщения о событиях activity и status, отправленных объектом Microphone.

```
import flash.events.ActivityEvent;
import flash.events.StatusEvent;
import flash.media.Microphone;

var deviceArray:Array = Microphone.names;
trace("Available sound input devices:");
for (var i:int = 0; i < deviceArray.length; i++)
{
    trace(" " + deviceArray[i]);
}

var mic:Microphone = Microphone.getMicrophone();
mic.gain = 60;
mic.rate = 11;
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
mic.setSilenceLevel(5, 1000);

mic.addEventListener(ActivityEvent.ACTIVITY, this.onMicActivity);
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

var micDetails:String = "Sound input device name: " + mic.name + '\n';
micDetails += "Gain: " + mic.gain + '\n';
micDetails += "Rate: " + mic.rate + " kHz" + '\n';
micDetails += "Muted: " + mic.muted + '\n';
micDetails += "Silence level: " + mic.silenceLevel + '\n';
micDetails += "Silence timeout: " + mic.silenceTimeout + '\n';
micDetails += "Echo suppression: " + mic.useEchoSuppression + '\n';
trace(micDetails);

function onMicActivity(event:ActivityEvent):void
{
    trace("activating=" + event.activating + ", activityLevel=" +
        mic.activityLevel);
}

function onMicStatus(event:StatusEvent):void
{
    trace("status: level=" + event.level + ", code=" + event.code);
}
```

При тестировании этого примера говорите в микрофон или создавайте шумы, чтобы понаблюдать за значениями инструкции trace на консоли или в окне отладки.

## Отправка звука на сервер мультимедийного содержимого и с сервера

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Когда ActionScript используется с потоковым сервером мультимедийного содержимого, таким как Flash Media Server, становятся доступны дополнительные возможности.

В частности, приложение может присоединить объект Microphone к объекту NetStream и передавать данные непосредственно с микрофона пользователя на сервер. Аудиоданные также можно передавать потоком с сервера приложению и воспроизводить в составе объекта MovieClip или с помощью объекта Video.

Кодек Speex доступен, начиная с Flash Player версии 10 и Adobe AIR версии 1.5. Для выбора кодека, используемого для сжатия отправляемых на медиа-сервер аудиоданных, настройте свойство `codec` объекта Microphone. Это свойство может иметь два значения, которые перечислены в классе SoundCodec. Если свойству codec задано значение `SoundCodec.SPEEX`, для сжатия аудио выбирается кодек Speex. Если свойству задано значение `SoundCodec.NELLYMOSER` (по умолчанию), для сжатия аудио выбирается кодек Nellymoser.

Дополнительные сведения см. в интерактивной документации по Flash Media Server на странице www.adobe.com/go/learn_fms_docs_ru.

## Захват аудиоданных с микрофона

**Flash Player 10.1 и более поздних версий, Adobe AIR 2 и более поздних версий**

В программах Flash Player 10.1 и AIR 2 и более поздних версиях информацию можно захватывать из данных микрофона в качестве байтового массива значений с плавающей запятой. Каждое значение представляет сэмпл монофонических аудиоданных.

Чтобы получить микрофонные данные, задайте прослушиватель событий для события `sampleData` объекта Microphone. Объект Microphone периодически отправляет события `sampleData` по мере того, как буфер микрофона заполняется звуковыми сэмплами. Объект SampleDataEvent обладает свойством `data`, которое представляет собой байтовый массив звуковых сэмплов. Каждый сэмпл представлен значениями с плавающей запятой, которые относятся к монофоническому звуковому сэмплу.

Следующий код захватывает звуковые данные с микрофона в объект ByteArray под названием `soundBytes`:

```
var mic:Microphone = Microphone.getMicrophone();
mic.setSilenceLevel(0, DELAY_LENGTH);
mic.addEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);
function micSampleDataHandler(event:SampleDataEvent):void {
    while(event.data.bytesAvailable) {
        var sample:Number = event.data.readFloat();
        soundBytes.writeFloat(sample);
    }
}
```

Байты сэмпла можно повторно использовать в качестве аудио для воспроизведения в объекте Sound. При этом необходимо присвоить свойству `rate` объекта Microphone значение 44, которое соответствует частоте дискретизации, используемой объектами Sound. Можно также преобразовать микрофонные сэмплы, захваченные при низкой частоте, в частоту 44 кГц, которая требуется объекту Sound. Учтите также, что объект Microphone захватывает монофонические сэмплы, тогда как объект Sound использует стереозвук. Поэтому каждый из байтов, захваченных объектом Microphone, нужно записывать в объект Sound дважды. В следующем примере 4 секунды микрофонных данных захватываются и воспроизводятся с помощью объекта Sound:

```
const DELAY_LENGTH:int = 4000;
var mic:Microphone = Microphone.getMicrophone();
mic.setSilenceLevel(0, DELAY_LENGTH);
mic.gain = 100;
mic.rate = 44;
mic.addEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);

var timer:Timer = new Timer(DELAY_LENGTH);
timer.addEventListener(TimerEvent.TIMER, timerHandler);
timer.start();

function micSampleDataHandler(event:SampleDataEvent):void
{
    while(event.data.bytesAvailable)
    {
        var sample:Number = event.data.readFloat();
        soundBytes.writeFloat(sample);
    }
}
var sound:Sound = new Sound();
var channel:SoundChannel;
function timerHandler(event:TimerEvent):void
{
    mic.removeEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);
    timer.stop();
    soundBytes.position = 0;
    sound.addEventListener(SampleDataEvent.SAMPLE_DATA, playbackSampleHandler);
    channel.addEventListener( Event.SOUND_COMPLETE, playbackComplete );
    channel = sound.play();
}

function playbackSampleHandler(event:SampleDataEvent):void
{
    for (var i:int = 0; i < 8192 && soundBytes.bytesAvailable > 0; i++)
    {
        trace(sample);
        var sample:Number = soundBytes.readFloat();
        event.data.writeFloat(sample);
        event.data.writeFloat(sample);
    }
}

function playbackComplete( event:Event ):void
{
    trace( "Playback finished.");
}
```

Дополнительные сведения о воспроизведении звуков из данных звукового сэмпла см. в разделе ««Работа с динамически создаваемым звуком» на странице 482».

# Пример звука: Podcast Player

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Подкаст — это звуковой файл, распространяемый через Интернет по требованию или по подписке. Как правило, подкасты публикуются по сериям, которые также называются каналами подкастов. Так как эпизоды подкастов могут иметь продолжительность от одной минуты до многих часов, для них обычно используется потоковое воспроизведение. Эпизоды подкастов, которые также называются элементами, обычно передаются как файлы в формате mp3. Видеоподкасты также популярны, но это приложение воспроизводит только аудиоподкасты, использующие mp3-файлы.

Этот пример не является полнофункциональным приложением для агрегации подкастов. Например, оно не управляет подписками на определенные подкасты и не запоминает прослушанные пользователем подкасты до следующего запуска приложения. Его можно использовать как отправную точку для создания более функционального агрегатора подкастов.

Пример PodcastPlayer демонстрирует следующие приемы программирования на ActionScript:

- чтение внешнего канала RSS и анализ его XML-содержимого;
- создание класса SoundFacade для упрощения загрузки и воспроизведения звуковых файлов;
- отображение прогресса воспроизведения звука;
- приостановка и возобновление воспроизведения звука.

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения Podcast Player находятся в папке Samples/PodcastPlayer. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| PodcastPlayer.mxml<br>или<br>PodcastPlayer.fla | Пользовательский интерфейс приложения для Flex (MXML) или Flash (FLA). |
| comp/example/programmingas3/podcastplayer/PodcastPlayer.as | Класс документа, отвечающий за поведение интерфейса пользователя проигрывателя подкастов (только Flash). |
| SoundPlayer.mxml | Компонент MXML, который отображает кнопки воспроизведения и полосы прогресса, а также управляет воспроизведением звука, только для Flex. |
| main.css | Стили пользовательского интерфейса приложения (только Flex). |
| images/ | Значки стиля кнопок (только Flex). |
| comp/example/programmingas3/podcastplayer/SoundPlayer.as | Класс символа фрагмента ролика SoundPlayer, отвечающий за поведение интерфейса пользователя проигрывателя (только Flash). |
| comp/example/programmingas3/podcastplayer/PlayButtonRenderer.as | Заказное средство визуализации ячеек, используемое для отображения кнопки воспроизведения в ячейке DataGrid (только Flash). |
| com/example/programmingas3/podcastplayer/RSSBase.as | Базовый класс, который обеспечивает общие свойства и методы для классов RSSChannel и RSSItem. |

| File | Описание |
|---|---|
| com/example/programmingas3/podcastplayer/RSSChannel.as | Класс ActionScript, который содержит данные о канале RSS. |
| com/example/programmingas3/podcastplayer/RSSItem.as | Класс ActionScript, который содержит данные об элементе RSS. |
| com/example/programmingas3/podcastplayer/SoundFacade.as | Основной класс ActionScript для приложения. Он содержит методы и события классов Sound и SoundChannel и добавляет поддержку для приостановки и возобновления воспроизведения. |
| com/example/programmingas3/podcastplayer/URLService.as | Класс ActionScript, который получает данные с удаленного URL-адреса. |
| playerconfig.xml | XML-файл, который содержит список каналов RSS, являющихся каналами подкастов. |
| comp/example/programmingas3/utils/DateUtil.as | Класс, используемый для упрощенного форматирования даты (только Flash). |

## Считывание данных RSS для канала подкастов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Приложение Podcast Player начинает работу со считывания информации о количестве каналов подкастов и их эпизодов.

1. Сначала приложение считывает XML-файл конфигурации, который содержит список каналов подкастов и отображает его пользователю.

2. Когда пользователь выбирает один из каналов подкастов, приложение считывает канал RSS для данного канала и отображает список его эпизодов.

В этом примере используется класс URLLoader для получения текстовых данных из удаленного местоположения или локального файла. Проигрыватель Podcast Player сначала создает объект URLLoader, чтобы получить список каналов RSS в формате XML из файла playerconfig.xml. Затем, когда пользователь выбирает из списка определенный канал, создается новый объект URLLoader для считывания данных RSS с URL-адреса этого канала.

## Упрощение загрузки и воспроизведения звука с помощью класса SoundFacade

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Архитектура звука в ActionScript 3.0 отличается мощностью и при этом сложностью. Приложения, в которых требуются только базовые функции загрузки и воспроизведения звука, могут использовать класс, который частично скрывает сложности, обеспечивая более простой набор методов и событий. В области шаблонов программного дизайна такой класс называется *фасадом*.

Класс SoundFacade представляет единый интерфейс для выполнения следующих задач:

- загрузка аудиофайлов с помощью объектов Sound, SoundLoaderContext и класса SoundMixer;
- воспроизведение звуковых файлов с помощью объектов Sound и SoundChannel;

- отправка событий о прогрессе воспроизведения;
- приостановка и возобновление воспроизведения звука с помощью объектов Sound и SoundChannel.

Класс SoundFacade пытается обеспечить максимальную функциональность звуковых классов ActionScript, при этом упрощая работу.

Следующий код демонстрирует объявление класса, свойства класса и конструктор `SoundFacade()`.

```
public class SoundFacade extends EventDispatcher
{
    public var s:Sound;
    public var sc:SoundChannel;
    public var url:String;
    public var bufferTime:int = 1000;

    public var isLoaded:Boolean = false;
    public var isReadyToPlay:Boolean = false;
    public var isPlaying:Boolean = false;
    public var isStreaming:Boolean = true;
    public var autoLoad:Boolean = true;
    public var autoPlay:Boolean = true;

    public var pausePosition:int = 0;

    public static const PLAY_PROGRESS:String = "playProgress";
    public var progressInterval:int = 1000;
    public var playTimer:Timer;

    public function SoundFacade(soundUrl:String, autoLoad:Boolean = true,
                                autoPlay:Boolean = true, streaming:Boolean = true,
                                bufferTime:int = -1):void
    {
        this.url = soundUrl;

        // Sets Boolean values that determine the behavior of this object
        this.autoLoad = autoLoad;
        this.autoPlay = autoPlay;
        this.isStreaming = streaming;

        // Defaults to the global bufferTime value
        if (bufferTime < 0)
        {
            bufferTime = SoundMixer.bufferTime;
        }

        // Keeps buffer time reasonable, between 0 and 30 seconds
        this.bufferTime = Math.min(Math.max(0, bufferTime), 30000);

        if (autoLoad)
        {
            load();
        }
    }
```

Класс SoundFacade расширяет класс EventDispatcher, чтобы иметь возможность отправлять собственные события. Сначала код класса объявляет свойства для объектов Sound и SoundChannel. Класс также сохраняет значение URL-адреса аудиофайла и свойство `bufferTime` для использования при потоковом воспроизведении звука. Кроме того, он принимает несколько логических значений параметров, которые определяют поведение при загрузке и воспроизведении.

- Параметр `autoLoad` сообщает объекту о том, что загрузка звука должна начаться сразу после создания этого объекта.
- Параметр `autoPlay` указывает на то, что воспроизведение звука должно начаться, как только будет загружен достаточный объем аудиоданных. Если звук потоковый, воспроизведение начнется, как только будет загружен достаточный объем данных, заданный в свойстве `bufferTime`.
- Параметр `streaming` указывает на то, что воспроизведение этого аудиофайла может начаться до завершения загрузки.

Параметр `bufferTime` имеет значение по умолчанию -1. Если конструктор обнаруживает отрицательное значение в параметре `bufferTime`, то задает свойству `bufferTime` значение `SoundMixer.bufferTime`. Благодаря этому приложение может использовать глобальное значение `SoundMixer.bufferTime` в соответствии с потребностями.

Если параметр `autoLoad` имеет значение `true`, конструктор немедленно вызывает следующий метод `load()` для загрузки аудиофайла.

```
public function load():void
{
    if (this.isPlaying)
    {
        this.stop();
        this.s.close();
    }
    this.isLoaded = false;

    this.s = new Sound();

    this.s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
    this.s.addEventListener(Event.OPEN, onLoadOpen);
    this.s.addEventListener(Event.COMPLETE, onLoadComplete);
    this.s.addEventListener(Event.ID3, onID3);
    this.s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
    this.s.addEventListener(SecurityErrorEvent.SECURITY_ERROR, onIOError);

    var req:URLRequest = new URLRequest(this.url);

    var context:SoundLoaderContext = new SoundLoaderContext(this.bufferTime, true);
    this.s.load(req, context);
}
```

Метод `load()` создает новый объект Sound и затем добавляет прослушиватели для всех важных событий звука. Затем он предписывает объекту Sound загрузить звук, используя объект SoundLoaderContext для передачи значения `bufferTime`.

Так как свойство `url` может быть изменено, экземпляр SoundFacade может использоваться для воспроизведения последовательности разных звуковых файлов: для этого нужно просто изменить свойство `url` и вызвать метод `load()`, чтобы загрузить новый звук.

Три следующих прослушивателя событий демонстрируют, как объект SoundFacade отслеживает прогресс загрузки и определяет время для начала воспроизведения звука.

```
public function onLoadOpen(event:Event):void
{
    if (this.isStreaming)
    {
        this.isReadyToPlay = true;
        if (autoPlay)
        {
            this.play();
        }
    }
    this.dispatchEvent(event.clone());
}

public function onLoadProgress(event:ProgressEvent):void
{
    this.dispatchEvent(event.clone());
}

public function onLoadComplete(event:Event):void
{
    this.isReadyToPlay = true;
    this.isLoaded = true;
    this.dispatchEvent(evt.clone());

    if (autoPlay && !isPlaying)
    {
        play();
    }
}
```

Метод `onLoadOpen()` выполняется после начала загрузки звука. Если звук может воспроизводиться в потоковом режиме, метод `onLoadComplete()` сразу устанавливает для флага `isReadyToPlay` значение `true`. Флаг `isReadyToPlay` определяет, когда приложение может начать воспроизведение звука, возможно, в ответ на действие пользователя, например нажатие кнопки «Play» (воспроизведение). Класс SoundChannel управляет буферизацией аудиоданных, поэтому не требуется специально проверять, загружен ли достаточный объем данных, перед вызовом метода `play()`.

Метод `onLoadProgress()` выполняется периодически в процессе загрузки. Он просто отправляет клон своего объекта ProgressEvent для кода, использующего объект SoundFacade.

После завершения загрузки аудиоданных выполняется метод `onLoadComplete()`, который при необходимости вызывает метод `play()` для непотоковых звуков. Сам метод `play()` приводится ниже.

```
public function play(pos:int = 0):void
{
    if (!this.isPlaying)
    {
        if (this.isReadyToPlay)
        {
            this.sc = this.s.play(pos);
            this.sc.addEventListener(Event.SOUND_COMPLETE, onPlayComplete);
            this.isPlaying = true;

            this.playTimer = new Timer(this.progressInterval);
            this.playTimer.addEventListener(TimerEvent.TIMER, onPlayTimer);
            this.playTimer.start();
        }
    }
}
```

Метод `play()` вызывает метод `Sound.play()`, если звук готов для воспроизведения. Созданный в результате этого объект SoundChannel сохраняется в свойстве `sc`. Затем метод `play()` создает объект Timer, который будет использоваться для отправки событий прогресса воспроизведения через регулярные интервалы.

## Отображение прогресса воспроизведения звука

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Создание объекта Timer для отслеживания воспроизведения — это сложная операция, но этот код нужно написать только один раз. Инкапсуляция этой логики Timer в неоднократно используемом классе, таком как SoundFacade, позволяет приложениям прослушивать те же типы событий прогресса как во время загрузки звука, так и во время его воспроизведения.

Объект Timer, созданный методом `SoundFacade.play()`, каждую секунду отправляет экземпляр TimerEvent. Следующий метод `onPlayTimer()` выполняется каждый раз при получении нового события TimerEvent.

```
public function onPlayTimer(event:TimerEvent):void
{
    var estimatedLength:int =
        Math.ceil(this.s.length / (this.s.bytesLoaded / this.s.bytesTotal));
    var progEvent:ProgressEvent =
        new ProgressEvent(PLAY_PROGRESS, false, false, this.sc.position, estimatedLength);
    this.dispatchEvent(progEvent);
}
```

Метод `onPlayTimer()` оценивает общий размер, как описано в разделе «Отслеживание воспроизведения» на странице 486. Затем он создает новый экземпляр ProgressEvent с типом события `SoundFacade.PLAY_PROGRESS`, в котором свойство `bytesLoaded` содержит текущую позицию объекта SoundChannel, а свойство `bytesTotal` указывает предполагаемую продолжительность аудиоданных.

## Приостановка и возобновление воспроизведения звука

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `SoundFacade.play()`, описанный выше, принимает параметр `pos`, который соответствует начальной позиции аудиоданных. Если `pos` имеет значение 0, воспроизведение звука запускается с самого начала.

Метод `SoundFacade.stop()` также принимает параметр `pos`, как показано ниже.

```
public function stop(pos:int = 0):void
{
    if (this.isPlaying)
    {
        this.pausePosition = pos;
        this.sc.stop();
        this.playTimer.stop();
        this.isPlaying = false;
    }
}
```

При каждом вызове метод `SoundFacade.stop()` задает свойство `pausePosition`, чтобы сообщить приложению позицию точки воспроизведения на тот случай, если пользователь захочет возобновить воспроизведение того же звука.

Методы `SoundFacade.pause()` и `SoundFacade.resume()`, показанные ниже, вызывают методы `SoundFacade.stop()` и `SoundFacade.play()` соответственно, каждый раз передавая значение для параметра `pos`.

```
public function pause():void
{
    stop(this.sc.position);
}

public function resume():void
{
    play(this.pausePosition);
}
```

Метод `pause()` передает текущее значение `SoundChannel.position` методу `play()`, который сохраняет это значение в свойстве `pausePosition`. Метод `resume()` начинает воспроизведение того же звука, используя в качестве начальной точки значение свойства `pausePosition`.

## Расширение примера Podcast Player

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Данный пример представляет собой каркас проигрывателя Podcast Player, демонстрирующий применение класса SoundFacade многократного пользования. Чтобы сделать приложение более полезным, в него можно добавить другие функции, включая следующие:

- Сохраняйте список каналов и информацию об использовании для каждого эпизода в экземпляре SharedObject, который может использоваться при следующем запуске приложения.
- Дайте пользователю возможность добавлять свои каналы RSS в список каналов подкастов.
- Запоминайте позицию точки воспроизведения, когда пользователь останавливает или закрывает эпизод, чтобы при следующем запуске приложения воспроизведение можно было начать с этого места.
- Загружайте mp3-файлы эпизодов для прослушивания в автономном режиме, когда пользователь не подключен к Интернету.
- Добавьте функции управления подписками, которые периодически проверяют наличие новых эпизодов в канале подкастов и автоматически обновляют их список.
- Добавьте функции поиска и просмотра подкастов с помощью API-интерфейса службы размещения подкастов, такой как Odeo.com.

# Глава 25. Работа с видео

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Flash-видео — это одна из выдающихся технологий в Интернете. Однако традиционное представление видео в прямоугольном экране с полосой прогресса и кнопками управления под ним является единственным возможным применением видеосодержимого. С помощью ActionScript можно осуществлять тонкую настройку загрузки, представления и воспроизведения видео, а также управлять этими процессами.

## Основы видео

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Одной из важных возможностей проигрывателя Adobe® Flash® Player и среды Adobe® AIR™ является воспроизведение видеоданных и манипулирование ими с помощью ActionScript так же, как другим визуальным содержимым, таким как изображения, анимация, текст и прочее. Создавая файл Flash Video (FLV) в Adobe Flash CS4 Professional, можно выбрать оболочку, которая включает общие элементы управления воспроизведением. Однако вы можете не ограничиваться доступными вариантами. ActionScript предоставляет возможность полностью контролировать процессы загрузки, отображения и воспроизведения видео, то есть можно создать собственную оболочку проигрывателя или использовать видеосодержимое нетрадиционным способом. Работа с видео в ActionScript подразумевает использование комбинации из нескольких классов.

- Класс Video : классическое окно видеосодержимого в рабочей области является экземпляром класса Video. Класс Video является экранным объектом, поэтому им можно манипулировать с помощью тех же приемов, что и другими экранными объектами, включая расположение, применение преобразований, фильтров, режимов наложения и прочее.
- Класс StageVideo: обычно класс Video использует программное раскодирование и визуализацию. Если устройство поддерживает аппаратное ускорение с использованием графического процессора, можно воспользоваться возможностями визуализации с аппаратным ускорением, переключившись на класс StageVideo. API-интерфейс StageVideo содержит набор событий, которые указывают коду, когда рекомендуется выполнять переключение между объектами StageVideo и Video. Класс StageVideo накладывает некоторые незначительные ограничения на воспроизведение видео. Если в приложении допускаются эти ограничения, реализуйте API-интерфейс StageVideo. См. раздел ««Рекомендации и ограничения» на странице 551».
- Класс NetStream: когда загружается видеофайл для управления кодом ActionScript, экземпляр NetStream представляет источник видеосодержимого, в данном случае это поток видеоданных. Вместе с экземпляром NetStream также используется объект NetConnection, который обеспечивает подключение к видеофайлу. Это своего рода туннель, по которому передаются видеоданные.
- Класс Camera: когда обрабатываются видеоданные с камеры, подключенной к компьютеру пользователя, экземпляр Camera представляет источник видеосодержимого, то есть камеру пользователя и передаваемые с нее видеоданные. Новшеством в средах Flash Player 11.4 и AIR 3.4 является то, что можно использовать камеру в качестве канала StageVideo.

Внешний видеофайл можно загружать методом последовательной загрузки со стандартного веб-сервера или методом потоковой передачи со специализированного сервера, такого как Adobe Flash® Media Server.

**Важные понятия и термины**

**Ключевая точка** Маркер, который можно поместить в определенный момент времени в видеофайле. Например, она может выполнять функцию закладки для быстрого поиска этого момента или предоставлять дополнительные данные, связанные с этим моментом.

**Кодировка** Процесс преобразования видеоданных из одного формата в другой, например, для преобразования исходного видео с высоким разрешением в формат, пригодный для распространения через Интернет.

**Кадр** Один сегмент видеоинформации. Каждый кадр подобен неподвижной картинке, которая представляет собой снимок одного момента времени. При воспроизведении последовательности кадров на большой скорости возникает иллюзия движения.

**Ключевой кадр** Видеокадр, содержащий полные сведения для кадра. Другие кадры, следующие за ключевым, содержат только информацию об изменениях по сравнению с ключевым кадром, а не полные сведения о кадре.

**Метаданные** Информация о видеофайле, которая встроена в видеофайл и извлекается при загрузке видео.

**Последовательная загрузка** Когда видеофайл передается со стандартного веб-сервера, видеоданные загружаются с помощью последовательной загрузки, то есть по очереди. Преимущество последовательной загрузки заключается в том, что воспроизведение видео можно начать до завершения загрузки всего файла. Однако в этом случае нельзя переходить вперед к еще незагруженным фрагментам видео.

**Потоковая передача** В качестве альтернативы последовательной загрузки можно использовать специальный видеосервер, который позволяет передавать видео через Интернет с помощью метода потоковой передачи (иногда называется «истинная потоковая передача»). При потоковой передаче компьютер пользователя никогда не загружает все видеоданные за один раз. Чтобы ускорить загрузку, в любой момент времени компьютеру нужна только часть всей видеоинформации. Так как передачей видеосодержимого управляет специальный сервер, любая часть видео может быть получена в любой момент времени, поэтому для ее просмотра не требуется ждать завершения загрузки всего файла.

# Сведения о форматах видео

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Помимо видеоформата Adobe FLV, Flash Player и Adobe AIR поддерживают видео и аудио, закодированное как H.264 и HE-AAC из стандартных форматов MPEG-4. Эти форматы позволяют осуществлять потоковую передачу видео высокого качества при низкой скорости потока. Разработчики могут использовать стандартные инструменты в данной отрасли, включая Adobe Premiere Pro и Adobe After Effects, для создания и предоставления привлекательного видеосодержимого.

| Тип | Формат | Контейнер |
|---|---|---|
| Видео | H.264 | MPEG-4: MP4, M4V, F4V, 3GPP |
| Видео | Sorenson Spark | FLV-файл |
| Видео | ON2 VP6 | FLV-файл |
| Аудио | AAC+ / HE-AAC / AAC v1 / AAC v2 | MPEG-4:MP4, M4V, F4V, 3GPP |

| Тип | Формат | Контейнер |
|---|---|---|
| Аудио | Mp3 | Mp3 |
| Аудио | Nellymoser | FLV-файл |
| Аудио | Speex | FLV-файл |

### Дополнительные разделы справки

Flash Media Server: поддерживаемые кодеки

Потоки динамического HTML от Adobe

## Кодирование видео для мобильных устройств

Среда AIR на Android может раскодировать видео H.264 в широком диапазоне. Однако только небольшое подмножество видео H.264 может стабильно воспроизводиться на мобильных телефонах. Это связано с тем, что на многих мобильных телефонах имеются ограничения по мощности обработки. Проигрыватель Adobe Flash Player может раскодировать видео H.264, используя встроенное аппаратное ускорение. Такое декодирование обеспечивает более высокое качество при более низком потреблении мощности.

Стандарт H.264 поддерживает несколько методов кодирования. Только профессиональные устройства могут стабильно воспроизводить видео со сложными профилями и уровнями. Однако на большинстве устройств можно проигрывать видео, закодированное с использованием базового профиля. На мобильных устройствах аппаратное ускорение доступно для нескольких из этих методов. Профиль и параметры уровня определяют подмножество методов кодирования и настройки, используемые кодировщиком. Для разработчиков это означает, что видео кодируется с выбранным разрешением, при котором оно будет качественно проигрываться на большинстве устройств.

Разрешение, при котором будет выгодно использовать аппаратное ускорение, зависит от устройства, однако следующие стандартные расширения поддерживаются большинством устройств.

| Соотношение сторон | Рекомендуемое разрешение | | |
|---|---|---|---|
| 4:3 | 640 × 480 | 512 × 384 | 480 × 360 |
| 16:9 | 640 × 360 | 512 x 288 | 480 × 272 |

***Примечание.*** *Flash Player поддерживает все уровни и профили стандарта H.264. Следуя этим рекомендациям, вы сможете использовать аппаратное ускорение и предоставить пользователю более высокое качество на большинстве устройств. Данные рекомендации не являются обязательными.*

Подробные сведения и описание параметров кодирования Adobe Media Encoder CS5 см. в статье Рекомендации по кодированию видео H.264 для Flash Player 10.1 на мобильных устройствах.

***Примечание.*** *В ОС iOS с помощью класса Video можно воспроизводить только видео, закодированное с использованием кодеков Sorenson Spark и On2 VP6. Для воспроизведения видео H.264 в видеопроигрывателе устройства запустите URL-адрес видео с помощью функции* `flash.net.navigateToURL()`*. Видео H.264 также можно воспроизвести, используя тег <video> на HTML-странице, отображаемой в объекте StageWebView.*

## Совместимость Flash Player и AIR с закодированными видеофайлами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Flash Player 7 поддерживает FLV-файлы, закодированные с помощью видеокодека Sorenson™ Spark™. Flash Player 8 поддерживает FLV-файлы, закодированные с помощью кодека Sorenson Spark или On2 VP6 в ПО Flash Professional 8. Видеокодек On2 VP6 поддерживает альфа-канал.

Flash Player 9.0.115.0 и более поздние версии поддерживают файлы, полученные из стандартного формата контейнера MPEG-4. К таким файлам относятся F4V, MP4, M4A, MOV, MP4V, 3GP и 3G2, если они содержат видео H.264 или аудио HE-AAC v2 или и то и другое. По сравнению со схожим профилем кодирования в Sorenson или On2, формат H.264 обеспечивает более высокое качество видеоизображения при невысоких скоростях потока. HE-AAC v2 — это расширение AAC, стандартного аудиоформата, определенного в видеостандарте MPEG-4. HE-AAC v2 использует приемы репликации спектральной полосы (SBR) и параметрического стерео (PS) с целью повышения эффективности кодирования при низкой скорости потока.

В следующей таблице перечислены поддерживаемые кодеки. Здесь также приводятся соответствующие форматы SWF-файлов и версии Flash Player или AIR, необходимые для их воспроизведения.

| **Кодек** | **Версия формата SWF-файла (самая ранняя поддерживаемая версия публикации)** | **Flash Player и AIR (самая ранняя версия, необходимая для воспроизведения)** |
|---|---|---|
| Sorenson Spark | 6 | Flash Player 6, Flash Lite 3 |
| On2 VP6 | 6 | Flash Player 8, Flash Lite 3.<br>Публикацию и воспроизведение видео On2 VP6 поддерживает только Flash Player 8 и более поздние версии. |
| H.264 (MPEG-4 Part 10) | 9 | Flash Player 9 обновление 3, AIR 1.0 |
| ADPCM | 6 | Flash Player 6, Flash Lite 3 |
| Mp3 | 6 | Flash Player 6, Flash Lite 3 |
| AAC (MPEG-4 Part 3) | 9 | Flash Player 9 обновление 3, AIR 1.0 |
| Speex (звук) | 10 | Flash Player 10, AIR 1.5 |
| Nellymoser | 6 | Flash Player 6 |

## Сведения о видеоформатах Adobe F4V и FLV

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Adobe предоставляет видеоформаты F4V и FLV для потоковой передачи содержимого в Flash Player и AIR. Полное описание этих видеоформатов см. на странице www.adobe.com/go/video_file_format_ru.

## Видеоформат F4V

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Начиная с Flash Player Update 3 (9.0.115.0) и AIR 1.0 поддерживается видеоформат Adobe F4V, который создан на базе формата ISO MP4. Подгруппы формата поддерживают разные функции. Проигрыватель Flash Player ожидает действующий F4V-файл, чтобы начать работу с одним из следующих полей верхнего уровня.

- ftyp

  Поле «ftyp» определяет функции, которые программа должна поддерживать для воспроизведения файла в определенном формате.

- moov

  Поле «moov» на самом деле является заголовком F4V-файла. Оно содержит одно или несколько других полей, которые также содержат другие поля, определяющие структуру F4V-данных. F4V-файл должен содержать только одно поле «moov».

- mdat

  Поле «mdat» содержит основные данные для F4V-файла. FV-файл содержит только одно поле «mdat». В файле обязательно должно быть поле «moov», так как без него невозможно обработать поле «mdat».

F4V-файлы поддерживают многобайтовые целые числа с порядком байтов от большего к меньшему, при котором самый значимый байт становится первым и получает наименьший индекс.

## Видеоформат FLV

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Видеоформат Adobe FLV содержи закодированные аудио- и видеоданные для передачи проигрывателем Flash Player. Чтобы преобразовать видеофайл QuickTime или Windows Media в FLV-файл можно использовать такой кодировщик, как Adobe Media Encoder или Sorenson™ Squeeze.

***Примечание.*** *FLV-файлы можно создавать путем импорта видео в инструмент разработки Flash и последующего экспорта в виде FLV-файла. Подключаемый модуль «Экспорт FLV-файла» позволяет экспортировать FLV-файлы из поддерживаемых приложений для редактирования видеосодержимого. Для загрузки FLV-файлов с веб-сервера необходимо зарегистрировать на сервере расширение файла и MIME-тип. Проверьте документацию сервера. Для FLV-файлов используется MIME-тип* `video/x-flv`*. Дополнительные сведения см. в разделе ««О настройке FLV-файлов с целью размещения на сервере» на странице 542».*

Дополнительные сведения о FLV-файлах см. в разделе ««Дополнительные темы о видеофайлах» на странице 541».

## Внешнее и встроенное видео

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Применение внешних видеофайлов открывает определенные возможности, недоступные при использовании импортированного видео.

- В приложение можно использовать более длинные видеоклипы, не опасаясь замедления воспроизведения. Внешние видеофайлы используют кэш, то есть большие файлы для сохранения разбиваются на небольшие фрагменты, а затем вызываются в динамическом режиме По этой причине внешние F4V- и FLV-файлы требуют меньше памяти, чем встроенные.

- Внешний видеофайл может иметь частоту кадров, отличную от частоты кадров SWF-файла, в котором он воспроизводится. Например, можно задать для SWF-файла частоту 30 кадров в секунду (кадр/сек), а видеофайл может иметь частоту 21 кадр/сек. Такая настройка позволяет лучше управлять внешним видеофайлом и обеспечивать более стабильное воспроизведение по сравнению с встроенным файлом. Кроме того, это позволяет воспроизводить видеофайлы с разной частотой кадров, не изменяя существующее содержимое SWF-файла.
- При использовании внешних видеофайлов воспроизведение SWF-содержимого не прерывается в процессе загрузки видео. Хотя иногда это все же происходит (например, в момент доступа к приводу компакт-дисков). Видеофайлы могут выполнять функции независимо от содержимого SWF-файла, не прерывая воспроизведение.
- Захватывать видеосодержимое легче при использовании внешних FLV-файлов, так как можно получать метаданные видео с помощью обработчиков событий.

# Понимание класса Video

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс Video позволяет отображать в приложении потоковое видео в реальном времени без внедрения в SWF-файл. Захватывать и воспроизводить видео в реальном времени можно с помощью метода `Camera.getCamera()`. Кроме того, класс Video можно использовать для воспроизведения видеофайлов через HTTP или из локальной файловой системы. Существует несколько разных способов использования класса Video в проектах.

- Можно загрузить видеофайл динамически с помощью классов NetConnection и NetStream и показать его в объекте Video.
- Можно захватывать ввод с камеры пользователя. Дополнительные сведения см. в разделе ««Работа с камерами» на странице 558».
- Можно использовать компонент FLVPlayback.
- Можно использовать элемент управления VideoDisplay.

***Примечание.*** *Экземпляры объекта Video в рабочей области являются экземплярами класса Video.*

Несмотря на то, что класс Video находится в пакете flash.media, он наследует методы и свойства класса flash.display.DisplayObject. Поэтому все функции экранного объекта, такие как матрицы преобразования и фильтры, также применяются к экземплярам Video.

Дополнительные сведения см. в разделах ««Операции с экранными объектами» на странице 186», ««Работа с геометрией» на странице 227» и ««Фильтрация экранных объектов» на странице 289».

# Загрузка видеофайлов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Загрузка видеофайлов с помощью классов NetStream и NetConnection выполняется в несколько этапов. Для достижения наилучших результатах добавление объекта Video в список отображения, присоединение объекта NetStream к экземпляру Video и вызов метода `play()` объекта NetStream рекомендуется выполнять в указанном порядке.

1 Создайте новый объект NetConnection. Если необходимо подключиться к локальному видеофайлу или файлу, не использующему сервер, такой как Adobe Flash Media Server 2, передайте значение `null` методу `connect()` для воспроизведения видеофайлов в HTTP-адреса или локального диска. При подключении к серверу установите для этого параметра URI-адрес приложения, в котором содержится видеофайл на сервере.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
```

2 Создайте новый объект Video, который показывает видео, и добавьте его в список отображения рабочей области, как в следующем фрагменте кода.

```
var vid:Video = new Video();
addChild(vid);
```

3 Создайте объект NetStream, передавая конструктору в качестве аргумента объект NetConnection. Следующий фрагмент кода соединяет объект NetStream с экземпляром NetConnection и настраивает обработчики события для потока.

```
var ns:NetStream = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS,netStatusHandler);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);

function netStatusHandler(event:NetStatusEvent):void
{
    // handle netStatus events, described later
}

function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}
```

4 Присоедините объект NetStream к объекту Video с помощью метода `attachNetStream()` объекта Video, как в следующем фрагменте кода.

```
vid.attachNetStream(ns);
```

5 Вызовите метод `play()` объекта NetStream, передавая в качестве аргумента URL видеофайла, чтобы начать воспроизведение видео. Следующий фрагмент кода воспроизводит видеофайл с именем video.mp4, который находится в том же каталоге, что SWF-файл.

```
ns.play("video.mp4");
```

### Дополнительные разделы справки

Flex: управление Spark VideoPlayer

spark.components.VideoDisplay

# Управление воспроизведением видео

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс NetStream имеет четыре основных метода для управления воспроизведением видео.

`pause()`: приостанавливает воспроизведение видеопотока. Если видео уже приостановлено, вызов данного метода ни к чему не приведет.

`resume()`: возобновляет воспроизведение видеопотока, который был приостановлен. Если видео уже воспроизводится, вызов данного метода ни к чему не приведет.

`seek()`: ищет ключевой кадр, расположенный ближе всех к указанному местоположению (смещение от начала потока, в секундах).

`togglePause()`: приостанавливает и возобновляет воспроизведение видеопотока.

***Примечание.*** *Метода `stop()` не существует. Чтобы остановить поток, необходимо приостановить воспроизведение и перейти к началу видеопотока.*

***Примечание.*** *Метод `play()` не возобновляет воспроизведение, он служит для загрузки видеофайлов.*

Следующий пример демонстрирует управление видео с помощью нескольких кнопок. Чтобы выполнить этот пример, создайте новый документ и добавьте четыре экземпляра Button в рабочее пространство (`pauseBtn`, `playBtn`, `stopBtn` и `togglePauseBtn`).

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

pauseBtn.addEventListener(MouseEvent.CLICK, pauseHandler);
playBtn.addEventListener(MouseEvent.CLICK, playHandler);
stopBtn.addEventListener(MouseEvent.CLICK, stopHandler);
togglePauseBtn.addEventListener(MouseEvent.CLICK, togglePauseHandler);

function pauseHandler(event:MouseEvent):void
{
    ns.pause();
}
function playHandler(event:MouseEvent):void
{
    ns.resume();
}
function stopHandler(event:MouseEvent):void
{
    // Pause the stream and move the playhead back to
    // the beginning of the stream.
    ns.pause();
    ns.seek(0);
}
function togglePauseHandler(event:MouseEvent):void
{
    ns.togglePause();
}
```

Нажатие экземпляра кнопки `pauseBtn` в процессе воспроизведения приостанавливает видеопоток. Если видео уже приостановлено, кнопка не действует. Нажатие кнопки `playBtn` возобновляет воспроизведение, если видеопоток был приостановлен. Если видео уже воспроизводится, кнопка не действует.

## Определение конца видеопотока

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы прослушивать начало и конец видеопотока, необходимо добавить прослушиватель события `netStatus` для экземпляра NetStream. Следующий код демонстрирует прослушивание различных кодов в процессе воспроизведения видеофайла.

```
ns.addEventListener(NetStatusEvent.NET_STATUS, statusHandler);
function statusHandler(event:NetStatusEvent):void
{
    trace(event.info.code)
}
```

Выше приведенный код генерирует следующий вывод:

```
NetStream.Play.Start
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Flush
NetStream.Play.Stop
NetStream.Buffer.Empty
NetStream.Buffer.Flush
```

В частности, необходимо прослушивать два кода: «NetStream.Play.Start» и «NetStream.Play.Stop», которые обозначают начало и конец воспроизведения видео. Следующий код использует инструкцию switch, чтобы фильтровать эти два кода и отслеживать сообщения.

```
function statusHandler(event:NetStatusEvent):void
{
    switch (event.info.code)
    {
        case "NetStream.Play.Start":
            trace("Start [" + ns.time.toFixed(3) + " seconds]");
            break;
        case "NetStream.Play.Stop":
            trace("Stop [" + ns.time.toFixed(3) + " seconds]");
            break;
    }
}
```

Прослушивая событие `netStatus` (`NetStatusEvent.NET_STATUS`), можно создать видеопроигрыватель, который после завершения воспроизведения текущего файла загружает следующий видеофайл из списка воспроизведения.

# Воспроизведение видео в полноэкранном режиме

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Flash Player и AIR позволяют создавать полноэкранные приложения для воспроизведения видео с поддержкой масштабирования видео во весь экран.

Если содержимое AIR выполняется в полноэкранном режиме на персональном компьютере, системный хранитель экрана и параметры энергосбережения отключаются до тех пор, пока не прекратится ввод видео или пока пользователь не выйдет из полноэкранного режима.

Дополнительные сведения см. в разделе ««Работа в полноэкранном режиме» на странице 179».

### Включение полноэкранного режима для проигрывателя Flash Player в обозревателе

Прежде чем добавить возможность перехода в полноэкранный режим для проигрывателя Flash Player в обозревателе, ее необходимо включить через шаблон «Опубликовать» для приложения. Шаблоны, допускающие переход в полноэкранный режим, включают теги `<object>` и `<embed>`, содержащие параметр `allowFullScreen`. Следующий пример демонстрирует параметр `allowFullScreen` в теге `<embed>`.

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    id="fullScreen" width="100%" height="100%"
    codebase="http://fpdownload.macromedia.com/get/flashplayer/current/swflash.cab">
    ...
    <param name="allowFullScreen" value="true" />
    <embed src="fullScreen.swf" allowFullScreen="true" quality="high" bgcolor="#869ca7"
        width="100%" height="100%" name="fullScreen" align="middle"
        play="true"
        loop="false"
        quality="high"
        allowScriptAccess="sameDomain"
        type="application/x-shockwave-flash"
        pluginspage="http://www.adobe.com/go/getflashplayer">
    </embed>
    ...
</object>
```

Во Flash выберите «Файл» > «Параметры публикации» и в открывшемся окне на вкладке «HTML» укажите шаблон «Только Flash — разрешить полноэкранный режим».

В Flex проверьте, что шаблон HTML включает теги `<object>` и `<embed>`, поддерживающие полноэкранный режим.

### Переход в полноэкранный режим

При воспроизведении содержимого Flash Player в обозревателе переход видео в полноэкранный режим выполняется в ответ на щелчок мыши или нажатие клавиши. Например, в полноэкранный режим можно перейти, когда пользователь щелкнет кнопку с надписью «Полноэкранный режим» или выберет команду «Полноэкранный режим» из контекстного меню. Чтобы отвечать на действия пользователя, добавьте прослушиватель событий для объекта, на который направлено действие. Следующий код добавляет прослушиватель событий для кнопки, которую пользователь нажимает для перехода в полноэкранный режим.

```
var fullScreenButton:Button = new Button();
fullScreenButton.label = "Full Screen";
addChild(fullScreenButton);
fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);

function fullScreenButtonHandler(event:MouseEvent)
{
    stage.displayState = StageDisplayState.FULL_SCREEN;

}
```

Код инициирует переход в полноэкранный режим, задавая свойству `Stage.displayState` значение `StageDisplayState.FULL_SCREEN`. Этот код масштабирует всю рабочую область до размеров полного экрана, изменяя размер видео пропорционально занимаемой им части рабочей области.

Свойство `fullScreenSourceRect` позволяет указать определенную часть рабочей области для масштабирования до полного экрана. Сначала определите прямоугольник, который нужно масштабировать до размера полного экрана. Затем назначьте его свойству `Stage.fullScreenSourceRect`. Эта версия функции `fullScreenButtonHandler()` добавляет две дополнительные строки кода, которые масштабируют на весь экран только видео.

```
private function fullScreenButtonHandler(event:MouseEvent)
{
    var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width, video.height);
    stage.fullScreenSourceRect = screenRectangle;
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

Несмотря на то, что в этом примере обработчик события вызывается в ответ на щелчок мыши, данный способ перехода в полноэкранный режим одинаково подходит для Flash Player и AIR. Определите прямоугольник, который требуется масштабировать, а затем задайте свойство `Stage.displayState`. Дополнительные сведения см. в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

В полном примере, который приведен ниже, добавляется код, который создает подключение и объект NetStream для видео и начинает его воспроизведение.

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import fl.controls.Button;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.FullScreenEvent;
    import flash.geom.Rectangle;

    public class FullScreenVideoExample extends Sprite
    {
        var fullScreenButton:Button = new Button();
        var video:Video = new Video();

        public function FullScreenVideoExample()
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("http://www.helpexamples.com/flash/video/water.flv");

            fullScreenButton.x = 100;
```

```
            fullScreenButton.y = 270;
            fullScreenButton.label = "Full Screen";
            addChild(fullScreenButton);
            fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);
        }

        private function fullScreenButtonHandler(event:MouseEvent)
        {
            var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width,
video.height);
            stage.fullScreenSourceRect = screenRectangle;
            stage.displayState = StageDisplayState.FULL_SCREEN;
        }

        public function onMetaData(infoObject:Object):void
        {
            // stub for callback function
        }
    }
}
```

Функция `onMetaData()` является функцией обратного вызова для обработки метаданных видео, если они имеются. Функцией обратного вызова называется функция, которую среда выполнения вызывает в ответ на некоторые типы событий. В этом примере функция `onMetaData()` является условием, необходимым для выполнения метода. Дополнительные сведения см. в разделе ««Написание методов обратного вызова для метаданных и ключевых точек» на странице 522»

### Выход из полноэкранного режима

Пользователь может выйти из полноэкранного режима с помощью одного из сочетаний клавиш, например, нажав клавишу Esc. Выйти из полноэкранного режима можно и с помощью кода ActionScript, задав свойству `displayState` значение `StageDisplayState.NORMAL`. Следующий код выключает полноэкранный режим, когда происходит событие `netStatus` сетевого статуса NetStream.Play.Stop.

```
videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);

private function netStatusHandler(event:NetStatusEvent)
{
    if(event.info.code == "NetStream.Play.Stop")
        stage.displayState = StageDisplayState.NORMAL;
}
```

### Аппаратное ускорение полноэкранного режима

При изменении масштаба прямоугольного сегмента рабочего стола до размера всего экрана Flash Player или AIR использует аппаратное ускорение, если оно доступно и включено. Среда выполнения использует видеоадаптер компьютера для ускорения увеличения масштаба видео или части рабочей области до размера полного экрана. В этих случаях приложения Flash Player часто могут работать более эффективно путем включения класса StageVideo вместо класса Video (или класса Camera; Flash Player 11.4/AIR 3.4 и более поздних версий).

Дополнительные сведения об аппаратном ускорении в полноэкранном режиме см. в разделе ««Работа в полноэкранном режиме» на странице 179». Дополнительные сведения о классе StageVideo см. в разделе ««Использование класса StageVideo для визуализации с аппаратным ускорением» на странице 549».

# Потоковые видеофайлы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для потоковой передачи файлов с сервера Flash Media Server, можно использовать классы NetConnection и NetStream, чтобы подключиться к экземпляру удаленного сервера и воспроизвести заданный поток. Чтобы указать сервер RTMP, методу `NetConnection.connect()` вместо значения `null` необходимо передать требуемый RTMP URL, например «rtmp://localhost/appName/appInstance». Для воспроизведения эфирного или записанного потока с указанного сервера Flash Media Server необходимо передать идентификатор (имя) эфирных данных, опубликованных методом `NetStream.publish()`, или имя файла записанного потока для воспроизведения методу `NetStream.play()`.

## Отправление видеоданных на сервер

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если требуется создавать более сложные приложения с объектами Video или Camera, сервер Flash Media Server предоставляет возможности потоковой передачи медиасодержимого и среду разработки для создания и распространения медиаприложений для широкой аудитории. Такое сочетание возможностей позволяет разработчикам создавать такие приложения, как видео по требованию, эфирные трансляции веб-мероприятий, потоковая передача Mp3-файлов, а также видеоблоги, видеосообщения и мультимедийный чат. Дополнительные сведения см. в интерактивной документации по Flash Media Server на странице www.adobe.com/go/learn_fms_docs_ru.

# Понимание ключевых точек

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Во время кодировки в видеофайл Adobe F4V или FLV можно встраивать ключевые точки. Раньше ключевые точки встраивались в фильмы, чтобы дать киномеханику зрительный сигнал о том, что заканчивается рулон кинопленки. В видеоформатах Adobe F4V и FLV ключевая точка позволяет запустить одно или несколько других действий в приложении в момент ее появления в видеопотоке.

В Flash-видео можно использовать несколько типов ключевых точек. С помощью кода ActionScript можно взаимодействовать с ключевыми точками, встроенными в видеофайл в процессе создания.

- Ключевые точки навигации: встраиваются в видеопоток и пакет метаданных в процессе кодирования видеофайла. Ключевые точки навигации дают пользователям возможность перейти к нужной части файла.
- Ключевые точки событий: встраиваются в видеопоток и пакет метаданных в процессе кодирования видеофайла. Можно написать код для обработки событий, которые запускаются в указанных точках в процессе воспроизведения видео.

- Ключевые точки ActionScript: доступны только для компонента Flash FLVPlayback. Ключевые точки ActionScript являются внешними ключевыми точками, которые создаются и используются с помощью кода ActionScript. Можно создать код для запуска этих ключевых точек в процессе воспроизведения видео. Эти ключевые точки не такие точные, как встроенные (до десятой доли секунды), так как видеопроигрыватель отслеживает их отдельно. Если вы планируете создать приложение, которое дает пользователям возможность переходить к ключевым точкам, эти точки необходимо создать и встроить при кодировании файла, а не с помощью ActionScript. Ключевые точки необходимо встроить в FLV-файл, чтобы они были более точными.

Ключевые точки навигации создают ключевой кадр в указанном местоположении, поэтому переместить точку воспроизведения в это место можно с помощью кода. В видеофайле можно задать определенные точки, к которым сможет переходить пользователь. Например, видео может иметь несколько разделов или сегментов, и для управления воспроизведением в видеофайл можно встроить ключевые точки навигации.

Дополнительные сведения о кодировании видеофайлов Adobe с ключевыми точками см. в разделе «Встроенные ключевые точки» руководства *Использование Flash*.

Манипулировать параметрами ключевых точек можно с помощью кода ActionScript. Параметры сигнальных точек входят в состав объекта события, поучаемого обработчиком обратного вызова.

Чтобы запускать определенные действия в коде, когда FLV-файл достигает конкретной ключевой точки, используйте обработчик события `NetStream.onCuePoint`.

Чтобы синхронизировать действие с ключевой точкой в видеофайле F4V, необходимо получить данные ключевой точки из функции обратного вызова `onMetaData()` или `onXMPData()` и запустить ключевую точку с помощью класса Timer в ActionScript 3.0. Дополнительные сведения о ключевых точках F4V см. в разделе ««Использование метода onXMPData()» на странице 534».

Дополнительные сведения об обработке ключевых точек и метаданных см. в разделе ««Написание методов обратного вызова для метаданных и ключевых точек» на странице 522».

# Написание методов обратного вызова для метаданных и ключевых точек

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В приложении можно запускать различные действия, когда проигрыватель получает определенные метаданные или достигает определенной ключевой точки. Когда происходят эти события, для их обработки необходимо использовать специальные методы обратного вызова. Класс NetStream определяет следующие события метаданных, которые могут происходить в процессе воспроизведения: `onCuePoint` (только для FLV-файлов), `onImageData`, `onMetaData`, `onPlayStatus`, `onTextData` и `onXMPData`.

Для этих обработчиков необходимо написать методы обратного вызова, иначе в среде выполнения Flash могут возникать ошибки. Например, следующий код воспроизводит FLV-файл с именем video.flv, который находится в той же папке, что и SWF-файл.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    trace(event.text);
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Предыдущий код загружает локальный видеофайл с именем video.flv и прослушивает событие `asyncError` (`AsyncErrorEvent.ASYNC_ERROR`). Это событие отправляется, когда собственный асинхронный код выдает исключение. В данном случае оно отправляется, когда видеофайл содержит метаданные или данные ключевой точки, но соответствующие прослушиватели не определены. Предыдущий код обрабатывает событие `asyncError` и игнорирует ошибку, если вы не заинтересованы в метаданных или ключевых точках, имеющихся в видеофайле. Если FLV-файл имеет метаданные и несколько ключевых точек, то метод `trace()` отобразит следующие сообщения об ошибках:

```
Error #2095: flash.net.NetStream was unable to invoke callback onMetaData.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
```

Ошибки происходят, так как объект NetStream не может найти метод обратного вызова `onMetaData` или `onCuePoint`. Определить методы обратного вызова в приложении можно несколькими способами.

### Дополнительные разделы справки

Flash Media Server: обработка метаданных в потоках

## Назначение экземпляра Object свойству client объекта NetStream

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если задать свойству `client` экземпляр класса Object или подкласса NetStream, можно изменить маршрут методов обратного вызова `onMetaData` и `onCuePoint` или полностью их игнорировать. Следующий пример демонстрирует, как с помощью пустого экземпляра Object можно игнорировать методы обратного вызова, не прослушивая событие `asyncError`.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var customClient:Object = new Object();

var ns:NetStream = new NetStream(nc);
ns.client = customClient;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Если требуется прослушивать метод обратного вызова `onMetaData` или `onCuePoint`, необходимо определить методы для его обработки, как показано в следующем примере.

```
var customClient:Object = new Object();
customClient.onMetaData = metaDataHandler;
function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
```

Предыдущий код прослушивает метод обратного вызова `onMetaData` и вызывает метод `metaDataHandler()`, который отслеживает строку. Когда среда выполнения Flash находит ключевые точки, то ошибки не создаются, даже если не определен метод обратного вызова `onCuePoint`.

## Создание пользовательского класса и определение методов для обработки методов обратного вызова

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Следующий код задает свойству `client` объекта NetStream пользовательский класс, CustomClient, который определяет обработчики для методов обратного вызова.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = new CustomClient();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Определение класса CustomClient выглядит следующим образом:

```
package
{
    public class CustomClient
    {
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
    }
}
```

Класс CustomClient определяет обработчика для метода обратного вызова `onMetaData`. Когда встречается ключевая точка и вызывается метод обратного вызова `onCuePoint`, отправляется событие `asyncError` (`AsyncErrorEvent.ASYNC_ERROR`) со строкой «flash.net.NetStream was unable to invoke callback onCuePoint» (Метод flash.net.NetStream не смог вызвать метод обратного вызова onCuePoint). Чтобы предотвратить эту ошибку, нужно определить метод обратного вызова `onCuePoint` в классе CustomClient или определить обработчик для события `asyncError`.

## Расширение класса NetStream и добавление методов для обработки функций обратного вызова

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Следующий код создает экземпляр класса CustomNetStream, который определяется в другом коде.

```
var ns:CustomNetStream = new CustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Следующий код определяет класс CustomNetStream, расширяющий класс NetStream, создает необходимый объект NetConnection и обрабатывает методы обратного вызова `onMetaData` и `onCuePoint`.

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public class CustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public function CustomNetStream()
        {
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
        public function onCuePoint(infoObject:Object):void
        {
            trace("cue point");
        }
    }
}
```

Если требуется переименовать методы `onMetaData()` и `onCuePoint()` в классе CustomNetStream, нужно использовать следующий код:

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public class CustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public var onMetaData:Function;
        public var onCuePoint:Function;
        public function CustomNetStream()
        {
            onMetaData = metaDataHandler;
            onCuePoint = cuePointHandler;
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
        private function metaDataHandler(infoObject:Object):void
        {
            trace("metadata");
        }
        private function cuePointHandler(infoObject:Object):void
        {
            trace("cue point");
        }
    }
}
```

## Расширение класса NetStream и превращение его в динамический класс

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Можно расширить класс NetStream и сделать его подкласс динамическим, чтобы обработчики обратного вызова `onCuePoint` и `onMetaData` добавлялись динамически. Это демонстрируется в следующем примере.

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Определение класса DynamicCustomNetStream выглядит следующим образом:

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public dynamic class DynamicCustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public function DynamicCustomNetStream()
        {
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
    }
}
```

Даже если нет обработчиков для методов обратного вызова `onMetaData` и `onCuePoint`, ошибки не генерируются, так как класс DynamicCustomNetStream является динамическим. Если требуется определить методы для обработчиков обратного вызова `onMetaData` и `onCuePoint`, можно использовать следующий код:

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.onMetaData = metaDataHandler;
ns.onCuePoint = cuePointHandler;
ns.play("http://www.helpexamples.com/flash/video/cuepoints.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
function cuePointHandler(infoObject:Object):void
{
    trace("cue point");
}
```

## Назначение this свойству client объекта NetStream

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если задать свойству `client` значение `this`, приложение ищет методы `onMetaData()` и `onCuePoint()` в текущей области действия. Это демонстрируется в следующем примере.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Если вызываются обработчики обратного вызова `onMetaData` и `onCuePoint` и нет методов для их обработки, ошибки не генерируются. Чтобы обработать эти методы обратного вызова, создайте в коде методы `onMetaData()` и `onCuePoint()`, как показано в следующем примере:

```
function onMetaData(infoObject:Object):void
{
    trace("metadata");
}
function onCuePoint(infoObject:Object):void
{
    trace("cue point");
}
```

# Использование ключевых точек и метаданных

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Используйте методы обратного вызова NetStream для захвата и обработки событий ключевых точек и метаданных в процессе воспроизведения видео.

## Использование ключевых точек

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В следующей таблице описаны методы обратного вызова, которые можно использовать для захвата ключевых точек F4V- и FLV-файлов в Flash Player и AIR.

| Среда выполнения | F4V | FLV |
|---|---|---|
| Flash Player 9/ AIR1.0 | | OnCuePoint |
| | | OnMetaData |
| Flash Player 10 | | OnCuePoint |
| | OnMetaData | OnMetaData |
| | OnXMPData | OnXMPData |

В следующем примере используется простой цикл `for..in`, повторяемый для каждого свойства в параметре `infoObject`, который получает функция `onCuePoint()`. При получении данных ключевой точки вызывается функция trace() для отображения сообщения.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onCuePoint(infoObject:Object):void
{
    var key:String;
    for (key in infoObject)
    {
        trace(key + ": " + infoObject[key]);
    }
}
```

На экран выводится следующее:

```
parameters:
name: point1
time: 0.418
type: navigation
```

Этот код использует один из нескольких приемов для назначения объекта, для которого выполняется метод обратного вызова. Можно использовать и другие приемы. Дополнительные сведения см. в разделе ««Написание методов обратного вызова для метаданных и ключевых точек» на странице 522».

## Использование метаданных видео

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Функции `OnMetaData()` и `OnXMPData()` можно использовать для доступа к метаданным в видеофайле, включая ключевые точки.

### Использование метода OnMetaData()

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метаданные содержат информацию о видеофайле, такую как продолжительность, ширину, высоту и частоту кадров. Метаданные, добавляемые в видеофайл, зависят то программного обеспечения, используемого для его кодирования.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onMetaData(infoObject:Object):void
{
    var key:String;
    for (key in infoObject)
    {
        trace(key + ": " + infoObject[key]);
    }
}
```

Предыдущий код генерирует следующий вывод:

```
width: 320
audiodelay: 0.038
canSeekToEnd: true
height: 213
cuePoints: ,,
audiodatarate: 96
duration: 16.334
videodatarate: 400
framerate: 15
videocodecid: 4
audiocodecid: 2
```

*Если видео не содержит аудиоданных, то метаданные, связанные со звуком (например, `audiodatarate`) возвращают значение `undefined`, так как во время кодировки в метаданные не были добавлены сведения о звуке.*

В предыдущем коде не отображается информация о ключевых точках. Чтобы показать метаданные о ключевых точках, можно использовать следующую функцию, которая рекурсивно отображает элементы в экземпляре Object.

```
function traceObject(obj:Object, indent:uint = 0):void
{
    var indentString:String = "";
    var i:uint;
    var prop:String;
    var val:*;
    for (i = 0; i < indent; i++)
    {
        indentString += "\t";
    }
    for (prop in obj)
    {
        val = obj[prop];
        if (typeof(val) == "object")
        {
            trace(indentString + " " + prop + ": [Object]");
            traceObject(val, indent + 1);
        }
        else
        {
            trace(indentString + " " + prop + ": " + val);
        }
    }
}
```

Если применить предыдущий код для отслеживания параметра `infoObject` в методе `onMetaData()`, мы получим следующий вывод:

```
width: 320
audiodatarate: 96
audiocodecid: 2
videocodecid: 4
videodatarate: 400
canSeekToEnd: true
duration: 16.334
audiodelay: 0.038
height: 213
framerate: 15
cuePoints: [Object]
    0: [Object]
        parameters: [Object]
            lights: beginning
        name: point1
        time: 0.418
        type: navigation
    1: [Object]
        parameters: [Object]
            lights: middle
        name: point2
        time: 7.748
        type: navigation
    2: [Object]
        parameters: [Object]
            lights: end
        name: point3
        time: 16.02
        type: navigation
```

Следующий код отображает метаданные для видео в формате MP4. Для выполнения этого примера требуется объект TextArea с именем `metaDataOut`, в который будут записываться метаданные

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.events.NetStatusEvent;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.MouseEvent;

    public class onMetaDataExample extends Sprite
    {
        var video:Video = new Video();

        public function onMetaDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);
            video.x = 185;
            video.y = 5;

            video.attachNetStream(videoStream);

            videoStream.play("video.mp4");

            videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
        }

        public function onMetaData(infoObject:Object):void
        {
            for(var propName:String in infoObject)
            {
                metaDataOut.appendText(propName + "=" + infoObject[propName] + "\n");
            }
        }

        private function netStatusHandler(event:NetStatusEvent):void
        {
            if(event.info.code == "NetStream.Play.Stop")
                stage.displayState = StageDisplayState.NORMAL;
        }
    }
}
```

Функция `onMetaData()` возвращает следующий вывод для данного аудиофайла:

```
moovposition=731965
height=352
avclevel=21
videocodecid=avc1
duration=2.36
width=704
videoframerate=25
avcprofile=88
trackinfo=[object Object]
```

### Использование информационного объекта

В следующей таблице приводятся возможные значения для метаданных видеофайла, которые передаются функции обратного вызова `onMetaData()` в экземпляре Object, который они получают.

| Параметр | Описание |
|---|---|
| aacaot | Тип аудиообъекта AAC; поддерживаются значения 0, 1 или 2. |
| avclevel | Номер уровня AVC IDC, например 10, 11, 20, 21 и т. д. |
| avcprofile | Номер профиля AVC, например 55, 77, 100 и т. д. |
| audiocodecid | Строка, обозначающая использованный аудиокодек (прием кодирования/декодирования), например «Mp3» или «mp4a». |
| audiodatarate | Число, обозначающие частоту, с которой был закодирован звук (килобайты в секунду). |
| audiodelay | Число, обозначающее время, которое соответствует времени «time 0» исходного FLV-файла. Для правильной синхронизации со звуком воспроизведение видео должно начинаться с задержкой. |
| canSeekToEnd | Логический параметр принимает значение `true`, если FLV-файл закодирован с ключевым кадром в последнем кадре, позволяющий перемещаться в конец последовательно загружаемого видеофайла. Значение параметра `false`, если FLV-файл не имеет ключевого кадра в последнем кадре. |
| cuePoints | Массив объектов, один для каждой ключевой точки, встроенный в FLV-файл. Значение не определено, если FLV-файл не имеет ключевых точек. Каждый объект имеет следующие свойства:<br>• `type`: строка, определяющая тип ключевой точки: «navigation» (ключевая точка навигации) или «event» (ключевая точка события).<br>• `name`: строка, являющаяся именем ключевой точки.<br>• `time`: число, являющееся временем ключевой точки в секундах с точностью до третьего десятичного знака (миллисекунды).<br>• `parameters`: необязательный параметр, имеющий пары «имя-значение», назначаемые пользователем при создании ключевых точек. |
| duration | Число, определяющее продолжительность видеофайла (в секундах). |
| framerate | Частота кадров FLV-файла. |
| height | Число, указывающее высоту видео в FLV-файле, в пикселах. |
| seekpoints | Массив, в котором перечислены доступные ключевые кадры в виде меток времени, в миллисекундах. Необязательно. |
| tags | Массив пар «ключ-значение», которые представляют сведения в атоме «ilst», эквивалентные тегам ID3 для MP4-файлов. Эти теги использует iTunes. Если они имеются, их можно использовать для отображения графических объектов. |
| trackinfo | Объект, предоставляющий информацию обо всех треках в MP4-файле, включая образец идентификатора описания. |

| Параметр | Описание |
|---|---|
| videocodecid | Строка, указывающая версию кодека, который использовался для кодирования видео, например «avc1» или «VP6F». |
| videodatarate | Скорость передачи видеоданных FLV-файла. |
| videoframerate | Частота кадров видео в формате MP4. |
| width | Число, указывающее ширину видео в FLV-файле, в пикселах. |

В следующей таблице приводятся возможные значения для параметра `videocodecid`.

| videocodecid | Имя кодека |
|---|---|
| 2 | Sorenson H.263 |
| 3 | Screen video (только для SWF версии 7 и более поздних) |
| 4 | VP6 (только для SWF версии 8 и более поздних) |
| 5 | VP6-видео с альфа-каналом (только для SWF версии 8 и более поздних) |

В следующей таблице приводятся возможные значения для параметра `audiocodecid`.

| audiocodecid | Имя кодека |
|---|---|
| 0 | несжатый |
| 1 | ADPCM |
| 2 | Mp3 |
| 4 | Nellymoser, 16 кГц, моно |
| 5 | Nellymoser, 8 кГц, моно |
| 6 | Nellymoser |
| 10 | AAC |
| 11 | Speex |

## Использование метода onXMPData()

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Функция обратного вызова `onXMPData()` получает информацию о расширяемой платформе метаданных Adobe Extensible Metadata Platform (XMP), встроенной в видеофайл Adobe F4V или FLV. Метаданные XMP включают ключевые точки и другие метаданные о видеофайле. Поддержка метаданных XMP впервые реализована в приложениях Flash Player 10 и Adobe AIR 1.5 и закреплена в последующих версиях.

Следующий пример обрабатывает сведения о ключевых точках в метаданных XMP.

```
package
{
    import flash.display.*;
    import flash.net.*;
    import flash.events.NetStatusEvent;
    import flash.media.Video;

    public class onXMPDataExample extends Sprite
    {
        public function onXMPDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;
            var video:Video = new Video();

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("video.f4v");
        }

        public function onMetaData(info:Object):void {
            trace("onMetaData fired");
        }

        public function onXMPData(infoObject:Object):void
        {
            trace("onXMPData Fired\n");
            //trace("raw XMP =\n");
            //trace(infoObject.data);
            var cuePoints:Array = new Array();
            var cuePoint:Object;
            var strFrameRate:String;
            var nTracksFrameRate:Number;
            var strTracks:String = "";
            var onXMPXML = new XML(infoObject.data);
            // Set up namespaces to make referencing easier
            var xmpDM:Namespace = new Namespace("http://ns.adobe.com/xmp/1.0/DynamicMedia/");
            var rdf:Namespace = new Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#");
            for each (var it:XML in onXMPXML..xmpDM::Tracks)
            {
```

```
                var strTrackName:String =
it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::trackName;
                var strFrameRateXML:String =
it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::frameRate;
                strFrameRate = strFrameRateXML.substr(1,strFrameRateXML.length);

                nTracksFrameRate = Number(strFrameRate);

                strTracks += it;
            }
            var onXMPTracksXML:XML = new XML(strTracks);
            var strCuepoints:String = "";
            for each (var item:XML in onXMPTracksXML..xmpDM::markers)
            {
                strCuepoints += item;
            }
            trace(strCuepoints);
        }
    }
}
```

В коротком видеофайле с именем startrekintro.f4v этот пример выводит следующие строки отслеживания: Эти строки отображают данные о ключевых точках навигации и событий в метаданных XMP.

```
onMetaData fired
onXMPData Fired

<xmpDM:markers xmlns:xmp="http://ns.adobe.com/xap/1.0/"
xmlns:xmpDM="http://ns.adobe.com/xmp/1.0/DynamicMedia/"
xmlns:stDim="http://ns.adobe.com/xap/1.0/sType/Dimensions#"
xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/"
xmlns:stEvt="http://ns.adobe.com/xap/1.0/sType/ResourceEvent#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#" xmlns:x="adobe:ns:meta/">
  <rdf:Seq>
    <rdf:li>
      <rdf:Description xmpDM:startTime="7695905817600" xmpDM:name="Title1"
xmpDM:type="FLVCuePoint" xmpDM:cuePointType="Navigation">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="Title" xmpDM:value="Star Trek"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
    <rdf:li>
      <rdf:Description xmpDM:startTime="10289459980800" xmpDM:name="Title2"
xmpDM:type="FLVCuePoint" xmpDM:cuePointType="Event">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="William Shatner" xmpDM:value="First Star"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Light Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
  </rdf:Seq>
</xmpDM:markers>
onMetaData fired
```

***Примечание.*** *В метаданных XMP время сохраняется не в секундах, а в тактах DVA. Чтобы вычислить время ключевой точки, необходимо разделить время начала на частоту кадров. Например, время 7695905817600, разделенное на частоту кадров 254016000000 равно 30:30.*

Чтобы полностью просмотреть необработанные данные XMP, включая частоту кадров, уберите идентификаторы комментариев (косые черты «//») перед вторым и третьим выражением `trace()` в начале функции `onXMPData()`.

Дополнительные сведения о метаданных XMP см. в разделах:

- partners.adobe.com/public/developer/xmp/topic.html
- www.adobe.com/devnet/xmp/

## Использование графических метаданных

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Событие `onImageData` отправляет графические данные в виде байтового массива по каналу данных AMF0. Данные могут иметь формат JPEG, PNG или GIF. Определите метод обратного вызова `onImageData()` для обработки этой информации так же, как методы обратного вызова для событий `onCuePoint` и `onMetaData`. В следующем примере используется метод `onImageData()` для получения и вывода на экран изображения.

```
public function onImageData(imageData:Object):void
{
    // display track number
    trace(imageData.trackid);
    var loader:Loader = new Loader();
    //imageData.data is a ByteArray object
    loader.loadBytes(imageData.data);
    addChild(loader);
}
```

## Использование текстовых метаданных

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Событие `onTextData` отправляет текстовые данные по каналу данных AMF0. Текстовые данные представлены в формате UTF-8 и содержат дополнительную информацию о форматировании на основе спецификации синхронизированного текста 3GP. Эта спецификация определяет стандартизированный формат субтитров. Определите метод обратного вызова `onTextData()` для обработки этой информации так же, как методы обратного вызова для событий `onCuePoint` и `onMetaData`. В следующем примере метод `onTextData()` отображает идентификационный номер трека и соответствующий текст.

```
public function onTextData(textData:Object):void
{
    // display the track number
    trace(textData.trackid);
    // displays the text, which can be a null string, indicating old text
    // that should be erased
    trace(textData.text);
}
```

# Отслеживание активности NetStream

**Flash Player 10.3 и более поздних версий, Adobe AIR 2.7 и более поздних версий**

Можно отслеживать активность NetStream для сбора информации, необходимой для поддержки анализа данных и отчетности по использованию мультимедийного содержимого. Возможности отслеживания, о которых пойдет речь в этом разделе, позволяют создавать библиотеки оценки использования мультимедийного содержимого, которые собирают данные, не требуя тесной связи с определенным видеопроигрывателем, используемым для воспроизведения содержимого. Благодаря этому разработчики клиентских приложений смогут выбирать любые видеопроигрыватели при использовании такой библиотеки. Используйте класс NetMonitor для отслеживания создания и активности объектов NetStream в приложении. Класс NetMonitor предоставляет список активных объектов NetStream, существующих в данное время, а также отправляет событие, когда создается объект NetStream.

Объект NetStream отправляет события, перечисленные в следующей таблице, в зависимости от типа воспроизводимого мультимедийного содержимого.

| Событие | Последовательная загрузка | Воспроизведение RTMP-потока | Воспроизведение HTTP-потока |
|---|---|---|---|
| NetStream.Play.Start | Да | Да | Нет |
| NetStream.Play.Stop | Да | Да | Нет |
| NetStream.Play.Complete | Да | Да | Нет |
| NetStream.SeekStart.Notify | Да | Да | Да |
| NetStream.Seek.Notify | Да | Да | Да |
| NetStream.Unpause.Notify | Да | Да | Да |
| NetStream.Unpause.Notify | Да | Да | Да |
| NetStream.Play.Transition | Неприменимо | Да | Неприменимо |
| NetStream.Play.TransitionComplete | Неприменимо | Да | Неприменимо |
| NetStream.Buffer.Full | Да | Да | Да |
| NetStream.Buffer.Flush | Да | Да | Да |
| NetStream.Buffer.Empty | Да | Да | Да |

Объект NetStreamInfo, связанный с экземпляром NetStream, также сохраняет последние объекты метаданных и данных XMP, обнаруженные в мультимедийном содержимом.

Когда мультимедийное содержимое воспроизводится HTTP-потоком, события NetStream.Play.Start, NetStream.Play.Stop и NetStream.Play.Complete не отправляются, так как приложение полностью управляет потоком мультимедиа. Видеопроигрыватель должен формировать и отправлять эти события для HTTP-потоков.

Подобным образом, события NetStream.Play.Transition и NetStream.Play.TransitionComplete не отправляются ни для последовательной загрузки, ни для мультимедийного HTTP-потока. Динамическое переключение скорости передачи данных является функцией RTMP-потока. Если видеопроигрыватель, использующий HTTP-поток, поддерживает подобную функцию, он может формировать и отправлять события перехода.

### Дополнительные разделы справки

Adobe Developer Connection: оценка использования потребления видеоданных в Flash

## Отслеживание событий NetStream

Ценные данные об использовании можно получить из событий двух типов: `netStatus` и `mediaTypeData`. Кроме того, можно использовать таймер для периодической регистрации положения точки воспроизведения NetStream.

События `netStatus` предоставляют информацию, которую можно использовать для определения того, какой объем потока просмотрел пользователь. События перехода буфера и RTMFP-потока также вызывают отправку события `netStatus`.

События `mediaTypeData` предоставляют метаданные и XMP-данные. Событие Netstream.Play.Complete отправляется как событие `mediaTypeData`. Другие встроенные в поток данные, включая ключевые точки, текст и изображения, также можно получить посредством событий `mediaTypeData`.

Следующий пример иллюстрирует создание класса, который отслеживает события состояния и данных, отправляемые активными объектами NetStream в приложении. Как правило, такой класс загружает и собирает данные, которые требуется проанализировать, на сервере.

```
package com.adobe.example
{
    import flash.events.NetDataEvent;
    import flash.events.NetMonitorEvent;
    import flash.events.NetStatusEvent;
    import flash.net.NetMonitor;
    import flash.net.NetStream;

    public class NetStreamEventMonitor
    {
        private var netmon:NetMonitor;
        private var heartbeat:Timer = new Timer( 5000 );

        public function NetStreamEventMonitor()
        {
            //Create NetMonitor object
            netmon = new NetMonitor();
            netmon.addEventListener( NetMonitorEvent.NET_STREAM_CREATE, newNetStream );

            //Start the heartbeat timer
            heartbeat.addEventListener( TimerEvent.TIMER, onHeartbeat );
            heartbeat.start();
        }

        //On new NetStream
        private function newNetStream( event:NetMonitorEvent ):void
        {
            trace( "New Netstream object");
            var stream:NetStream = event.netStream;
            stream.addEventListener(NetDataEvent.MEDIA_TYPE_DATA, onStreamData);
            stream.addEventListener(NetStatusEvent.NET_STATUS, onStatus);
        }

        //On data events from a NetStream object
        private function onStreamData( event:NetDataEvent ):void
        {

            var netStream:NetStream = event.target as NetStream;
            trace( "Data event from " + netStream.info.uri + " at " + event.timestamp );
            switch( event.info.handler )
            {
                case "onMetaData":
                    //handle metadata;
                    break;
                case "onXMPData":
                    //handle XMP;
                    break;
                case "onPlayStatus":
                    //handle NetStream.Play.Complete
                case "onImageData":
                    //handle image
                    break;
                case "onTextData":
```

```
                //handle text
                break;
            default:
                //handle other events

        }
    }

    //On status events from a NetStream object
    private function onStatus( event:NetStatusEvent ):void
    {
        trace( "Status event from " + event.target.info.uri + " at " + event.target.time );
        //handle status events
    }
    //On heartbeat timer
    private function onHeartbeat( event:TimerEvent ):void
    {
        var streams:Vector.<NetStream> = netmon.listStreams();
        for( var i:int = 0; i < streams.length; i++ )
        {
            trace( "Heartbeat on " + streams[i].info.uri + " at " + streams[i].time );
            //handle heartbeat event
        }
    }

  }
}
```

## Определение домена проигрывателя

URL-адрес и домен веб-страницы, на которой пользователь просматривает мультимедийное содержимое, не всегда легкодоступны. Если это позволяет веб-сайт, выступающий в роли хоста, можно использовать класс ExternalInterface для получения точного URL-адреса. Однако некоторые веб-сайты, на которых разрешено использовать сторонние видеопроигрыватели, не позволяют применять класс ExternalInterface. В таких случаях можно получить домен текущей веб-страницы из свойства `pageDomain` класса Security. Полный URL-адрес не разглашается из соображений безопасности и конфиденциальности пользователей.

Домен страницы можно получить из статического свойства `pageDomain` класса Security:

```
var domain:String = Security.pageDomain;
```

# Дополнительные темы о видеофайлах

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Следующие темы освещают некоторые специфические вопросы работы с FLV-файлами.

## О настройке FLV-файлов с целью размещения на сервере

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При работе с FLV-файлами может возникнуть необходимость настроить сервер для работы с файлами формата FLV. MIME (Multipurpose Internet Mail Extensions, многоцелевые расширения почты в Интернете) — это стандартизированная спецификация данных, которая позволяет отправлять через Интернет файлы, не содержащие ASCII-символы. Веб-обозреватели и клиенты электронной почты настроены для интерпретации различных MIME-типов, поэтому они могут отправлять и получать видео, аудио, графику и текст с форматированием. Для загрузки FLV-файлов с веб-сервера, может потребоваться зарегистрировать расширение файла и MIME-тип на веб-сервере, поэтому рекомендуется предварительно проверить документацию веб-сервера. Для FLV-файлов используется MIME-тип `video/x-flv`. Полные данные о типе FLV-файла выглядят следующим образом:

- Mime-тип: video/x-flv
- Расширение файла: .flv
- Обязательные параметры: нет
- Дополнительные параметры: нет
- Сведения о кодировке: FLV-файлы являются двоичными файлами; для некоторых приложений может потребоваться указать подтип приложения/потока октетов
- Проблемы безопасности: нет
- Опубликованная спецификация: www.adobe.com/go/video_file_format_ru

Корпорация Microsoft изменила процесс обработки потокового медиасодержимого веб-сервером Microsoft Internet Information Services (IIS) 6.0 (по сравнению с предыдущими версиями). В предыдущих версиях IIS не требовалось вносить изменений для потоковой передачи видео Flash. В IIS 6.0, веб-сервере по умолчанию для Windows 2003, требуется указать MIME-тип, чтобы FLV-файлы распознавались как потоковое медиасодержимое.

Когда SWF-файл, содержащий внешний потоковый FLV-файл, помещаются на сервер Microsoft Windows Server® 2003 и открываются в обозревателе, SWF-воспроизводится правильно, но FLV-видео не передается потоком. Эта проблема затрагивает все FLV-файлы, размещаемые на сервере Windows Server 2003, включая созданные в более ранних версиях инструмента разработки Flash и Macromedia Flash Video Kit для Dreamweaver MX 2004 от компании Adobe. Эти файлы работают правильно при тестировании в других операционных системах.

Сведения о настройке Microsoft Windows 2003 и Microsoft IIS Server 6.0 для передачи потокового видео FLV см. на странице www.adobe.com/go/tn_19439_ru.

## О позиционировании локальных FLV-файлов в системе Macintosh

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если при попытке воспроизвести локальный FLV-файл с несистемного диска компьютера Apple® Macintosh® путь указан с использованием относительной косой черты (/), видео не показывается. К несистемным дискам, помимо прочего, относятся компакт-диски, жесткие диски, разбитые на разделы, съемные носители и подключенные устройства хранения данных.

***Примечание.*** *Причиной этой проблемы является ограничение операционной системы, а не проигрывателя Flash Player или среды AIR.*

Чтобы воспроизвести FLV-файл с несистемного диска на компьютере Macintosh, ссылка на него должна содержать абсолютный путь с синтаксисом через двоеточие (:), а не косую черту (/). Ниже представлены отличия между этими двумя типами синтаксиса

- Синтаксис через косую черту**:** myDrive/myFolder/myFLV.flv
- Синтаксис через двоеточие**:** (Mac OS®) myDrive:myFolder:myFLV.flv

# Пример видеоданных: Video Jukebox

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Следующий пример создает простой видеоавтомат (Video Jukebox), который динамически загружает список видеофайлов для воспроизведения в последовательном порядке. Так можно создать приложение, которое дает пользователю возможность просмотреть серию учебных видеоматериалов или указывает, какие рекламные ролики нужно показать перед воспроизведением видеофайла, запрашиваемого пользователем. В этом примере используются следующие функции ActionScript 3.0:

- обновление точки воспроизведения на основе прогресса воспроизведения видеофайла;
- прослушивание и анализ метаданных видеофайла;
- обработка определенных кодов в сетевом потоке;
- загрузка, воспроизведение, приостановка и остановка динамического загруженного FLV-файла;
- изменение размеров объекта Video в списке отображения на основе метаданных сетевого потока.

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения VideoJukebox находятся в папке Samples/VideoJukebox. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| VideoJukebox.fla<br>или<br>VideoJukebox.mxml | Основной файл приложения для Flex (MXML) или Flash (FLA). |
| VideoJukebox.as | Класс, обеспечивающий основную функциональность приложения. |
| playlist.xml | Файл, в котором перечислены видеофайлы, предназначенные для загрузки в видеоавтомат. |

## Загрузка внешнего файла со списком видеофайлов для воспроизведения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Внешний файл playlist.xml указывает, какие видеофайлы необходимо загрузить и в каком порядке их нужно воспроизводить. Для загрузки XML-файла необходимо использовать объекты URLLoader и URLRequest, как показано в следующем коде.

```
uldr = new URLLoader();
uldr.addEventListener(Event.COMPLETE, xmlCompleteHandler);
uldr.load(new URLRequest(PLAYLIST_XML_URL));
```

Этот код помещается в конструктор класса VideoJukebox, поэтому файл загружается перед выполнением основной части кода. Сразу после завершения загрузки XML-файла вызывается метод `xmlCompleteHandler()`, который преобразует внешний файл в объект XML, как показано в следующем коде.

```
private function xmlCompleteHandler(event:Event):void
{
    playlist = XML(event.target.data);
    videosXML = playlist.video;
    main();
}
```

Объект XML с именем `playlist` содержит необработанные данные в формате XML из внешнего файла, в то время как XML видео является объектом XMLList, который содержит только видеоузлы. Пример файла playlist.xml можно найти в следующем коде.

```
<videos>
    <video url="video/caption_video.flv" />
    <video url="video/cuepoints.flv" />
    <video url="video/water.flv" />
</videos>
```

В завершение, метод `xmlCompleteHandler()` вызывает функцию `main()`, которая создает экземпляры различных компонентов в списке отображения, а также объекты NetConnection и NetStream, используемые для загрузки внешних FLV-файлов.

## Создание пользовательского интерфейса

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для создания пользовательского интерфейса необходимо перетащить пять экземпляров Button в список отображения, присвоив им следующие имена: `playButton`, `pauseButton`, `stopButton`, `backButton` и `forwardButton`.

Для каждого экземпляра Button необходимо назначить обработчик события `click`, как показано в следующем коде.

```
playButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
pauseButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
stopButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
backButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
forwardButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
```

Метод `buttonClickHandler()` использует инструкцию switch, чтобы определить, какой экземпляр Button, получил щелчок мыши, как показано в следующем коде.

```
private function buttonClickHandler(event:MouseEvent):void
{
    switch (event.currentTarget)
    {
        case playButton:
            ns.resume();
            break;
        case pauseButton:
            ns.togglePause();
            break;
        case stopButton:
            ns.pause();
            ns.seek(0);
            break;
        case backButton:
            playPreviousVideo();
            break;
        case forwardButton:
            playNextVideo();
            break;
    }
}
```

Затем добавьте в список отображения объект Slider и присвойте ему имя экземпляра `volumeSlider`. Следующий код задает свойству `liveDragging` экземпляра Slider значение `true` и определяет прослушиватель для события `change`.

```
volumeSlider.value = volumeTransform.volume;
volumeSlider.minimum = 0;
volumeSlider.maximum = 1;
volumeSlider.snapInterval = 0.1;
volumeSlider.tickInterval = volumeSlider.snapInterval;
volumeSlider.liveDragging = true;
volumeSlider.addEventListener(SliderEvent.CHANGE, volumeChangeHandler);
```

Добавьте в список отображения объект ProgressBar и присвойте ему имя экземпляра `positionBar`. Задайте его свойству `mode` значение manual (вручную), как показано в следующем коде.

```
positionBar.mode = ProgressBarMode.MANUAL;
```

В завершение, добавьте в список отображения объект Label и присвойте ему имя экземпляра `positionLabel`. Значение этого экземпляра Label будет задаваться экземпляром Timer.

## Прослушивание метаданных объекта Video

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Когда проигрыватель Flash Player находит метаданные для каждого загруженного видеофайла, вызывается обработчик обратного вызова `onMetaData()` для свойства `client` объекта NetStream. Следующий код создает экземпляр Object и определяет указанный метод обратного вызова.

```
client = new Object();
client.onMetaData = metadataHandler;
```

Метод `metadataHandler()` копирует свои данные в свойство meta, ранее определенное в коде. Это позволяет получать метаданные для текущего видео в любой момент в рамках всего приложения. Затем размер объекта Video в рабочей области изменяется в соответствии с шириной и высотой, указанными в метаданных. В завершение, выполняется перемещение и изменение размера экземпляра ProgressBar с именем positionBar в соответствии с размером текущего воспроизводимого видео. Ниже приводится полный код метода `metadataHandler()`.

```
private function metadataHandler(metadataObj:Object):void
{
    meta = metadataObj;
    vid.width = meta.width;
    vid.height = meta.height;
    positionBar.move(vid.x, vid.y + vid.height);
    positionBar.width = vid.width;
}
```

## Динамическая загрузка видеоданных

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для динамической загрузки каждого видеофайла приложение использует объекты NetConnection и NetStream. Следующий код создает объект NetConnection и передает значение `null` методу `connect()`. Указав значение `null`, Flash Player подключается к видеофайлу на локальном сервере, а не на удаленном, таком как Flash Media Server.

Следующий код создает экземпляры NetConnection и NetStream, определяет прослушиватель для события `netStatus` и назначает экземпляр Object с именем`client` свойству `client`.

```
nc = new NetConnection();
nc.connect(null);

ns = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
ns.client = client;
```

Метод `netStatusHandler()` вызывается каждый раз, когда изменяется состояние видео, в том числе при запуске, остановке или буферизации воспроизведения, а также когда не удается найти видеопоток. В следующем коде представлен обработчик событий `netStatusHandler()`.

```
private function netStatusHandler(event:NetStatusEvent):void
{
    try
    {
        switch (event.info.code)
        {
            case "NetStream.Play.Start":
                t.start();
                break;
            case "NetStream.Play.StreamNotFound":
            case "NetStream.Play.Stop":
                t.stop();
                playNextVideo();
                break;
        }
    }
    catch (error:TypeError)
    {
        // Ignore any errors.
    }
}
```

Предыдущий пример оценивает свойство code объекта info и отфильтровывает коды «NetStream.Play.Start», «NetStream.Play.StreamNotFound» или «NetStream.Play.Stop». Остальные коды игнорируются Если запускается сетевой поток, код запускает экземпляр Timer, который обновляет точку воспроизведения. Если сетевой поток не удается обнаружить или он остановлен, экземпляр Timer останавливается и приложение пытается воспроизвести следующий видеофайл в списке воспроизведения.

Каждый раз при выполнении экземпляра Timer экземпляр ProgressBar с именем `positionBar` обновляет текущую позицию, вызывая метод `setProgress()` класса ProgressBar, также обновляется экземпляр Label с именем `positionLabel`, отражая прошедшее время и общее время для текущего видеофайла.

```
private function timerHandler(event:TimerEvent):void
{
    try
    {
        positionBar.setProgress(ns.time, meta.duration);
        positionLabel.text = ns.time.toFixed(1) + " of " meta.duration.toFixed(1) + " seconds";
    }
    catch (error:Error)
    {
        // Ignore this error.
    }
}
```

## Управление громкостью видео

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Управлять громкостью динамически загружаемого видеофайла можно с помощью свойства `soundTransform` объекта NetStream. Приложение VideoJukebox позволяет изменять уровень громкости путем изменения значения экземпляра Slider с именем `volumeSlider`. Следующий код демонстрирует изменение громкости путем присвоения значения компонента Slider объекту SoundTransform, который затем передается свойству `soundTransform` объекта NetStream.

```
private function volumeChangeHandler(event:SliderEvent):void
{
    volumeTransform.volume = event.value;
    ns.soundTransform = volumeTransform;
}
```

## Управление воспроизведением видео

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Остальной код приложения управляет воспроизведением, когда видеопоток заканчивается или пользователь переходит к предыдущему или следующему видеофайлу.

Следующий метод получает URL-адрес видеофайла из объекта XMLList для выделенного в данный момент индекса.

```
private function getVideo():String
{
    return videosXML[idx].@url;
}
```

Метод `playVideo()` вызывает метод `play()` объекта NetStream, чтобы загрузить выделенный в данный момент видеофайл.

```
private function playVideo():void
{
    var url:String = getVideo();
    ns.play(url);
}
```

Метод `playPreviousVideo()` уменьшает текущий индекс видеофайла, вызывает метод `playVideo()` для загрузки нового видеофайла и отображает строку прогресса.

```
private function playPreviousVideo():void
{
    if (idx > 0)
    {
        idx--;
        playVideo();
        positionBar.visible = true;
    }
}
```

Последний метод, `playNextVideo()`, увеличивает индекс видео и вызывает метод `playVideo()`. Если текущий видеофайл является последним в списке воспроизведения, вызывается метод `clear()` для объекта Video, а свойству `visible` экземпляра ProgressBar задается значение `false`.

```
private function playNextVideo():void
{
    if (idx < (videosXML.length() - 1))
    {
        idx++;
        playVideo();
        positionBar.visible = true;
    }
    else
    {
        idx++;
        vid.clear();
        positionBar.visible = false;
    }
}
```

# Использование класса StageVideo для визуализации с аппаратным ускорением

**Flash Player 10.2 или более поздней версии**

Проигрыватель Flash Player оптимизирует производительность работы с видео за счёт использования аппаратного ускорения для декодирования H.264. Дополнительно производительность можно повысить с помощью интерфейса API StageVideo. Класс StageVideo позволяет программе использовать возможности визуализации с аппаратным ускорением.

Среды выполнения, которые поддерживают API-интерфейс StageVideo:

- Flash Player 10.2 или более поздней версии

***Примечание.*** *В среде Flash Player 11.4 или AIR 3.4 и более поздних версий можно использовать входной сигнал с камеры вместе с классом StageVideo.*

Исходный код для загрузки и дополнительные сведения о функции видео рабочей области см. в статье Приступая к работе с видео рабочей области.

Учебник по быстрому запуску StageVideo см. в статье Работа с видео рабочей области.

## Сведения об аппаратном ускорении с использованием класса StageVideo

Визуализация с аппаратным ускорением, включающая масштабирование видео, преобразование цветов и блитирование, повышает производительность декодирования с аппаратным ускорением. На устройствах, поддерживающих (аппаратное) ускорение с использованием графического процессора, с помощью объекта flash.media.StageVideo можно обрабатывать видео непосредственно аппаратными средствами устройства. При прямой обработке ресурсы центрального процессора освобождаются для выполнения других задач, в то время как графический процессор занимается обработкой видео. С другой стороны, существующий класс Video обычно использует программную визуализацию. Программная визуализация выполняется центральным процессором, для чего может использоваться значительная часть ресурсов системы.

В настоящее время полное ускорение с использованием графического процессора поддерживается только несколькими устройствами. Тем не менее функция видео рабочей области позволяет приложениям в максимальной степени использовать преимущества доступного на устройствах функционала аппаратного ускорения.

Класс StageVideo не делает класс Video устаревшим. Использование обоих классов позволяет достичь оптимального качества при отображении видео, поддерживаемого ресурсами устройства, в любой момент времени. Воспользоваться преимуществами аппаратного ускорения можно за счет настройки в приложении прослушивания соответствующих событий и переключения между классами StageVideo и Video в требуемые моменты времени.

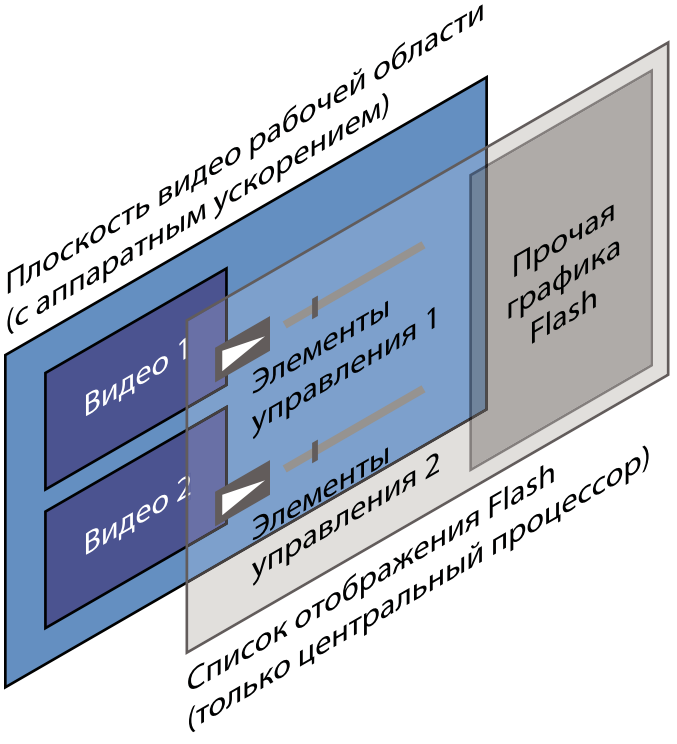
Класс StageVideo накладывает некоторые ограничения на использование видео. Перед реализацией StageVideo ознакомьтесь с рекомендациями и убедитесь, что они применимы к приложению. Если ограничения допустимы, используйте класс StageVideo при обнаружении средой Flash Player возможностей визуализации с аппаратным ускорением. См. раздел ««Рекомендации и ограничения» на странице 551».

## Параллельные плоскости: список отображения для видео рабочей области и Flash

В модели видео рабочей области приложение Flash Player можно отделить видео из списка отображения. Приложение Flash Player разделяет составной экран на две плоскости с порядком размещения по оси Z:

**Плоскость видео рабочей области** Плоскость видео рабочей области находится на заднем плане. В ней отображается только видео с аппаратным ускорением. В результате этого плоскость недоступна, если аппаратное ускорение не поддерживается или не доступно на устройстве. В ActionScript объекты StageVideo обрабатывают видео, воспроизводимые в плоскости видео рабочей области.

**Плоскость списка отображения Flash** Элементы из списка отображения Flash размещаются на плоскости, находящейся перед плоскостью видео рабочей области. В список отображения включены все элементы, визуализация которых осуществляется с помощью среды выполнения, включая элементы управления воспроизведением. Когда аппаратная визуализация недоступна, видео может воспроизводиться только в этой плоскости с использованием объекта класса Video. Видео рабочей области всегда отображается позади графики из списка отображения Flash.




*Плоскости отображения видео*

Объект StageVideo отображается в фиксированной прямоугольной области окна. Под плоскостью видео рабочей области нельзя размещать другие объекты. Однако над плоскостью видео рабочей области можно разместить другую графику с помощью списка отображения Flash. Видео рабочей области воспроизводится одновременно со списком отображения Таким образом, эти два механизма можно применять вместе для создания унифицированных визуальных эффектов с использованием двух отдельных плоскостей. Например, переднюю плоскость можно использовать для элементов управления воспроизведением, которые контролируют видео рабочей области в задней плоскости.

## Видео рабочей плоскости и кодек H.264

В приложениях Flash Player аппаратное ускорение видео реализуется в два этапа:

1. Кодирование видео в формат H.264
2. Реализация API-интерфейса StageVideo

Для получения наилучшего результата необходимо выполнить оба этапа. Кодек H.264 позволяет максимально использовать преимущества аппаратного ускорения: от декодирования видео до его визуализации.

В видео рабочей области исключается повторное считывание данных из графического процессора в центральный процессор. Другими словами графический процессор больше не отправляет декодированные кадры в центральный процессор для совмещения с объектами списка отображения. Вместо этого графический процессор передает битовые блоки декодированных и визуализированных кадров непосредственно на экран под объекты списка отображения. В этом методе уменьшается использование центрального процессора и памяти, а также улучшается точность пикселей.

### Дополнительные разделы справки

«Сведения о форматах видео» на странице 509

## Рекомендации и ограничения

При воспроизведении видео в полноэкранном режиме видео рабочей области всегда доступно, если устройством поддерживается аппаратное ускорение. Однако Flash Player также выполняется в браузере. В контексте браузера на доступность видео рабочей области влияет параметр `wmode`. Если необходимо использовать видео рабочей области, рекомендуется всегда устанавливать `wmode="direct"`. Видео рабочей области несовместимо с другими параметрами `wmode` в других режимах, кроме полноэкранного. Данное ограничение означает, что в среде выполнения видео рабочей области может становиться недоступным в непредсказуемых случаях. Например, если во время воспроизведения видео пользователь выходит из полноэкранного режима, вновь происходит возврат в контекст браузера. Если для параметра `wmode` браузера не установлено значение `"direct"`, видео рабочей области может неожиданно стать недоступным. Проигрыватель Flash Player передает изменения контекста воспроизведения приложений с помощью набора событий. При реализации API-интерфейса StageVideo нужно сделать резервную копию объекта Video на случай, если видео рабочей области станет недоступным.

Прямая связь с оборудованием означает, что видео рабочей области ограничивает некоторые функции видео. Видео рабочей области накладывает следующие ограничения:

- Для каждого SWF-файла проигрыватель Flash Player разрешает не более чем четырем объектам StageVideo отображать видео одновременно. Однако фактическое ограничение может быть ниже, в зависимости от аппаратных ресурсов устройства.
- Хронометраж видео не синхронизируется с хронометражем содержимого, которое отображается в среде выполнения.

- Область отображения видео может иметь только прямоугольную форму. Нельзя использовать более сложные области отображения, например области, имеющие эллиптическую или неправильную форму.
- Нельзя поворачивать видео.
- Вы не можете растровый кэш видеоролик или использовать объект BitmapData, чтобы получить доступ к нему.
- Нельзя применять фильтры к видео.
- Нельзя применять преобразования цветов к видео.
- Нельзя применять значение альфа к видео.
- Режимы наложения, которые применяются к объектам в плоскости списка отображения видео, не применяются к видео рабочей области.
- Видео можно разместить только в пределах полных пикселей.
- Хотя рендеринг с использованием графического процессора является лучшим решением, доступным для аппаратных средств конкретного устройства, он не обеспечивает 100 % «идентичности пикселей». Небольшие варианты имеют место из-за разниц платформы и драйвера.
- Несколько устройств не делают поддержки все необходимые цветовые пространства. Например, некоторые устройства не делают Купленной 709 поддержки, стандарт H.264. В таких случаях для быстрого отображения можно использовать стандарт BT.601.
- Вы не можете использовать видеоролик рабочей область с параметрами WMODE, например, нормальный, непрозрачный, или прозрачный. Видео рабочей области поддерживает параметр `WMODE=direct` в других режимах, кроме полноэкранного. Параметр WMODE не работает в Safari 4 и более поздних версиях, а также в IE 9 и более поздних версиях.

В большинстве случаев эти ограничения не влияют на работу проигрывателей видео. Если эти ограничения являются приемлемыми, используйте видео рабочей области во всех возможных случаях.

### Дополнительные разделы справки

«Работа в полноэкранном режиме» на странице 179

## Использование API-интерфейсов StageVideo

Видео рабочей области представляет собой механизм среды выполнения, который улучшает качество воспроизведения видео и повышает производительность устройства. Среда выполнения создает и поддерживает работу данного механизма. Разработчик должен настроить конфигурацию приложения, чтобы можно было воспользоваться всеми его преимуществами.

Для использования видео рабочей области необходимо реализовать среду обработчиков событий, которые будут обнаруживать, что видео становится недоступным. При получении уведомления о том, что видео рабочей области стало доступным, извлеките объект StageVideo из свойства `Stage.stageVideos`. Среда выполнения заполняет данный объект Vector, используя один или несколько объектов StageVideo. Затем для отображения потокового видео вместо объекта Video можно использовать один из предоставленных объектов StageVideo.

В проигрывателе Flash Player при получении уведомления о том, что видео рабочей области больше недоступно, переключите поток видео на объект Video.

***Примечание.*** *Объекты StageVideo нельзя создавать.*

## Свойство Stage.stageVideos

Свойство `Stage.stageVideos` — это объект Vector, который предоставляет доступ к экземплярам StageVideo. Этот вектор может содержать до четырех объектов StageVideo, в зависимости от аппаратных средств и системных ресурсов. На мобильных устройствах это может быть один объект или ни одного.

Когда видео рабочей области не доступно, этот вектор не содержит объектов. Чтобы избежать возникновения ошибок среды выполнения, убедитесь, что обращение к элементам этого вектора происходит только когда последнее событие `StageVideoAvailability`, сообщает о доступности видео рабочей области.

## События StageVideo

API-интерфейс StageVideo включает следующие события:

**StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY** Отправляется при изменении свойства `Stage.stageVideos`. Свойство `StageVideoAvailabilityEvent.availability` имеет значение `AVAILABLE` или `UNAVAILABLE`. Используйте это событие, чтобы определить, содержит ли свойство `stageVideos` объекты StageVideo, вместо того, чтобы напрямую проверять длину вектора `Stage.stageVideos`.

**StageVideoEvent.RENDER_STATE** Отправляется, если объект NetStream или Camera был присоединен к объекту StageVideo и сейчас воспроизводится. Указывает тип используемого сейчас декодирования: аппаратное, программное или недоступно (ничего не отображается). Цель события содержит свойства `videoWidth` и `videoHeight`, которые можно использовать для изменения размера окна просмотра видео.

***Важная информация.*** *Для координат, полученных от целевого объекта StageVideo, используются координаты рабочей области, поскольку они не являются частью стандартного списка отображения.*

**VideoEvent.RENDER_STATE** Отправляется, когда используется объект Video. Указывает тип используемого декодирования: программное или с аппаратным ускорением. Если это событие показывает декодирование с аппаратным ускорением, переключитесь на объект StageVideo, если это возможно. Цель события видео содержит свойства `videoWidth` и `videoHeight`, которые можно использовать для изменения размера окна просмотра видео.

## Рабочий процесс для реализации функции StageVideo

Для реализации функции StageVideo выполните указанные ниже действия верхнего уровня.

1. Прослушивайте событие `StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY`, чтобы узнать, когда вектор изменился`Stage.stageVideos` . См. раздел «Использование события StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY» на странице 555.
2. Если событие `StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY` сообщает о доступности видео рабочей области, используйте объект Vector `Stage.stageVideos` в обработчике событий для получения доступа к объекту StageVideo.
3. Присоедините объект NetStream с использованием функции `StageVideo.attachNetStream()` или присоедините объект Camera с использованием функции `StageVideo.attachCamera()`.
4. Воспроизведение видео использование `NetStream.play ()`.
5. Прослушивайте событие `StageVideoEvent.RENDER_STATE` на объекте StageVideo, чтобы определить состояние воспроизведения видео. Получение этого события также означает, что были инициализированы или изменены свойства ширины и высоты видео. См. раздел ««Использование событий StageVideoEvent.RENDER_STATE и VideoEvent.RENDER_STATE» на странице 556».

6 Прослушивайте событие `VideoEvent.RENDER_STATE` для объекта Video. Это событие предлагает те же самые состояния, что и событие `StageVideoEvent.RENDER_STATE`. Таким образом, его можно также использовать для определения доступности ускорения графического процессора. Получение этого события также означает, что были инициализированы или изменены свойства ширины и высоты видео. См. раздел ««Использование событий StageVideoEvent.RENDER_STATE и VideoEvent.RENDER_STATE» на странице 556».

## Инициализация прослушивателей событий StageVideo

Настройте прослушиватели событий StageVideoAvailabilityEvent и VideoEvent при инициализации приложения. Например, эти прослушиватели можно инициализировать в обработчике событий `flash.events.Event.ADDED_TO_STAGE`. Данное событие гарантирует, что приложение отображается в рабочей области:

```
public class SimpleStageVideo extends Sprite
    private var nc:NetConnection;
    private var ns:NetStream;

    public function SimpleStageVideo()
    {
        // Constructor for SimpleStageVideo class
        // Make sure the app is visible and stage available
        addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
    }

    private function onAddedToStage(event:Event):void
    {
        //...
        // Connections
        nc = new NetConnection();
        nc.connect(null);
        ns = new NetStream(nc);
        ns.addEventListener(NetStatusEvent.NET_STATUS, onNetStatus);
        ns.client = this;

        // Screen
        video = new Video();
        video.smoothing = true;

        // Video Events
        // the StageVideoEvent.STAGE_VIDEO_STATE informs you whether
        // StageVideo is available
        stage.addEventListener(StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY,
            onStageVideoState);
        // in case of fallback to Video, listen to the VideoEvent.RENDER_STATE
        // event to handle resize properly and know about the acceleration mode running
        video.addEventListener(VideoEvent.RENDER_STATE, videoStateChange);
        //...
    }
```

## Использование события StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY

В обработчике `StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY` укажите, какой объект, Video или StageVideo, будет использоваться в зависимости от доступности StageVideo. Если для свойства `StageVideoAvailabilityEvent.availability` установлено значение `StageVideoAvailability.AVAILABLE`, используйте StageVideo. В этом случае у вас будет уверенность, что вектор Stage.stageVideos содержит один или несколько объектов StageVideo. Получите объект StageVideo из свойства `Stage.stageVideos` и присоедините к нему объект NetStream. Поскольку объекты StageVideo всегда отображаются на заднем плане, удалите все существующие объекты Video (всегда находятся на переднем плане). С помощью обработчика этого события можно также добавить прослушиватель для события `StageVideoEvent.RENDER_STATE`.

Если для свойства `StageVideoAvailabilityEvent.availability` установлено значение `StageVideoAvailability.UNAVAILABLE`, не используйте StageVideo и не обращайтесь к вектору `Stage.stageVideos`. В этом случае присоедините объект NetStream к объекту Video. В конце добавьте объект StageVideo или Video в рабочую область и вызовите `NetStream.play()`.

В коде ниже показано, как обрабатывать событие `StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY`.

```
private var sv:StageVideo;
private var video:Video;

private function onStageVideoState(event:StageVideoAvailabilityEvent):void
{
    // Detect if StageVideo is available and decide what to do in toggleStageVideo
    toggleStageVideo(event.availability == StageVideoAvailability.AVAILABLE);
}

private function toggleStageVideo(on:Boolean):void
{
    // To choose StageVideo attach the NetStream to StageVideo
    if (on)
    {
        stageVideoInUse = true;
        if ( sv == null )
        {
            sv = stage.stageVideos[0];
            sv.addEventListener(StageVideoEvent.RENDER_STATE, stageVideoStateChange);
                sv.attachNetStream(ns);
        }

        if (classicVideoInUse)
        {
            // If you use StageVideo, remove from the display list the
            // Video object to avoid covering the StageVideo object
```

```
            // (which is always in the background)
            stage.removeChild ( video );
            classicVideoInUse = false;
        }
    } else
    {
        // Otherwise attach it to a Video object
        if (stageVideoInUse)
            stageVideoInUse = false;
        classicVideoInUse = true;
        video.attachNetStream(ns);
        stage.addChildAt(video, 0);
    }

    if ( !played )
    {
        played = true;
        ns.play(FILE_NAME);
    }
}
```

***Важная информация.*** *При первом обращении приложения к элементу вектора в Stage.stageVideos[0] свойство «rect» имеет по умолчанию значение 0,0,0,0, а в свойствах «pan» и «zoom» используются значения по умолчанию. Для этих параметров всегда можно установить свои значения. Свойства* `videoWidth` *и* `videoHeight` *для цели события* `StageVideoEvent.RENDER_STATE` *или* `VideoEvent.RENDER_STATE` *можно использовать для расчета размера окна просмотра видео.*

Полный исходный код этого примера приложения можно загрузить на странице «Приступая к работе со Stage Video».

## Использование событий StageVideoEvent.RENDER_STATE и VideoEvent.RENDER_STATE

Объекты StageVideo и Video отправляют события, чтобы сообщить программе об изменении среды визуализации. Этими событиями являются `StageVideoEvent.RENDER_STATE` и `VideoEvent.RENDER_STATE`.

Объект StageVideo или Video объект отправляет событие состояния визуализации, когда присоединяется объект NetStream и запускается воспроизведение. Объект также отправляет это событие при изменении среды визуализации, например после изменения размеров окна просмотра видео. Используйте эти уведомления, чтобы восстановить для окна просмотра текущие значения `videoHeight` и `videoWidth` целевого объекта события.

Сообщается о следующих состояниях визуализации:

- `RENDER_STATUS_UNAVAILABLE`
- `RENDER_STATUS_SOFTWARE`
- `RENDER_STATUS_ACCELERATED`

Состояния визуализации показывают, когда используется декодирование с аппаратным ускорением (независимо от класса воспроизводимого в настоящее время видео). По свойству `StageVideoEvent.status` можно узнать о доступности необходимого декодирования. Если это свойство имеет значение «unavailable», объект StageVideo не сможет воспроизвести видео. Этот статус означает, что нужно немедленно заново присоединить объект NetStream к объекту Video. Другие статусы сообщают о текущих условиях визуализации приложения.

В таблице ниже описывается влияние всех значений состояния визуализации для объектов StageVideoEvent и VideoEvent в проигрывателе Flash Player.

| | **VideoStatus.ACCELERATED** | **VideoStatus.SOFTWARE** | **VideoStatus.UNAVAILABLE** |
|---|---|---|---|
| StageVideoEvent | Декодирование и визуализация выполняются на аппаратном уровне. (Наилучшая производительность.) | Визуализация выполняется на аппаратном уровне, а декодирование — на программном. (Приемлемое быстродействие.) | Для обработки видео недоступны ресурсы графического процессора, и ничего не отображается. **Переход к объекту Video.** |
| VideoEvent | Визуализация выполняется на программном уровне, а декодирование — на аппаратном. (Приемлемая производительности только на современных настольных системах. Низкая производительность полноэкранном режиме.) | Визуализация и декодирование выполняются на программном уровне. (Наихудшая производительность. Низкая производительность полноэкранном режиме.) | (Н/Д) |

## Цветовые пространства

Видео рабочей области использует аппаратные возможности для обеспечения поддержки цветовых пространств. SWF-содержимое может содержать метаданные с указанием предпочтительного цветового пространства. Однако возможность использования цветового пространства определяется графическим оборудованием устройства. Одни устройства могут поддерживать несколько цветовых пространств, тогда как другие могут не поддерживать ни одного. Если оборудование не поддерживает запрошенное цветовое пространство, Flash Player попытается найти наиболее похожее цветовое пространства из набора поддерживаемых.

Чтобы запросить, какие цветовые пространства поддерживает оборудование, используйте свойство `StageVideo.colorSpaces`. Данное свойство возвращает список поддерживаемых цветовых пространств в векторе String:

```
var colorSpace:Vector.<String> = stageVideo.colorSpaces();
```

Чтобы узнать, какое цветовое пространство используется в воспроизводимом видео, проверьте свойство `StageVideoEvent.colorSpace`. Проверьте это свойство в обработчике события `StageVideoEvent.RENDER_STATE`:

```
var currColorSpace:String;

//StageVideoEvent.RENDER_STATE event handler
private function stageVideoRenderState(event:Object):void
{
    //...
    currColorSpace = (event as StageVideoEvent).colorSpace;
    //...
}
```

Если Flash Player не может найти замену для неподдерживаемого цветового пространства, будет использовано цветовое пространство по умолчанию — BT.601. Например, видео с кодировкой H.264 обычно использует цветовое пространство BT.709. Если BT.709 не поддерживается на устройстве, свойство `colorSpace` возвращает `"BT601"`. Если свойство `StageVideoEvent.colorSpace` имеет значение `"unknown"`, это означает, что оборудование не предоставляет возможности запроса цветового пространства.

Если текущее цветовое пространство окажется неподходящим для приложения, можно переключиться с объекта StageVideo на объект Video. Класс Video поддерживает все цветовые пространства за счет программного совмещения.

# Глава 26. Работа с камерами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В качестве источника видеоданных для отображения и манипулирования с помощью ActionScript можно использовать камеру, подключенную к компьютеру пользователя. Класс Camera представляет собой механизм, встроенный в ActionScript и предназначенный для работы с камерой компьютера или устройства.

На мобильных устройствах можно также использовать класс CameraUI. Класс CameraUI запускает отдельное приложение камеры, чтобы предоставить пользователю возможность захватывать неподвижные изображения и видео. По завершении захвата приложение может обращаться к данным изображения или видео посредством объекта MediaPromise.

### Дополнительные разделы справки

Кристиан Кантрелл (Christian Cantrell): Использование CameraUI в рамках межплатформного подхода

Микаэль Шез (Michaël CHAIZE): Android, AIR и камера

## Понимание класса Camera

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Объект Camera позволяет подключить локальную камеру пользователя и передавать видео локально (на экран пользователя) или удаленно на сервер (такой как Flash Media Server).

С помощью класса Camera можно получить доступ к следующим типам информации о камере пользователя:

- Какие камеры, из установленных на компьютере или устройстве пользователя, доступны
- установлена ли камера;
- разрешен или запрещен проигрывателю Flash Player доступ к камере пользователя;
- какая камера активна в данный момент;
- ширина и высота захватываемого видео.

Класс Camera включает несколько полезных методов и свойств для работы с камерами. Например, статическое свойство `Camera.names` содержит массив имен камер, в данный момент установленных на компьютере пользователя. Также для отображения имени текущей активной камеры можно использовать свойство `name`.

***Примечание.*** *При передачи видеопотока с камеры по сети всегда следует обрабатывать сетевые прерывания. Сетевые прерывания могут возникать по многим причинам, особенно на мобильных устройствах.*

# Отображение содержимого камеры на экране

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для подключения к камере может потребоваться не такой длинный код, как для загрузки видео с помощью классов NetConnection и NetStream. Класс Camera может также создавать трудности в связи с тем, что проигрыватель Flash Player требует, чтобы пользователь дал разрешение на подключение к камере, к ней можно будет получить доступ.

Следующий код демонстрирует, как с помощью класса Camera можно подключиться к локальной камере пользователя.

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

***Примечание.*** *Класс Camera не имеет собственного конструктора. Для создания нового экземпляра Camera используется статический метод* `Camera.getCamera()`.

# Разработка приложения для камеры

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Создавая приложение, которое подключается к камере пользователя, код следует писать с учетом следующего.

- Проверьте, установлена ли камера на компьютере пользователя в данный момент. Обработайте ситуацию, когда камера не доступна.
- Только для проигрывателя Flash Player: проверьте, разрешил ли пользователь доступ к камере. По соображениям безопасности, проигрыватель открывает диалоговое окно «Параметры Flash Player», в котором пользователь может предоставить или запретить доступ к своей камере. В результате этого проигрыватель Flash Player не может подключиться к камере пользователя и передавать видеопоток, если не получит явного разрешения. Если пользователь выберет «Разрешить», приложение сможет подключиться к его камере. Если пользователь выберет «Запретить», приложение не сможет получить доступ к камере. В обоих случаях приложения должны вежливо реагировать на ответ пользователя.
- В среде AIR проверьте, поддерживается ли класс Camera для профилей устройств, поддерживаемых вашим приложением.
- Класс Camera не поддерживается в мобильных браузерах.
- Класс Camera не поддерживается в приложениях AIR для мобильных устройств, которые используют режим визуализации с помощью графического процессора.
- На мобильных устройствах одновременно активной может быть только одна камера.

### Дополнительные разделы справки

Кристоф Конрац (Christophe Coenraets): Многопользовательское видео

Марк Доэрти (Mark Doherty): Приложения Android Radar (источник)

# Подключение к камере пользователя

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы подключиться к камере пользователя, во-первых, нужно добавить новый экземпляр Camera, создав переменную типа Camera и присвоив ей значение, возвращенное методом `Camera.getCamera()`.

Во-вторых, необходимо создать новый объект Video и присоединить к нему объект Camera.

В-третьих, нужно добавить объект Video в список отображения. Выполнение действий 2 и 3 требуется по той причине, что класс Camera не расширяет класс DisplayObject и его объекты нельзя добавить в список отображения напрямую. Для отображения видео, захватываемого камерой, нужно создать новый объект Video, и вызвать метод `attachCamera()`.

Следующий код демонстрирует выполнение этих трех действий.

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Обратите внимание, что если к компьютеру пользователя не подключена камера, приложение ничего не отображает.

В реальной ситуации приложение должно выполнять дополнительные действия. Дополнительные сведения см. в разделах ««Проверка установки камер» на странице 560» и ««Определение разрешений для доступа к камере» на странице 561».

### Дополнительные разделы справки

Ли Бримелоу (Lee Brimelow): Как получить доступ к камере на устройствах Android

# Проверка установки камер

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Прежде чем пытаться использовать методы или свойства экземпляра Camera, необходимо убедиться, что к компьютеру пользователя подключена камера. Установку камеры можно проверить двумя способами.

- Проверите статическое свойство `Camera.names`, которое содержит массив с именами доступных камер. Как правило, этот массив содержит одну или меньше строк, так как большинство пользователей одновременно устанавливают не больше одной камеры. Следующий код проверяет свойство `Camera.names`, чтобы узнать, есть ли у пользователя доступные камеры.

  ```
  if (Camera.names.length > 0)
  {
      trace("User has at least one camera installed.");
      var cam:Camera = Camera.getCamera(); // Get default camera.
  }
  else
  {
      trace("User has no cameras installed.");
  }
  ```

- Проверьте значение, возвращенное статическим методом `Camera.getCamera()`. Если камеры недоступны или не установлены, этот метод возвращает значение `null`, в противном случае он возвращает ссылку на объект Camera. Следующий код использует метод `Camera.getCamera()`, чтобы узнать, есть ли у пользователя доступные камеры.

```
var cam:Camera = Camera.getCamera();
if (cam == null)
{
    trace("User has no cameras installed.");
}
else
{
    trace("User has at least 1 camera installed.");
}
```

Так как класс Camera не расширяет класс DisplayObject, его экземпляр нельзя добавить в список отображения напрямую с помощью метода `addChild()`. Для отображения видео, захватываемого с камеры, необходимо создать новый объект Video и вызвать для него метод `attachCamera()`.

Следующий код присоединяет камеру, если она есть; в противном случае приложение ничего не показывает.

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
```

**Камеры мобильных устройств**

Класс Camera не поддерживается в среде выполнения Flash Player в браузерах для мобильных устройств.

В приложениях AIR для мобильных устройств можно получать доступ к одной или нескольким камерам на устройстве. На мобильных устройствах можно использовать камеры и спереди и сзади, но одновременно может отображаться вывод только одной из них. (В результате присоединения второй камеры первая отсоединяется.) Камера на передней панели устройства зеркально отображается по горизонтали на устройствах iOS, на устройствах Android этого не происходит.

# Определение разрешений для доступа к камере

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В собственной изолированной программной среде приложение AIR может получать доступ к любой камере без разрешения пользователя. Однако на платформе Android для программы должно быть указано разрешение Android CAMERA в дескрипторе программы.

Прежде чем проигрыватель Flash Player сможет показать вывод с камеры, пользователь должен дать ему явное разрешение на доступ к камере. Когда вызывается метод `attachCamera()`, проигрыватель Flash Player выводит окно «Параметры Flash Player», в котором пользователь может разрешить или запретить проигрывателю Flash Player доступ к камере и микрофону. Если пользователь нажимает кнопку «Разрешить», Flash Player отображает вывод камеры в экземпляре Video в рабочей области. Если пользователь нажимает кнопку «Запретить», Flash Player не может подключиться к камере, и объект Video ничего не показывает.

Если требуется узнать, предоставил ли пользователь доступ к камере проигрывателю Flash Player, можно прослушивать событие `status` камеры (`StatusEvent.STATUS`), как показано в следующем коде.

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
function statusHandler(event:StatusEvent):void
{
    // This event gets dispatched when the user clicks the "Allow" or "Deny"
    // button in the Flash Player Settings dialog box.
    trace(event.code); // "Camera.Muted" or "Camera.Unmuted"
}
```

Функция `statusHandler()` вызывается сразу после того, как пользователь нажимает кнопку «Разрешить» или «Запретить». Определить, какую кнопку нажал пользователь, можно двумя способами:

- Параметр `event` функции `statusHandler()` содержит свойство code со строкой «Camera.Muted» или «Camera.Unmuted». Если свойство имеет значение «Camera.Muted», пользователь нажал кнопку «Запретить» и проигрыватель Flash Player не может получить доступ к камере. Это демонстрируется на следующем примере.

```
function statusHandler(event:StatusEvent):void
{
    switch (event.code)
    {
        case "Camera.Muted":
            trace("User clicked Deny.");
            break;
        case "Camera.Unmuted":
            trace("User clicked Accept.");
            break;
    }
}
```

- Класс Camera содержит свойство `muted`, которое предназначено только для чтения и указывает, запретил пользователь доступ (`true`) или разрешил (`false`) через панель «Конфиденциальность Flash Player». Это демонстрируется на следующем примере.

```
function statusHandler(event:StatusEvent):void
{
    if (cam.muted)
    {
        trace("User clicked Deny.");
    }
    else
    {
        trace("User clicked Accept.");
    }
}
```

Отслеживая отправку события status, можно добавить код для обработки ответа пользователя на запрос доступа к камере, чтобы выполнить соответствующие действия. Например, если пользователь нажимает кнопку «Запретить», можно показать ему сообщение о том, что для участия в видеочате необходимо нажать кнопку «Разрешить», или удалить объект Video из списка отображения, чтобы освободить системные ресурсы.

В среде AIR объект Camera не отправляет события status, так как разрешение на использование камеры не является динамическим.

# Обеспечение максимального качества видео с камеры

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

По умолчанию новые экземпляры класса Video имеют ширину 320 пикселов и высоту 240 пикселов. Чтобы обеспечить максимальное качество видео, необходимо всегда следить за тем, чтобы объект Video соответствовал размерам видео, возвращаемого объектом Camera. Ширину и высоту объекта Camera можно получить с помощью его свойств `width` и `height`, после чего нужно задать свойства `width` и `height` объекта Video в соответствии с полученными значениями, либо передать ширину и высоту камеры конструктору класса Video, как показано в следующем примере.

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video(cam.width, cam.height);
    vid.attachCamera(cam);
    addChild(vid);
}
```

Так как метод `getCamera()` возвращает ссылку на объект Camera (или значение `null`, если камера недоступна), то его методы и свойства можно получить, даже если пользователь запрещает доступ к камере. Это позволяет задать размер экземпляра Video в соответствии с высотой и шириной камеры.

```
var vid:Video;
var cam:Camera = Camera.getCamera();

if (cam == null)
{
    trace("Unable to locate available cameras.");
}
else
{
    trace("Found camera: " + cam.name);
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    vid = new Video();
    vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
    if (cam.muted)
    {
        trace("Unable to connect to active camera.");
    }
    else
    {
        // Resize Video object to match camera settings and
        // add the video to the display list.
        vid.width = cam.width;
        vid.height = cam.height;
        addChild(vid);
    }
    // Remove the status event listener.
    cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}
```

Сведения о полноэкранном режиме см. в пункте «Работа в полноэкранном режиме» раздела ««Установка свойств рабочей области» на странице 177».

# Мониторинг состояния камеры

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс Camera содержит несколько свойств, которые позволяют отслеживать текущее состояние объекта Camera. Например, следующий код отображает некоторые свойства экземпляра Camera с помощью объекта Timer и экземпляра TextField в списке отображения.

```
var vid:Video;
var cam:Camera = Camera.getCamera();
var tf:TextField = new TextField();
tf.x = 300;
tf.autoSize = TextFieldAutoSize.LEFT;
addChild(tf);

if (cam != null)
{
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    vid = new Video();
    vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
    if (!cam.muted)
    {
        vid.width = cam.width;
        vid.height = cam.height;
        addChild(vid);
        t.start();
    }
    cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}

var t:Timer = new Timer(100);
t.addEventListener(TimerEvent.TIMER, timerHandler);
function timerHandler(event:TimerEvent):void
{
    tf.text = "";
    tf.appendText("activityLevel: " + cam.activityLevel + "\n");
    tf.appendText("bandwidth: " + cam.bandwidth + "\n");
    tf.appendText("currentFPS: " + cam.currentFPS + "\n");
    tf.appendText("fps: " + cam.fps + "\n");
    tf.appendText("keyFrameInterval: " + cam.keyFrameInterval + "\n");
    tf.appendText("loopback: " + cam.loopback + "\n");
    tf.appendText("motionLevel: " + cam.motionLevel + "\n");
    tf.appendText("motionTimeout: " + cam.motionTimeout + "\n");
    tf.appendText("quality: " + cam.quality + "\n");
}
```

Через каждую десятую долю секунды (100 миллисекунд) объект Timer отправляет событие`timer`, и функция `timerHandler()` обновляет текстовое поле в списке отображения.

# Глава 27. Цифровые средства защиты авторских прав (DRM)

**Flash Player 10.1 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Программный пакет Adobe® Access™ является решением по защите содержимого. Оно позволяет владельцам содержимого, распространителям и рекламодателям получить новые источники дохода за счет предоставления прямого доступа к платному содержимому. Adobe Access может использоваться издателями для шифрования содержимого, создания политик и выдачи лицензий. Adobe Flash Player и Adobe AIR включают в себя библиотеку DRM — модуль Adobe Access. Этот модуль позволяет среде выполнения обмениваться данными с сервером лицензий Adobe Access и воспроизводить защищенное содержимое. Таким образом, среда выполнения завершает жизненный цикл содержимого, защищенного модулем Adobe Access и распространенного сервером Flash Media Server.

С помощью модуля Adobe Access поставщики содержимого могут предоставлять как бесплатное, так и платное содержимое. Например, клиент хочет смотреть телевизионную программу без рекламы. Клиент регистрируется и платит издателю содержимого. После этого клиент может использовать учетные данные для получения доступа к программе и воспроизведения программы без рекламы.

В другом примере клиент хочет просматривать содержимое в автономном режиме в поездке при отсутствии доступа к Интернету. Такой автономный технологический процесс поддерживается в приложениях AIR. После регистрации и внесения платы издателю за содержимое пользователь может получать доступ к содержимому и связанному приложению AIR и загружать их с веб-сайта издателя. С помощью приложения AIR пользователь может просматривать содержимое в автономном режиме в течение разрешенного периода времени. Содержимое также можно предоставлять для совместного использования другим устройствам в одной группе устройств, использующей домены (**новая возможность в 3.0**).

Модуль Adobe Access также поддерживает анонимный доступ, при котором не требуется проверка подлинности пользователя. Например, издатель может воспользоваться анонимным доступом для распространения рекламных материалов. Анонимный доступ также обеспечивает свободный доступ к текущему содержимому в течении определенного периода. Поставщик содержимого также может указать тип и версию проигрывателя, необходимого для воспроизведения содержимого.

Когда пользователь пытается воспроизвести защищенное содержимое в проигрывателе Flash Player или среде Adobe AIR, приложение должно вызывать API-интерфейсы DRM. API-интерфейсы DRM запускают технологический процесс для воспроизведения защищенного содержимого. Среда выполнения через модуль Adobe Access связывается с сервером лицензий. Сервер лицензий при необходимости проверяет подлинность пользователя и выдает лицензию для разрешения воспроизведения защищенного содержимого. Среда выполнения получает лицензию и расшифровывает содержимое для воспроизведения.

Ниже описано, как разрешить приложению проигрывать содержимое, защищенное технологией Adobe Access. Необязательно знать принципы шифрования содержимого или обслуживания политик в Adobe Access. Однако предполагается, что для проверки подлинности пользователя и получения лицензии осуществляется связь с сервером лицензий Adobe Access. Также предполагается, что разрабатывается приложение, специально предназначенное для воспроизведения содержимого, защищенного Adobe Access.

Общую информацию о пакете Adobe Access, в том числе принципы создания политик, см. в документации, включенной в комплект поставки Adobe Access.

### Дополнительные разделы справки

Пакет flash.net.drm

flash.net.NetConnection

flash.net.NetStream

# Описание технологического процесса защищенного содержимого

**Flash Player 10.1 и более поздних версий, Adobe AIR 2.0 и более поздних версий**

***Важная информация***. В состав Flash Player 11.5 и более поздних версий уже включен модуль Adobe Access, поэтому обновление (посредством метода `SystemUpdater.update(SystemUpdaterType.DRM)`) не требуется. Это относится к следующим браузерам и платформам:

- Компонент Flash Player 11.5 ActiveX — для всех платформ, за исключением Internet Explorer на Windows 8 на компьютерах с процессорами Intel
- Плагин Flash Player 11.5 — для всех браузеров;
- Adobe AIR (для настольных и мобильных систем).

Это означает, что обновление *все же требуется* в следующих случаях:

- Internet Explorer на Windows 8; на компьютерах с процессорами Intel
- Flash Player 11.4 и более старых версий, за исключением браузера Google Chrome версии 22 и более поздней (для всех платформ) или 21 и более поздней (для Windows).

***Примечание.*** *Вызов метода `SystemUpdater.update(SystemUpdaterType.DRM)` в системах с Flash Player 11.5 и более поздних версий не приведет к ошибкам, и при этом ничего не будет загружено.*

Следующий рабочий процесс высокого уровня демонстрирует, как приложение может получать и воспроизводить защищенное содержимое. В этом рабочем процессе подразумевается, что приложение создано специально для воспроизведения содержимого, защищенного Adobe Access:

1. Получите метаданные содержимого.
2. При необходимости обработайте обновления в проигрывателе Flash Player.
3. Проверьте, доступна ли лицензия локально. Если доступна, загрузите ее и перейдите к действию 7. Если нет, перейдите к действию 4.
4. Проверьте, требуется ли проверка подлинности. Если нет, перейдите к действию 7.
5. Если проверка подлинности требуется, получите учетные данные для проверки подлинности у пользователя и передайте их на сервер лицензий.
6. Если требуется регистрация домена, выполните ее (в AIR 3.0 и более поздних версиях).
7. После выполнения проверки подлинности загрузите лицензию с сервера.
8. Воспроизведите содержимое.

Если операция выполнена без ошибок и пользователь авторизован для просмотра содержимого, объект NetStream отправляет объект DRMStatusEvent. В программе начинается воспроизведение. Объект DRMStatusEvent содержит необходимую информацию о ваучере, которая определяет политику и разрешения для пользователя. Например, в нем хранится информация о том, может ли содержимое быть доступно в автономном режиме, или информация об окончании срока действия лицензии. Эти данные могут использоваться приложением для сообщения пользователю статуса его разрешения. Например, в приложении может отображаться число дней, оставшихся до конца периода просмотра.

Если пользователю разрешен доступ в автономном режиме, ваучер кэшируется и зашифрованное содержимое загружается на компьютер пользователя. Содержимое доступно в течение срока, определяемого периодом кэширования лицензии. Свойство события `detail` содержит запись `"DRM.voucherObtained"`. Приложение определяет, где хранить содержимое в системе пользователя, чтобы оно было доступно в автономном режиме. Можно также предварительно загрузить ваучеры с помощью класса DRMManager.

***Примечание.*** *Кэширование и предварительная загрузка ваучеров поддерживается как в AIR, так и в Flash Player. Однако загрузка и хранение зашифрованного содержимого поддерживается только в AIR.*

Ошибки обрабатывает исключительно приложение. Это касается и случаев, когда пользовать вводит верные учетные данные, но ваучер, защищающий зашифрованное содержимое, запрещает доступ к содержимому. Например, пользователь, прошедший проверку подлинности, не может получать доступ к содержимому, если не оплачены права доступа. Такая ситуация может настать, если два зарегистрированных участника пытаются обратиться к содержимому издателя, за которое заплатил только один из них. Приложение должно сообщить пользователю об ошибке и предложить варианты действий. Типичным предлагаемым вариантом являются инструкции по регистрации и оплате просмотра.

## Подробный технологический процесс API-интерфейса

**Flash Player 10.1 и более поздних версий, AIR 2.0 и более поздних версий**

С помощью этой последовательности действий приводится более подробное описание технологического процесса защищенного содержимого. В этом рабочем процессе описаны API-интерфейсы, используемые специально для воспроизведения содержимого, защищенного Adobe Access.

1. С использованием объекта URLLoader загрузите байты файла метаданных защищенного содержимого. Задайте этот объект для переменной, например `metadata_bytes`.

   Для всего содержимого, управляемого Adobe Access, предусмотрены метаданные Adobe Access. При создании пакета содержимого эти метаданные можно сохранить в качестве отдельного файла метаданных (.metadata) наряду с содержимым. Дополнительные сведения см. в документации по Adobe Access.

2. Создайте экземпляр DRMContentData. Поместите этот код в блок try-catch:

   ```
   new DRMContentData(metadata_bytes)
   ```

   где *`metadata_bytes`* — объект URLLoader, полученный в действии 1.

3. (только для проигрывателя Flash Player) Среда выполнения проверяет наличие модуля Adobe Access. Если он не обнаружен, выдается исключение IllegalOperationError, содержащее ошибку DRMErrorEvent с кодом 3344 или DRMErrorEvent с кодом 3343.

   Для обработки этой ошибки загрузите модуль Adobe Access с помощью API-интерфейса SystemUpdater. По завершении загрузки модуля объект SystemUpdater отправляет событие COMPLETE. Включите прослушиватель для этого события, который выполняет возврат к действию 2 при отправке этого события. Эти действия иллюстрирует следующий код:

```
flash.system.SystemUpdater.addEventListener(Event.COMPLETE, updateCompleteHandler);
flash.system.SystemUpdater.update(flash.system.SystemUpdaterType.DRM)
```

```
private function updateCompleteHandler (event:Event):void {
    /*redo step 2*/
    drmContentData = new DRMContentData(metadata_bytes);
}
```

Если необходимо обновить сам проигрыватель, отправляется событие состояния. Дополнительные сведения об обработке этого события см. в разделе ««Прослушивание события обновления» на странице 584».

***Примечание.*** *В приложениях AIR установщик AIR обрабатывает обновление модуля Adobe Access и требуемых обновлений среды выполнения.*

4 Создайте прослушиватели для прослушивания событий DRMStatusEvent и DRMErrorEvent, отправляемых из объекта DRMManager:

```
DRMManager.addEventListener(DRMStatusEvent.DRM_STATUS, onDRMStatus);
DRMManager.addEventListener(DRMErrorEvent.DRM_ERROR, onDRMError);
```

В прослушивателе DRMStatusEvent проверьте, что ваучер действителен (не является нулевым). В прослушивателе DRMErrorEvent обработайте события DRMErrorEvent. См. разделы ««Использование класса DRMStatusEvent» на странице 576» и ««Использование класса DRMErrorEvent» на странице 581».

5 Загрузите ваучер (лицензию), необходимый для воспроизведения содержимого.

Сначала попробуйте загрузить локально сохраненную лицензию для воспроизведения содержимого:

```
DRMManager.loadvoucher(drmContentData, LoadVoucherSetting.LOCAL_ONLY)
```

По завершении загрузки объект DRMManager отправляет событие `DRMStatusEvent.DRM_Status`.

6 Если объект DRMVoucher не является нулевым, ваучер действителен. Перейдите к действию 13.

7 Если объект DRMVoucher является нулевым, проверьте метод проверки подлинности, требуемый политикой для этого содержимого. Используйте свойство `DRMContentData.authenticationMethod`.

8 Если методом проверки подлинности является `ANONYMOUS`, перейдите к действию 13.

9 Если методом проверки подлинности является `USERNAME_AND_PASSWORD`, приложение должно предоставлять механизм, позволяющий пользователю вводить учетные данные. Передайте эти учетные данные на сервер лицензий для проверки подлинности пользователя:

```
DRMManager.authenticate(metadata.serverURL, metadata.domain, username, password)
```

Объект DRMManager отправляет событие `DRMAuthenticationErrorEvent`, если проверка подлинности выполнена неудачно, или событие `DRMAuthenticationCompleteEvent`, если проверка подлинности выполнена удачно. Создайте прослушиватели для этих событий.

10 Если для метода проверки подлинности задано значение `UNKNOWN`, необходимо применить пользовательский метод. В данном случае поставщик содержимого предусмотрел выполнение внешней проверки подлинности без использования API-интерфейсов ActionScript 3.0. Пользовательская процедура проверки подлинности должна генерировать маркер авторизации, который можно передать методу `DRMManager.setAuthenticationToken()`.

11 Если происходит сбой проверки подлинности, приложение должно возвращаться к выполнению действия 9. Убедитесь, что в приложении предусмотрен механизм для обработки и ограничения повторяющихся сбоев проверки подлинности. Например, после трех попыток отображается сообщение о сбое проверки подлинности и невозможности воспроизведения содержимого.

12 Чтобы использовать сохраненную метку вместо приглашения к вводу учетных данных, задайте метку с помощью метода `DRMManager.setAuthenticationToken()`. После этого лицензия загружается с сервера лицензий и содержимое воспроизводится, как в действии 8.

13 (необязательно) Если проверка подлинности выполнена, можно получить метку проверки подлинности, которая представляет собой байтовый массив, кэшированный в памяти. Получите эту метку с помощью свойства `DRMAuthenticationCompleteEvent.token`. Можно сохранить и использовать метку проверки подлинности, чтобы пользователям не нужно было постоянно вводить учетные данные для этого содержимого. Сервер лицензий определяет период, в течение которого метка проверки подлинности действительна.

14 Если проверка подлинности выполнена загрузите лицензию с сервера лицензий:

```
DRMManager.loadvoucher(drmContentData, LoadVoucherSetting.FORCE_REFRESH)
```

По завершении загрузки объект DRMManager отправляет событие DRMStatusEvent.DRM_STATUS. Прослушивайте это событие, при его отправке можно воспроизвести содержимое.

15 Воспроизведите видео. Для этого создайте объект NetStream и вызовите метод `play()`:

```
stream = new NetStream(connection);
stream.addEventListener(DRMStatusEvent.DRM _STATUS, drmStatusHandler);
stream.addEventListener(DRMErrorEvent.DRM_ERROR, drmErrorHandler);
stream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
stream.client = new CustomClient();
video.attachNetStream(stream);
stream.play(videoURL);
```

## DRMContentData и объекты сеансов

Когда создается экземпляр `DRMContentData`, он используется как объект сеанса, ссылающийся на модуль Flash Player DRM. Все API-интерфейсы `DRMManager`, получаемые этим экземпляром `DRMContentData` будут использовать данный модуль DRM. Однако существует два API-интерфейса `DRMManager`, которые не используют экземпляр `DRMContentData`. Они перечислены ниже.

1 `authenticate()`

2 `setAuthenticationToken()`

Так как отсутствует связанный экземпляр `DRMContentData`, при вызове этих API-интерфейсов `DRMManager` будет использоваться последний модуль DRM с диска. Это может вызывать проблемы, если в середине рабочего процесса DRM приложения произойдет обновление модуля DRM. Рассмотрим следующую ситуацию.

1 Приложение создает объект `DRMContentData` с именем `contentData1`, который использует модуль DRM с именем *AdobeCP1*.

2 Приложение вызывает метод `DRMManager.authenticate(contentData1.serverURL,...)`.

3 Приложение вызывает метод `DRMManager.loadVoucher(contentData1, ...)`.

Если обновление модуля DRM произойдет, прежде чем приложение перейдет к шагу 2, то метод `DRMManager.authenticate()` будет выполнять проверку подлинности с помощью модуля DRM с именем *AdobeCP2*. Метод `loadVoucher()` в шаге 3 вернет ошибку, так как он будет по-прежнему использовать модуль DRM с именем *AdobeCP1*. Обновление может произойти, если другое приложение вызовет обновление модуля DRM. Чтобы этого не происходило, можно вызвать обновление модуля DRM при запуске приложения.

## События, связанные с DRM

Среда выполнения отправляет многочисленные события, когда в приложении предпринимается попытка воспроизвести защищенное содержимое:

- Событие DRMDeviceGroupErrorEvent (только для AIR), отправляемое DRMManager
- Событие DRMAuthenticateEvent (только для AIR), отправляемое объектом NetStream
- Событие DRMAuthenticationCompleteEvent, отправляемое объектом DRMManager
- Событие DRMAuthenticationErrorEvent, отправляемое объектом DRMManager
- Событие DRMErrorEvent, отправляемое объектами NetStream и DRMManager
- Событие DRMStatusEvent, отправляемое объектами NetStream и DRMManager
- Событие StatusEvent
- Событие NetStatusEvent. См. раздел ««Прослушивание события обновления» на странице 584».

Для поддержки содержимого, защищенного Adobe Access, добавьте прослушиватели событий для обработки событий DRM.

## Предварительная загрузка ваучеров для автономного воспроизведения

**Adobe AIR 1.5 и более поздних версий**

Можно предварительно загрузить ваучеры (лицензии), требуемые для воспроизведения содержимого, защищенного Adobe Access. Предварительно загруженные ваучеры дают пользователям возможность просматривать содержимое независимо от наличия доступа к Интернету. (Но для самого процесса предварительной загрузки требуется подключение к Интернету.) Для предварительной загрузки ваучеров можно использовать метод `preloadEmbeddedMetadata()` класса NetStream и класс DRMManager. В среде AIR 2.0 и более поздней версии для непосредственной предварительной загрузки ваучеров можно использовать объект DRMContentData. Этот метод является предпочтительным, поскольку он позволяет обновлять объект DRMContentData независимо от содержимого. (Метод `preloadEmbeddedData()` извлекает объект DRMContentData из содержимого.)

### Использование объекта DRMContentData

**Adobe AIR 2.0 и более поздних версий**

Следующие действия описывают рабочий процесс предварительной загрузки ваучера для защищенного файла мультимедиа с использованием объекта DRMContentData.

1. Получите бинарные метаданные для упакованного содержимого. При использовании базового средства упаковки Java Adobe Access этот файл метаданных автоматически создается с расширением *.metadata*. Эти метаданные можно загрузить, к примеру, с помощью класса URLLoader.
2. Создайте объект DRMContentData, передав метаданные функции конструктора:

```
var drmData:DRMContentData = new DRMContentData( metadata );
```

3. Остальные действия аналогичны действиям в технологическом процессе, описанном в разделе ««Описание технологического процесса защищенного содержимого» на странице 567».

## Использование метода preloadEmbeddedMetadata()

**Adobe AIR 1.5 и более поздних версий**

Следующие действия описывают рабочий процесс предварительной загрузки ваучера для DRM-защищенного файла мультимедиа с использованием метода `preloadEmbeddedMetadata()`.

1. Загрузите и сохраните мультимедийный файл. (Метаданные DRM могут быть предварительно загружены только из локально сохраненных файлов.)
2. Создайте объекты NetConnection и NetStream, обеспечив реализацию функций обратного вызова `onDRMContentData()` и `onPlayStatus()` в клиентском объекте NetStream.
3. Создайте объект NetStreamPlayOptions и установите в свойстве `stream` адрес URL для локального мультимедийного файла.
4. Вызовите метод `preloadEmbeddedMetadata()` класса NetStream, передав ему объект NetStreamPlayOptions, идентифицирующий мультимедийный файл для разбора.
5. Если мультимедийный файл содержит метаданные DRM, то вызывается функция `onDRMContentData()`. Метаданные передаются этой функции в виде объекта DRMContentData.
6. Используйте объект DRMContentData для получения ваучера с помощью метода `loadVoucher()` класса DRMManager.

   Если свойство `authenticationMethod` объекта `DRMContentData` имеет значение `flash.net.drm.AuthenticationMethod.USERNAME_AND_PASSWORD`, то перед загрузкой ваучера выполните проверку подлинности пользователя на сервере управления правами на мультимедийное содержимое. Свойства `serverURL` и `domain` объекта DRMContentData могут быть переданы методу `authenticate()` класса DRMManager вместе с учетными данными пользователя.
7. Функция обратного вызова `onPlayStatus()` вызывается, когда выполнен разбор файла. Если функция `onDRMContentData()` не была вызвана, то файл не содержит метаданных, требуемых для получения ваучера. То, что функция не была вызвана, также может означать, что этот файл не защищен Adobe Access.

Следующий пример кода для среды AIR показывает способ начальной загрузки ваучера для локального файла мультимедиа:

```
package
{
import flash.display.Sprite;
import flash.events.DRMAuthenticationCompleteEvent;
import flash.events.DRMAuthenticationErrorEvent;
import flash.events.DRMErrorEvent;
import flash.ev ents.DRMStatusEvent;
import flash.events.NetStatusEvent;
import flash.net.NetConnection;
import flash.net.NetStream;
import flash.net.NetStreamPlayOptions;
import flash.net.drm.AuthenticationMethod;
import flash.net.drm.DRMContentData;
import flash.net.drm.DRMManager;
import flash.net.drm.LoadVoucherSetting;
public class DRMPreloader extends Sprite
{
    private var videoURL:String = "app-storage:/video.flv";
    private var userName:String = "user";
    private var password:String = "password";
    private var preloadConnection:NetConnection;
```

```
    private var preloadStream:NetStream;
    private var drmManager:DRMManager = DRMManager.getDRMManager();
    private var drmContentData:DRMContentData;
    public function DRMPreloader():void {
        drmManager.addEventListener(
            DRMAuthenticationCompleteEvent.AUTHENTICATION_COMPLETE,
            onAuthenticationComplete);
        drmManager.addEventListener(DRMAuthenticationErrorEvent.AUTHENTICATION_ERROR,
            onAuthenticationError);
        drmManager.addEventListener(DRMStatusEvent.DRM_STATUS, onDRMStatus);
        drmManager.addEventListener(DRMErrorEvent.DRM_ERROR, onDRMError);
        preloadConnection = new NetConnection();
        preloadConnection.addEventListener(NetStatusEvent.NET_STATUS, onConnect);
        preloadConnection.connect(null);
    }

    private function onConnect( event:NetStatusEvent ):void
    {
        preloadMetadata();
    }
    private function preloadMetadata():void
    {
        preloadStream = new NetStream( preloadConnection );
        preloadStream.client = this;
        var options:NetStreamPlayOptions = new NetStreamPlayOptions();
        options.streamName = videoURL;
        preloadStream.preloadEmbeddedData( options );
    }
    public function onDRMContentData( drmMetadata:DRMContentData ):void
    {
        drmContentData = drmMetadata;
        if ( drmMetadata.authenticationMethod == AuthenticationMethod.USERNAME_AND_PASSWORD )
        {
            authenticateUser();
        }
        else
            {
                getVoucher();
            }
    }
    private function getVoucher():void
    {
        drmManager.loadVoucher( drmContentData, LoadVoucherSetting.ALLOW_SERVER );
    }

    private function authenticateUser():void
    {
        drmManager.authenticate( drmContentData.serverURL, drmContentData.domain, userName,
password );
    }
    private function onAuthenticationError( event:DRMAuthenticationErrorEvent ):void
    {
        trace( "Authentication error: " + event.errorID + ", " + event.subErrorID );
    }

    private function onAuthenticationComplete( event:DRMAuthenticationCompleteEvent ):void
    {
```

```
        trace( "Authenticated to: " + event.serverURL + ", domain: " + event.domain );
        getVoucher();
    }
    private function onDRMStatus( event:DRMStatusEvent ):void
    {
        trace( "DRM Status: " + event.detail);
        trace("--Voucher allows offline playback = " + event.isAvailableOffline );
        trace("--Voucher already cached          = " + event.isLocal );
        trace("--Voucher required authentication = " + !event.isAnonymous );
    }
    private function onDRMError( event:DRMErrorEvent ):void
    {
        trace( "DRM error event: " + event.errorID + ", " + event.subErrorID + ", " + event.text );
    }
    public function onPlayStatus( info:Object ):void
    {
        preloadStream.close();
    }
}
}
```

# Относящиеся к DRM элементы и события класса NetStream

**Flash Player 10.1 и более поздних версий, Adobe AIR 1.0**

Класс NetStream предоставляет одностороннее соединение для передачи потоков между проигрывателем Flash Player или приложением AIR и сервером Flash Media или локальной файловой системой. (Класс NetStream также поддерживает промежуточную загрузку.) Объект NetStream — это канал внутри объекта NetConnection. Класс NetStream отправляет четыре связанные с DRM события:

| Событие | Описание |
|---|---|
| drmAuthenticate<br>(только для AIR) | Задано в классе DRMAuthenticateEvent. Это событие отправляется, когда объект NetStream пытается воспроизвести защищенное содержимое, для которого пользователь должен ввести учетные данные для проверки подлинности перед воспроизведением.<br>Свойства этого события включают header, usernamePrompt, passwordPrompt и urlPrompt — их можно использовать для получения из задания учетных данных. Это событие повторяется, пока объект NetStream не получит действительные учетные данные. |

| Событие | Описание |
|---|---|
| drmError | Задано в классе DRMErrorEvent и отправляется, когда объект NetStream пытается воспроизвести защищенное содержимое и обнаруживает связанную с DRM ошибку. В частности, ошибка DRM объекта события отправляется при неуспешной авторизации пользователя. Эта ошибка может быть вызвана тем, что пользователь не приобрел права на просмотр содержимого. Другая возможная причина — поставщик содержимого не поддерживает программу просмотра. |
| drmStatus | Задано в классе DRMStatusEvent. Это событие отправляется, когда начинается воспроизведение защищенного содержимого (когда пользователь прошел проверку подлинности и получил разрешение на воспроизведение содержимого). Объект DRMStatusEvent содержит информацию, относящуюся к ваучеру. Ваучер содержит информацию о том, может ли содержимое просматриваться в автономном режиме, когда истекает срок действия ваучера и когда содержимое перестанет быть доступно для просмотра. |
| status | Задано в классе events.StatusEvent и отправляется, только когда приложение пытается воспроизвести защищенное содержимое путем вызова метода NetStream.play(). Код-значение свойства status равен DRM.encryptedFLV. |

Класс NetStream включает следующие относящиеся к DRM методы, которые используются только в среде AIR:

| Метод | Описание |
|---|---|
| resetDRMVouchers() | Удаляет все локально кэшируемые данные ваучера DRM. Приложение должно будет загрузить ваучер повторно, чтобы пользователь смог обращаться к зашифрованному содержимому.<br><br>Например, в следующем коде из кэша удаляются все ваучеры:<br><br>NetStream.resetDRMVouchers(); |
| setDRMAuthenticationCredentials() | Передает набор учетных данных для проверки подлинности, а именно имя пользователя, пароль и тип проверки подлинности, в объект NetStream для выполнения проверки подлинности. Действительными типами проверки подлинности являются `"drm"` и `"proxy"`. Тип проверки подлинности `"drm"` сопоставляет учетные данные с информацией на сервере Adobe Access. При использовании типа проверки подлинности `proxy` проверка подлинности учетных данных выполняется на прокси-сервере; они должны совпадать с данными, запрашиваемыми прокси-сервером. Например, если программа используется сотрудником организации, то она должна пройти проверку подлинности на прокси-сервере, прежде чем пользователю будет разрешен доступ в Интернет. Параметр proxy разрешает этот тип проверки подлинности. Если проверка подлинности не анонимная, то после проверки подлинности на прокси-сервере пользователю все равно нужно пройти проверку подлинности в среде Adobe Access, чтобы получить ваучер и воспроизвести содержимое. Можно во второй раз использовать метод `setDRMAuthenticationCredentials()` с параметром "`drm`" для выполнения проверки подлинности в среде Adobe Access. |
| preloadEmbeddedMetadata() | Выполняется разбор встроенных метаданных в локальном мультимедийном файле. Если обнаруживаются относящиеся к DRM метаданные, AIR вызывает функцию обратного вызова `onDRMContentData()`. |

Кроме того, в среде AIR объект NetStream вызывает функции обратного вызова `onDRMContentData()` и `onPlayStatus()` как результат обращения к методу `preloadEmbeddedMetaData()`. Функция `onDRMContentData()` вызывается, когда в мультимедийном файле обнаруживаются метаданные DRM. Функция `onPlayStatus()` вызывается по завершении синтаксического анализа файла. Функции `onDRMContentData()` и `onPlayStatus()` должны быть определены в объекте `client`, назначенном экземпляру класса NetStream. Если для предварительной загрузки ваучеров и воспроизведения содержимого используется один и тот же объект NetStream, дождитесь вызова `onPlayStatus()`, создаваемого методом `preloadEmbeddedMetaData()`, перед началом воспроизведения.

В приведенном ниже коде в среде AIR для проверки подлинности пользователя используются имя пользователя (administrator), пароль (password) и тип проверки подлинности drm. Метод setDRMAuthenticationCredentials() должен предоставлять учетные данные, которые совпадают с учетными данными, известными поставщику содержимого и принимаемыми им. Эти учетные данные являются теми же учетными данными, которые разрешают пользователю просмотр содержимого. Сюда не включен код для воспроизведения видео и проверки успешности соединения с потоком видео.

```
var connection:NetConnection = new NetConnection();
connection.connect(null);

var videoStream:NetStream = new NetStream(connection);

videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                             drmAuthenticateEventHandler)

private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
{
    videoStream.setDRMAuthenticationCredentials("administrator", "password", "drm");
}
```

# Использование класса DRMStatusEvent

**Flash Player 10.1, Adobe AIR 1.0 и более поздних версий**

Объект NetStream отправляет событие DRMStatusEvent, когда начинается успешное воспроизведение содержимого, защищенного Adobe Access. (Под успешным воспроизведением понимается, что лицензия проверена и пользователь прошел проверку подлинности и получил разрешение на просмотр содержимого). DRMStatusEvent также отправляется, когда доступ разрешается анонимным пользователям. Лицензия проверяется на предмет того, могут ли анонимные пользователи, для которых не требуется проверка подлинности, получать доступ для воспроизведения содержимого. Доступ анонимных пользователей может быть запрещен по разным причинам. Например, доступ анонимного пользователя к содержимому может быть запрещен после окончания срока действия лицензии.

Объект DRMStatusEvent содержит информацию, относящуюся к лицензии. Эта информация включает сведения о том, может ли лицензия предоставляться в автономном режиме, когда истекает срок действия ваучера и когда содержимое перестанет быть доступным для просмотра. Приложение может использовать эти данные для сообщения пользователю его статуса и разрешений.

## Свойства DRMStatusEvent

**Flash Player 10.1, Adobe AIR 1.0 и более поздних версий**

Класс DRMStatusEvent содержит ряд свойств. Некоторые свойства стали доступны в версиях AIR, предшествующих версии 1.0. Для получения полных сведений о версии см. *справочник ActionScript® 3.0 для Adobe® Flash® Professional CS5*.

Для свойств, которые не поддерживаются в проигрывателе Flash Player 10.1, класс DRMVoucher предоставляет равнозначные свойства для проигрывателя Flash Player.

| Свойство | Описание |
|---|---|
| contentData | Объект DRMContentData содержит метаданные DRM, встроенные в содержимое. |
| detail (только для AIR) | Строка, описывающая контекст события status. В DRM 1.0 единственным действительным значением является DRM.voucherObtained. |
| isAnonymous (только для AIR) | Указывает, доступно ли содержимое, защищенное Adobe Access, пользователю без ввода учетных данных для проверки подлинности (true) или нет (false). Значение false указывает, что пользователь должен ввести имя пользователя и пароль, совпадающие с теми, которые известны поставщику содержимого и ожидаются им. |
| isAvailableOffline (только для AIR) | Указывает, может ли содержимое, защищенное Adobe Access, просматриваться в автономном режиме. Если может, значение равно true, если нет — false. Чтобы защищенное содержимое можно было воспроизводить в автономном режиме, ваучер необходимо сохранить в кэш компьютера. |
| isLocal | Указывает, будет ли ваучер, необходимый для воспроизведения содержимого, кэшироваться локально. |
| offlineLeasePeriod (только для AIR) | Количество дней, оставшихся до истечения срока просмотра в автономном режиме. |
| policies (только для AIR) | Заказной объект, который может содержать особые свойства DRM. |
| voucher | Ваучер DRM. |
| voucherEndDate (только для AIR) | Дата истечения срока действия ваучера, после которой содержимое не доступно для просмотра. |

## Создание обработчика DRMStatusEvent

**Flash Player 10.1, Adobe AIR 1.0 и более поздних версий**

В следующем примере показано, как создавать обработчик события, выводящий информацию о статусе DRM для объекта NetStream, вызывавшем событие. Добавьте этот обработчик событий в объект NetStream, указывающий на защищенное содержимое.

```
function drmStatusEventHandler(event:DRMStatusEvent):void
{
    trace(event);
}
function drmStatusEventHandler(event:DRMStatusEvent):void
{
    trace(event);
}
```

# Использование класса DRMAuthenticateEvent

**Adobe AIR 1.0 и более поздних версий**

Объект DRMAuthenticateEvent отправляется, когда объект NetStream пытается воспроизвести защищенное содержимое, для которого требуются учетные данные для проверки подлинности перед воспроизведением.

Обработчик DRMAuthenticateEvent отвечает за сбор нужных учетных данных (имя пользователя, пароль, тип) и передачу значений методу `NetStream.setDRMAuthenticationCredentials()` для валидации. Любое приложение AIR должно содержать некий механизм для получения учетных данных пользователя. Например, в приложении пользователю может быть доступен простой пользовательский интерфейс для ввода значений имени пользователя и пароля. Также обеспечьте механизм обработки и ограничения повторяющихся попыток проверки подлинности.

## Свойства DRMAuthenticateEvent

**Adobe AIR 1.0 и более поздних версий**

Класс DRMAuthenticateEvent содержит ряд свойств:

| Свойство | Описание |
|---|---|
| authenticationType | Указывает, с чем должны сравниваться предоставленные учетные данные при проверки подлинности: данные Adobe Access (drm) или прокси-сервера (proxy). Например, параметр proxy позволяет приложению выполнять проверку подлинности на прокси-сервере, если она требуется перед предоставлением пользователю доступа в Интернет. Если проверка подлинности не анонимная, то после проверки подлинности на прокси-сервере пользователю все равно нужно пройти проверку подлинности в среде Adobe Access, чтобы получить ваучер и воспроизвести содержимое. Можно во второй раз использовать метод setDRMAuthenticationCredentials() с параметром "drm" для выполнения проверки подлинности в среде Adobe Access. |
| header | Файл с зашифрованным содержимым, предоставляемый сервером. В нем содержится информация о контексте зашифрованного содержимого.<br><br>Эта строка заголовка может передаваться приложению Flash, чтобы оно могла создать диалоговое окно для ввода имени пользователя и пароля. Строку заголовка можно использовать в качестве инструкций для диалогового окна. Например, в качестве заголовка можно использовать строку «Введите свои имя пользователя и пароль». |
| netstream | Объект NetStream, являющийся инициатором события. |
| passwordPrompt | Запрос пароля, инициируемый сервером. В строке может содержаться указание о типе требуемого пароля. |
| urlPrompt | Запрос URL, инициируемый сервером. Строка может содержать указание, куда отправляются имя пользователя и пароль. |
| usernamePrompt | Запрос имени пользователя, инициируемый сервером. В строке может содержаться указание на тип требуемого имени пользователя. Например, поставщик содержимого может потребовать в качестве имени пользователя адрес электронной почты. |

Приведенные выше строки предоставляются только сервером FMRMS. Сервер Adobe Access Server их не использует.

## Создание обработчика DRMAuthenticateEvent

**Adobe AIR 1.0 и более поздних версий**

В примере ниже создается обработчик события, который передает набор жестко заданных учетных данных объекту NetStream, являющемуся инициатором события. (Сюда не включен код для воспроизведения видео и проверки успешности соединения с потоком видео.)

```
var connection:NetConnection = new NetConnection();
connection.connect(null);

var videoStream:NetStream = new NetStream(connection);

videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                             drmAuthenticateEventHandler)

private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
{
    videoStream.setDRMAuthenticationCredentials("administrator", "password", "drm");
}
```

## Создание интерфейса для получения учетных данных

**Adobe AIR 1.0 и более поздних версий**

В случае, когда для доступа к защищенному содержимому требуется проверка подлинности, приложение AIR обычно получает учетные данные для проверки подлинности через пользовательский интерфейс.

Ниже приведен пример простого пользовательского интерфейса запроса учетных данных для Flex. Он состоит из объекта panel с двумя объектами TextInput: один для имени пользователя и один для пароля. На панели также есть кнопка, вызывающая метод `credentials()`.

```
<mx:Panel x="236.5" y="113" width="325" height="204" layout="absolute" title="Login">
    <mx:TextInput x="110" y="46" id="uName"/>
    <mx:TextInput x="110" y="76" id="pWord" displayAsPassword="true"/>
    <mx:Text x="35" y="48" text="Username:"/>
    <mx:Text x="35" y="78" text="Password:"/>
    <mx:Button x="120" y="115" label="Login" click="credentials()"/>
</mx:Panel>
```

Метод `credentials()` — это определяемый пользователем метод, передающий значения имени пользователя и пароля методу `setDRMAuthenticationCredentials()`. После передачи значений метод `credentials()` сбрасывает значения в объектах TextInput.

```
<mx:Script>
    <![CDATA[
        public function credentials():void
        {
            videoStream.setDRMAuthenticationCredentials(uName, pWord, "drm");
            uName.text = "";
            pWord.text = "";
        }
    ]]>
</mx:Script>
```

Для реализации этого простого интерфейса можно сделать панель частью нового состояния. Новое состояние создается на основе исходного с помощью объекта DRMAuthenticateEvent. В примере ниже содержится объект VideoDisplay с атрибутом source, который указывает на защищенный FLV-файл. В данном случае метод `credentials()` изменен таким образом, чтобы возвращать программу в исходное состояние. Это происходит после передачи учетных данных пользователя и сброса значений объекта TextInput.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    width="800"
    height="500"
    title="DRM FLV Player"
    creationComplete="initApp()" >

    <mx:states>
        <mx:State name="LOGIN">
            <mx:AddChild position="lastChild">
                    <mx:Panel x="236.5" y="113" width="325" height="204" layout="absolute"
                            title="Login">
                    <mx:TextInput x="110" y="46" id="uName"/>
                    <mx:TextInput x="110" y="76" id="pWord" displayAsPassword="true"/>
                    <mx:Text x="35" y="48" text="Username:"/>
                    <mx:Text x="35" y="78" text="Password:"/>
                    <mx:Button x="120" y="115" label="Login" click="credentials()"/>
                    <mx:Button x="193" y="115" label="Reset" click="uName.text='';
                            pWord.text='';"/>
                </mx:Panel>
            </mx:AddChild>
        </mx:State>
    </mx:states>

    <mx:Script>
        <![CDATA[
                import flash.events.DRMAuthenticateEvent;
            private function initApp():void
            {
                videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                                        drmAuthenticateEventHandler);
            }

            public function credentials():void
            {
                videoStream.setDRMAuthenticationCredentials(uName, pWord, "drm");
                uName.text = "";
                pWord.text = "";
                currentState='';
            }

            private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
            {
                currentState='LOGIN';
            }
        ]]>
    </mx:Script>

    <mx:VideoDisplay id="video" x="50" y="25" width="700" height="350"
        autoPlay="true"
        bufferTime="10.0"
        source="http://www.example.com/flv/Video.flv" />
</mx:WindowedApplication>
```

# Использование класса DRMErrorEvent

**Flash Player 10.1 и более поздних версий, Adobe AIR 1.0**

Adobe Flash Player и Adobe AIR отправляют объект DRMErrorEvent, когда объект NetStream, пытающийся воспроизвести защищенное содержимое, обнаруживает относящуюся к DRM ошибку. Если учетные данные пользователя недопустимы в приложении AIR, объект DRMAuthenticateEvent отправляется многократно, пока пользователь не введет действительные учетные данные или приложение не запретит дальнейшие попытки. Приложение должно прослушивать все события ошибок DRM, чтобы обнаружить, определить и обработать их.

Даже если учетные данные пользователя верны, условия ваучера содержимого могут запрещать просмотр зашифрованного содержимого. Например, доступ может быть запрещен, если пользователь пытается просмотреть содержимое в неавторизованной программе. Неавторизованной считается программа, которая не была авторизована издателем зашифрованного содержимого. В этом случае отправляется объект DRMErrorEvent.

События ошибок также могут отправляться, если содержимое повреждено или если версия приложения не соответствует требованиям ваучера. В приложении должен быть работающий механизм для обработки ошибок.

## Свойства DRMErrorEvent

**Flash Player 10.1 и более поздних версий, Adobe AIR 1.0**

Полный список ошибок см. в разделе «Коды ошибок среды выполнения» в справочнике ActionScript® 3.0 для Adobe® Flash® Professional CS5. Относящиеся к DRM ошибки начинаются с кода 3300.

## Создание обработчика DRMErrorEvent

**Flash Player 10.1 и более поздних версий, Adobe AIR 1.0**

В примере ниже показано, как создать обработчик событий для объекта NetStream, являющегося инициатором события. Он вызывается, когда объект NetStream обнаруживает ошибку при попытке воспроизведения защищенного содержимого. Как правило, при возникновении ошибки программа выполняет задачи по очистке. Затем она сообщает об ошибке пользователю и предлагает варианты решения проблемы.

```
private function drmErrorEventHandler(event:DRMErrorEvent):void
{
    trace(event.toString());
}
```

# Использование класса DRMManager

**Flash Player 10.1 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Используйте класс DRMManager для управления ваучерами и сеансами связи с сервером регистрации авторских прав на мультимедийные ресурсы в приложениях.

**Управление ваучерами (только для AIR)**

Когда пользователь воспроизводит защищенное содержимое, среда выполнения получает и кэширует лицензию, необходимую для просмотра содержимого. Если программа сохраняет файл на компьютере и лицензией предусмотрена возможность автономного воспроизведения, пользователь может открывать содержимое в программе AIR. Автономное воспроизведение возможно, даже если отсутствует подключение к серверу регистрации авторских прав на мультимедийные ресурсы. С помощью метода `preloadEmbeddedMetadata()` объектов DRMManager и NetStream можно предварительно кэшировать ваучер. Приложению не потребуется получать лицензию, необходимую для просмотра содержимого. Например, приложение может загрузить мультимедийный файл, а затем получить ваучер на него, пока пользователь подключен к Интернету.

Для предварительной загрузки файла используйте метод `preloadEmbeddedMetadata()` класса NetStream, чтобы получить объект DRMContentData. Объект DRMContentData содержит адрес URL и домен сервера регистрации авторских прав на мультимедийные ресурсы, который может предоставить такую лицензию, а также указывает, требуется ли проверка подлинности пользователя. С помощью этой информации можно вызвать метод `loadVoucher()` класса DRMManager, чтобы получить и кэшировать ваучер. Рабочий процесс предварительной загрузки ваучеров подробно описан в разделе «Предварительная загрузка ваучеров для автономного воспроизведения» на странице 571.

**Управление сеансами**

Класс DRMManager также можно использовать для проверки подлинности пользователя при входе на сервер регистрации авторских прав на мультимедийные ресурсы и для управления длительными сеансами.

Вызовите метод `authenticate()` класса DRMManager, чтобы установить сеанс связи с сервером регистрации авторских прав на мультимедийные ресурсы. Если проверка подлинности прошла успешно, класс DRMManager отправляет объект события DRMAuthenticationComplete. Этот объект содержит маркер сеанса. Данный маркер можно сохранить для создания сеансов в будущем, что избавит пользователя от необходимости предоставлять свои учетные данные. Передайте этот маркер методу `setAuthenticationToken()`, чтобы установить новый сеанс с проверкой подлинности. (Срок действия и другие атрибуты маркера определяются настройками сервера, создавшего этот маркер. Код программы AIR не должен интерпретировать структуру данных маркера, так как это может повлиять на последующие обновления AIR.)

Маркеры проверки подлинности можно передавать на другие компьютеры. Чтобы защитить маркеры, их можно хранить в зашифрованном локальном хранилище AIR. Дополнительные сведения см. в разделе ««Зашифрованная локальная система хранения данных» на странице 764».

## События DRMStatus

**Flash Player 10.1 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Класс DRMManager отправляет объект DRMStatusEvent после того, как успешно завершается вызов метода `loadVoucher()`.

Если ваучер получен, то свойство `detail` (только для AIR) объекта события принимает значение DRM.voucherObtained, а свойство `voucher` содержит объект DRMVoucher.

Если ваучер не получен, то свойство `detail` (только для AIR) по-прежнему принимает значение DRM.voucherObtained, однако значением свойства `voucher` является `null`. Ваучер не может быть получен в том случае, если, например, для LoadVoucherSetting используется значение *localOnly* и локально кэшированный ваучер отсутствует.

Если вызов метода `loadVoucher()` завершается неудачно (возможно, причина в ошибке проверки подлинности или связи), то DRMManager отправляет объект DRMErrorEvent или DRMAuthenticationErrorEvent.

## События DRMAuthenticationComplete

**Flash Player 10.1 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Класс DRMManager отправляет объект DRMAuthenticationCompleteEvent, если проверка подлинности пользователя с помощью метода `authenticate()` прошла успешно.

Объект DRMAuthenticationCompleteEvent содержит многократно используемую метку, которую можно применять для сохранения проверки подлинности пользователя в различных сеансах приложения. Передайте этот маркер в метод `setAuthenticationToken()` класса DRMManager, чтобы восстановить сеанс. (Создатель маркера устанавливает такие его атрибуты, как срок действия. Компания Adobe не предоставляет API-интерфейс для проверки атрибутов метки.)

## События DRMAuthenticationError

**Flash Player 10.1 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Класс DRMManager отправляет объект DRMAuthenticationErrorEvent, когда пользователь не может успешно пройти проверку подлинности с помощью методов `authenticate()` или `setAuthenticationToken()`.

# Использование класса DRMContentData

**Flash Player 10.1 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

Объект DRMContentData содержит свойства метаданных содержимого, защищенного Adobe Access. Свойства DRMContentData содержат информацию, необходимую для получения лицензионного ваучера для просмотра содержимого. Класс DRMContentData можно использовать для получения файла метаданных, связанного с содержимым, как описано в разделе ««Подробный технологический процесс API-интерфейса» на странице 568».

Дополнительные сведения см. в описании класса DRMContentData в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

# Обновление проигрывателя Flash Player для поддержки Adobe Access

**Flash Player 10.1 или более поздних версий**

***Важная информация***. В состав Flash Player 11.5 и более поздних версий уже включен модуль Adobe Access, поэтому обновление (посредством метода `SystemUpdater.update(SystemUpdaterType.DRM)`) не требуется. Это относится к следующим браузерам и платформам:

- Компонент Flash Player 11.5 ActiveX control — для всех платформ, за исключением Internet Explorer на Windows 8;
- Плагин Flash Player 11.5 — для всех браузеров;
- Adobe AIR (для настольных и мобильных систем).

Это означает, что обновление *все же требуется* в следующих случаях:

- Internet Explorer на Windows 8;
- Flash Player 11.4 и более старых версий, за исключением браузера Google Chrome версии 22 и более поздней (для всех платформ) или 21 и более поздней (для Windows).

***Примечание.*** *Вызов метода `SystemUpdater.update(SystemUpdaterType.DRM)` в системах с Flash Player 11.5 и более поздних версий не приведет к ошибкам, и при этом ничего не будет загружено.*

Для поддержки Adobe Access в проигрывателе Flash Player требуется модуль Adobe Access. Когда проигрыватель Flash Player пытается воспроизвести защищенное содержимое, в среде выполнения показано, необходимо ли загрузить модуль или новую версию проигрывателя Flash Player. Таким способом проигрыватель Flash Player предоставляет разработчикам SWF возможность отказаться от обновления.

В большинстве случаев для воспроизведения защищенного содержимого разработчикам SWF потребуется выполнить обновление для получения необходимого модуля Adobe Access или версии проигрывателя. Для обновления можно использовать API-интерфейс SystemUpdater, позволяющий получить последнюю версию модуля Adobe Access или проигрывателя Flash Player.

API-интерфейс SystemUpdater позволяет выполнять только одно обновление за один раз. Ошибка с кодом 2202 обозначает, что обновление уже выполняется в текущем или другом экземпляре среды выполнения. Например, если обновление выполняется в экземпляре Flash Player в браузере Internet Explorer, выполнение обновления в экземпляре Flash Player в браузере Firefox невозможно.

API-интерфейс SystemUpdater поддерживается только для настольных платформ.

***Примечание.*** *Для версий Flash Player, предшествующих 10.1, используйте механизм обновления, поддерживаемый в более ранних версиях проигрывателя (загрузка вручную или установка с сайта www.adobe.com или ExpressInstall). Помимо этого, установщик AIR обрабатывает необходимые обновления для Adobe Access и не поддерживает API-интерфейс SystemUpdater.*

## Прослушивание события обновления

**Flash Player 10.1 или более поздних версий**

Когда требуется обновление модуля Adobe Access, объект NetStream отправляет событие NetStatusEvent с кодовым значением `DRM.UpdateNeeded`. Это значение показывает, что объекту NetStream не удается воспроизвести защищенный поток с использованием какого-либо из установленных в данный момент модулей Adobe Access. Прослушайте это событие и вызовите следующий код:

```
SystemUpdater.update(flash.system.SystemUpdaterType.DRM)
```

Этот код позволяет обновить модуль Adobe Access, установленный в проигрывателе. Согласие пользователя на обновление этого модуля не требуется.

Если модуль Adobe Access не найден, возникает ошибка. См. действие 3 в разделе ««Подробный технологический процесс API-интерфейса» на странице 568».

***Примечание.*** *Если метод play() вызывается для зашифрованного потока в версиях проигрывателя, предшествующих 10.1, отправляется объект NetStatusEvent с кодовым значением NetStream.Play.StreamNotFound. В более ранних версиях проигрывателя используйте механизм обновления, поддерживаемый в этих проигрывателях (обновление вручную или установка с сайта www.adobe.com или ExpressInstall).*

Если требуется обновление самого проигрывателя, объект SystemUpdater отправляет событие StatusEvent с кодовым значением `DRM.UpdateNeededButIncompatible`. Для обновления проигрывателя требуется согласие пользователя. В приложении предусмотрите интерфейс, в котором пользователь сможет дать согласие и запустить обновление проигрывателя. Прослушайте событие StatusEvent и вызовите следующий код:

```
SystemUpdater.update(flash.system.SystemUpdaterType.SYSTEM);
```

Этот код позволяет запустить обновление проигрывателя.

Дополнительные события класса SystemUpdater описаны в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

По завершении обновления проигрывателя выполняется переход на страницу, на которой началось обновление. Модуль Adobe Access загружен, и можно начать воспроизведение потока.

# Внешние лицензии

**Flash Player 11 и более поздних версий, Adobe AIR 3.0 и более поздних версий**

Можно также получать внешние лицензии (без обращения на сервер лицензий Adobe Access). Для этого сохраните ваучер (лицензию) на диске и в памяти с помощью метода `storeVoucher`.

Для воспроизведения зашифрованных видео в Flash Player и AIR, соответствующая среда выполнения должна получить ваучер DRM для этого видео. Ваучер DRM содержит ключ расшифровки видео и генерируется сервером лицензий Adobe Access, который развернут клиентом.

Среда выполнения Flash Player/AIR, как правило, получает этот ваучер, отправляя запрос на сервер лицензий Adobe Access, указанный в метаданных DRM этого видео (класс `DRMContentData`). Приложение Flash/AIR может запускать отправку этого запроса на лицензию, вызвав метод `DRMManager.loadVoucher()`. Либо среда выполнения Flash Player/AIR автоматически запросит лицензию при запуске воспроизведения зашифрованного видео, если на диске или в памяти отсутствует лицензия для содержимого. В любом случае производительность приложения Flash/AIR замедляется при обращении к серверу лицензий Adobe Access.

`DRMManager.storeVoucher()` позволяет приложению Flash/AIR отправлять внешние ваучеры DRM в среду выполнения Flash Player/AIR. После этого среда выполнения может пропустить процесс запроса лицензии и использовать перенаправленные ваучеры для воспроизведения зашифрованных видео. Ваучер DRM все равно должен создаваться сервером лицензий Adobe Access, прежде чем его можно будет получить как внешнюю лицензию. Однако существует возможность хранения ваучеров на любом сервере HTTP, а не на сервере лицензий Adobe Access для пользователей.

`DRMManager.storeVoucher()` также используется для обеспечения поддержки совместного использования ваучера DRM несколькими устройствами. В Adobe Access 3.0 эта функция называется поддержкой домена. Если развертывание поддерживает эту возможность, можно зарегистрировать несколько устройств в группе в помощью метода `DRMManager.addToDeviceGroup()`. Если в группе есть компьютер с действующим ваучером, привязанным к домену, для определенного содержимого, приложение AIR может извлекать сериализованные ваучеры DRM с помощью метода `DRMVoucher.toByteArray()`, эти ваучеры можно импортировать на устройства с помощью метода `DRMManager.storeVoucher()`.

## Регистрация устройств

Ваучеры DRM привязаны к устройству конечного пользователя. По этой причине приложениям Flash/AIR потребуется уникальный идентификатор для устройства пользователя, чтобы ссылаться на правильный сериализованный ваучер DRM. Следующий сценарий демонстрирует процесс регистрации устройства.

Предположим, что уже выполнены следующие задачи:

- настроено ПО Adobe Access Server SDK;
- настроен HTTP-сервер для получения предварительно выпущенных лицензий;
- создано приложение Flash для просмотра защищенного содержимого.

На этапе регистрации устройства выполняются следующие операции.

1. Приложение Flash создает произвольно генерируемый идентификатор.
2. Приложение Flash вызывает метод `DRMManager.authenticate()`. Приложение должно включить произвольно сгенерированный идентификатор в запрос на проверку подлинности. Например, включите идентификатор в поле «Имя пользователя».
3. Действие, упомянутое в п. 2, приводит к тому, что Adobe Access отправляет запрос на проверку подлинности серверу пользователя. Этот запрос включает сертификат устройства.
   a. Сервер извлекает и сохраняет сертификат устройства и сгенерированный идентификатор из запроса.
   b. Подсистема пользователя предварительно генерирует лицензии для данного сертификата устройства, сохраняет их и предоставляет доступ к ним, связывая их со сгенерированным идентификатором.
4. Сервер отвечает на запрос сообщением об успешном выполнении.
5. Приложение Flash сохраняет сгенерированный идентификатор локально в объекте LSO (локальный общий объект).

После регистрации устройства приложение Flash использует сгенерированный идентификатор так же, как использовался бы идентификатор устройства в предыдущей схеме.

1. Приложение Flash попытается найти сгенерированный идентификатор в объекте LSO.
2. Если удается найти сгенерированный идентификатор, приложение Flash будет использовать его при загрузке предварительно созданных лицензий. Приложение Flash отправляет лицензии клиенту Adobe Access для использования с помощью метода `DRMManager.storeVoucher()`.
3. Если не удается найти сгенерированный идентификатор, приложение Flash выполняет регистрацию устройства.

### Восстановление заводских настроек

Когда пользователь устройства выбирает восстановление заводских настроек, сертификат устройства удаляется. Чтобы продолжить воспроизведение защищенного содержимого, приложению Flash потребуется повторно выполнить регистрацию устройства. Если приложение Flash отправляет устаревшую предварительно созданную лицензию, клиент Adobe Access отклоняет ее, так как лицензия зашифрована для более ранней версии идентификатора устройства.

## Поддержка домена

**Flash Player 11 и более поздних версий, Adobe AIR 3.0 и более поздних версий**

Если в метаданных содержимого указано, что требуется регистрация в домене, приложение AIR может вызвать API-интерфейс для присоединения к группе устройств. Это действие инициирует запрос на регистрацию в домене, который необходимо отправить серверу домена. Когда выпускается лицензия для группы устройств, ее можно экспортировать и использовать совместно с другими устройствами, который присоединились к группе.

После этого информация о группе устройств используется объектом `DRMContentDataVoucherAccessInfo`, который затем будет использоваться для представления информации, необходимой для успешного извлечения и использования ваучера.

## Воспроизведение защищенного содержимого с помощью поддержки домена

Для воспроизведения зашифрованного содержимого с помощью Adobe Access выполните следующие действия.

1. Используя `VoucherAccessInfo.deviceGroup`, проверьте, требуется ли регистрация группы устройств.
2. Если требуется ли проверка подлинности
   - **a** Используйте свойство `DeviceGroupInfo.authenticationMethod`, чтобы узнать, требуется ли проверка подлинности.
   - **b** Если проверка подлинности требуется, выполните ОДНО из следующих действий для авторизации пользователя.
     - Получите имя пользователя и пароль. Вызовите метод `DRMManager.authenticate(deviceGroup.serverURL, deviceGroup.domain, username, password)`.
     - Получите кэшированный, предварительно созданный маркер проверки подлинности и вызовите метод `DRMManager.setAuthenticationToken()`.
   - **c** Вызовите метод `DRMManager.addToDeviceGroup()`.
3. Получите ваучер для содержимого, выполнив одно из следующих действий.
   - **a** Используйте метод `DRMManager.loadVoucher()`.
   - **b** Получите ваучер от другого устройства, зарегистрированного в той же группе устройств. Предоставьте ваучер для `DRMManager` с помощью метода `DRMManager.storeVoucher()`.

**4** Воспроизведите зашифрованное содержимое с помощью метода `NetStream.play()`.

Чтобы экспортировать лицензию для содержимого, любое из устройств может предоставить необработанные байты лицензии с помощью метода `DRMVoucher.toByteArray()` после получения лицензии от сервера лицензий Adobe Access. Поставщики содержимого, как правило ограничивают число устройств, которые можно зарегистрировать в группе. Если достигнут предел, может потребоваться вызвать метод `DRMManager.removeFromDeviceGroup()` для неиспользуемого устройства, прежде чем зарегистрировать текущее.

# Предварительный просмотр

Приложение Flash может отправлять запрос на предварительный просмотр лицензии, то есть, прежде чем просить пользователя приобрести содержимое, она может выполнить операцию предварительного просмотра, чтобы определить, отвечает ли устройство пользователя всем критериям, необходимым для воспроизведения. Предварительный просмотр лицензии — это возможность пользователя просматривать лицензию (чтобы узнать, какие права предоставляет лицензия), в отличие от предварительного просмотра содержимого (то есть просмотра небольшого фрагмента содержимого перед покупкой). Имеется ряд параметров, уникальных для каждого устройства, в том числе доступные выходы, их состояние защиты, доступная версия среды выполнения/DRM, а также уровень безопасности клиента DRM. Режим предварительного просмотра лицензии позволяет среде выполнения/клиенту DRM проверить операционную логику сервера лицензий и предоставить пользователю информацию, необходимую для принятия взвешенного решения. Таким образом пользователь может узнать, что представляет собой действительная лицензия, но на самом деле не получает ключ для расшифровки содержимого. Поддержка предварительного просмотра лицензий является необязательной. Она необходима, только если реализуется пользовательское приложение с использованием этой функции.

# Предоставление содержимого

Adobe Access не зависит от механизма предоставления содержимого, так как Flash Player абстрагирует сетевой слой и просто предоставляет защищенное содержимое подсистеме Adobe Access. Таким образом, содержимое может передаваться через HTTP, HTTP Dynamic Streaming, RTMP или RTMPE.

Однако могут возникать некоторые проблемы, так как необходимо получить метаданные защищенного содержимого (обычно содержащиеся в файле с расширением .metadata), прежде чем Adobe Access сможет получить лицензию для расшифровки содержимого. В частности, при использовании протокола RTMP/RTMPE через Flash Media Server (FMS) клиенту можно передавать только данные в форматах FLV и F4V. По этой причине клиент должен извлекать большой двоичный объект метаданных другими способами. Один из вариантов решения этой проблемы заключается в том, чтобы разместить метаданные на веб-сервере HTTP и в клиентском видеопроигрывателе предусмотреть получение соответствующих метаданных в зависимости от воспроизводимого содержимого.

```
private function getMetadata():void{

    extrapolated-path-to-metadata = "http://metadatas.mywebserver.com/" + videoname;
    var urlRequest : URLRequest = new URLRequest(extrapolated-path-to-the-metadata + ".metadata");
    var urlStream : URLStream = new URLStream();
    urlStream.addEventListener(Event.COMPLETE, handleMetadata);
    urlStream.addEventListener(IOErrorEvent.NETWORK_ERROR, handleIOError);
    urlStream.addEventListener(IOErrorEvent.IO_ERROR, handleIOError);
    urlStream.addEventListener(IOErrorEvent.VERIFY_ERROR, handleIOError);
    try{
        urlStream.load(urlRequest);
     }catch(se:SecurityError){
        videoLog.text += se.toString() + "\n";
     }catch(e:Error){
        videoLog.text += e.toString() + "\n";
     }
}
```

# Среда Open Source Media Framework

Платформа OSMF (Open Source Media Framework) основана на ActionScript и предоставляет максимальную гибкость и полное управление процессом создания собственных мультимедийных приложений. Дополнительные сведения о платформе OSMF см. на сайте разработчика OSMF.

## Рабочий процесс воспроизведения защищенного содержимого

1 Создайте экземпляр `MediaPlayer`.

```
player = new MediaPlayer();
```

2 Зарегистрируйте событие `MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE` для проигрывателя. Это событие будет отправляться, если содержимое защищено с помощью DRM.

```
player.addEventListener(MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE,
onDRMCapabilityChange);
```

3 В обработчике событий получите экземпляр `DRMTrait`. `DRMTrait` — это интерфейс, через который вызываются методы, связанные с DRM, например `authenticate()`. При загрузке содержимого, защищенного с помощью DRM платформа OSMF выполняет операции проверки DRM и отправляет события состояния. Добавьте обработчик события `DRMEvent.DRM_STATE_CHANGE` для `DRMTrait`.

```
private function onDRMCapabilityChange
    (event :MediaPlayerCapabilityChangeEvent) :void
        {
            if (event.type == MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE
                && event.enabled)
            {
                drmTrait = player.media.getTrait(MediaTraitType.DRM) as DRMTrait;
                drmTrait.addEventListener
                    (DRMEvent.DRM_STATE_CHANGE, onDRMStateChange);
            }
        }
```

4 Обрабатывайте события DRM в методе `onDRMStateChange()`.

```
private function onDRMStateChange(event :DRMEvent) :void
{
    trace ( "DRMState: ",event.drmState);
    switch(event.drmState)
    {
        case DRMState.AUTHENTICATION_NEEDED:
            // Identity-based content
            var authPopup :AuthWindow = AuthWindow.create(_parentWin);
            authPopup.serverURL = event.serverURL;
            authPopup.addEventListener("dismiss", function () :void {
                trace ("Authentication dismissed");
                if(_drmTrait != null)
                {
                    //Ignore authentication. Just
                    //try to acquire a license.
                    _drmTrait.authenticate(null, null);
                }
            });
            authPopup.addEventListener("authenticate",
                           function (event :AuthWindowEvent) :void {
                if(_drmTrait != null)
                {
                    _drmTrait.authenticate(event.username, event.password);
                }
            });
            authPopup.show();
            break;
        case DRMState.AUTHENTICATING:
            //Display any authentication message.
            trace("Authenticating...");
            break;
        case DRMState.AUTHENTICATION_COMPLETE:
            // Start to retrieve voucher and playback.
            // You can display the voucher information at this point.
            if(event.token)
            // You just received the authentication token.
            {
                trace("Authentication success. Token: \n", event.token);
            }
            else
            // You have got the voucher.
            {
                trace("DRM License:");
                trace("Playback window period: ",
                    !isNaN(event.period) ? event.period == 0 ?
                    "<unlimited>" : event.period : "<none>");
```

```
                trace("Playback window end date: ",
                    event.endDate != null ? event.endDate : "<none>");
                trace("Playback window start date: ",
                    event.startDate != null ? event.startDate : "<none>");
            }
            break;
        case DRMState.AUTHENTICATION_ERROR:
            trace ("DRM Error:", event.mediaError.errorID +
                "[" + DRMErrorEventRef.getDRMErrorMnemonic
                (event.mediaError.errorID) + "]");
            //Stop everything.
            player.media = null;
            break;
        case DRMState.DRM_SYSTEM_UPDATING:
            Logger.log("Downloading DRM module...");
            break;
        case DRMState.UNINITIALIZED:
            break;
    }
}
```

# Глава 28. Добавление PDF-содержимого в приложения AIR

**Adobe AIR 1.0 и более поздних версий**

Приложения для Adobe® AIR® могут выполнять визуализацию не только SWF- и HTML-, но и PDF-содержимого. Приложения AIR осуществляют визуализацию PDF с помощью класса HTMLLoader, механизма WebKit и подключаемого модуля Adobe® Reader® для обозревателей. В приложении AIR PDF-содержимое может либо растягиваться по всей длине и ширине, либо заполнять какую-то часть интерфейса. Внешний модуль Adobe Reader для браузера управляет отображением PDF-файлов в приложениях AIR. Изменения, внесенные в интерфейс панели инструментов Reader (например, элементы управления позицией, привязками и видимостью), сохраняются при последующем просмотре PDF-файлов и в приложениях AIR и в браузере.

***Важная информация.*** *Для визуализации PDF в среде AIR пользователю необходимо установить Adobe Reader или Adobe® Acrobat® версии 8.1 или более поздней.*

## Определение возможности отображения PDF

**Adobe AIR 1.0 и более поздних версий**

Если у пользователя не установлено ПО Adobe Reader или Adobe Acrobat 8.1 или более поздней версии, то PDF-содержимое не будет отображаться в приложении AIR. Чтобы определить, будет ли у пользователя отображаться PDF, прежде всего проверьте свойство `HTMLLoader.pdfCapability`. Его значением будет одно из следующих значений класса HTMLPDFCapability:

| Константа | Описание |
|---|---|
| HTMLPDFCapability.STATUS_OK | Обнаружена подходящая версия (8.1 или более поздняя) Adobe Reader, PDF-содержимое может быть загружено в объект HTMLLoader. |
| HTMLPDFCapability.ERROR_INSTALLED_READER_NOT_FOUND | ПО Adobe Reader не обнаружено. Объект HTMLLoader не будет отображать PDF-содержимое. |
| HTMLPDFCapability.ERROR_INSTALLED_READER_TOO_OLD | ПО Adobe Reader обнаружено, но его версия устарела. Объект HTMLLoader не будет отображать PDF-содержимое. |
| HTMLPDFCapability.ERROR_PREFERRED_READER_TOO_OLD | Обнаружена подходящая версия (8.1 или более поздняя) Adobe Reader, но версия Adobe Reader, отвечающая за обработку PDF-содержимого, старше, чем 8.1. Объект HTMLLoader не будет отображать PDF-содержимое. |

Если в системе Windows в данное время запущено ПО Adobe Acrobat или Adobe Reader версии 7.x или более новой, то будет использоваться именно эта версия, даже если установлена более поздняя, поддерживающая PDF. В этом случае, если значение свойства `pdfCapability` равно `HTMLPDFCapability.STATUS_OK`, то, когда приложение AIR попытается загрузить PDF, устаревшая версия Acrobat или Reader выведет предупреждение (в приложении AIR при этом не будет сгенерировано исключение). Если такая ситуация возможна, постарайтесь сообщить своим конечным пользователям, что необходимо закрыть Acrobat перед запуском приложения. Например, подобное сообщение можно вывести, если PDF-файл не загружается за разумное время.

AIR, установленная на ОС Linux, выполняет поиск Adobe Reader, используя путь (PATH), экспортированный пользователем (если он содержит команду acroread), а также в каталоге /opt/Adobe/Reader.

Следующий код определяет, может ли система пользователя отображать PDF-содержимое в приложении AIR. Если система пользователя не поддерживает отображение PDF, отслеживается код ошибки, соответствующий объекту ошибки HTMLPDFCapability.

```
 if(HTMLLoader.pdfCapability == HTMLPDFCapability.STATUS_OK)
{
    trace("PDF content can be displayed");
}
else
{
    trace("PDF cannot be displayed. Error code:", HTMLLoader.pdfCapability);
}
```

# Загрузка PDF-содержимого

**Adobe AIR 1.0 и более поздних версий**

PDF можно добавлять в приложения AIR путем создания экземпляра HTMLLoader, задания его размеров и загрузки пути к PDF-файлу.

Ниже показано, как загрузить PDF с внешнего сайта. Замените URLRequest путем к доступному внешнему PDF-файлу.

```
 var request:URLRequest = new URLRequest("http://www.example.com/test.pdf");
pdf = new HTMLLoader();
pdf.height = 800;
pdf.width = 600;
pdf.load(request);
container.addChild(pdf);
```

Содержимое также может быть загружено через URL-адреса файлов и характерные для AIR схемы URL-адресов, например app и app-storage. Так, ниже показано, как загрузить файл test.pdf в подкаталог PDFs каталога приложения:

app:/js_api_reference.pdf

Дополнительные сведения о схемах URL в AIR см. в разделе ««Схемы URI» на странице 874».

# Сценарии передачи PDF

**Adobe AIR 1.0 и более поздних версий**

С помощью JavaScript можно управлять PDF-содержимым точно так же, как это делается на веб-странице в обозревателе.

Расширения JavaScript для Acrobat, помимо прочего, предлагают следующие возможности:

- управление навигацией и масштабированием страницы
- обработка форм в документе
- управление событиями мультимедиа

Подробную информацию о расширениях JavaScript для Adobe Acrobat см. в Центре разработки Adobe Acrobat по адресу http://www.adobe.com/devnet/acrobat/javascript.html.

## Основы коммуникации между HTML и PDF

**Adobe AIR 1.0 и более поздних версий**

JavaScript на HTML-странице может отправлять сообщение JavaScript в PDF-документе, вызывая метод `postMessage()` объекта DOM, представляющего PDF. Например, рассмотрим следующий вложенный PDF-документ:

```
<object id="PDFObj" data="test.pdf" type="application/pdf" width="100%" height="100%"/>
```

Показанный ниже код JavaScript в HTML-содержимом отправляет сообщение коду JavaScript в PDF-файле:

```
pdfObject = document.getElementById("PDFObj");
pdfObject.postMessage(["testMsg", "hello"]);
```

Для получения этого сообщения PDF-файл может содержать JavaScript. В некоторых контекстах код JavaScript может добавляться к PDF-файлам, например на уровне документа, папки, страницы, поля и группы объектов. Здесь рассматривается только контекст уровня документа, который определяет проверяемые при открытии PDF-документа сценарии.

PDF-файл может добавлять объекту `hostContainer` свойство `messageHandler`. Свойство `messageHandler` — это объект, который определяет функции обработчиков, отвечающих на сообщения. Например, следующий код определяет функцию для обработки сообщений, получаемых PDF-файлом от домашнего контейнера (т. е. HTML-содержимого, в которое вложен PDF-файл):

```
this.hostContainer.messageHandler = {onMessage: myOnMessage};

function myOnMessage(aMessage)
{
    if(aMessage[0] == "testMsg")
    {
        app.alert("Test message: " + aMessage[1]);
    }
    else
    {
        app.alert("Error");
    }
}
```

Код JavaScript на HTML-странице может вызывать метод `postMessage()` объекта PDF, содержащегося на странице. Вызов этого метода отправляет сообщение (`"Hello from HTML"`) коду JavaScript в PDF-файле на уровне документа:

```
<html>
    <head>
    <title>PDF Test</title>
    <script>
        function init()
        {
            pdfObject = document.getElementById("PDFObj");
            try {
                 pdfObject.postMessage(["alert", "Hello from HTML"]);
            }
            catch (e)
            {
                alert( "Error: \n name = " + e.name + "\n message = " + e.message );
            }
        }
    </script>
    </head>
    <body onload='init()'>
        <object
            id="PDFObj"
            data="test.pdf"
            type="application/pdf"
            width="100%" height="100%"/>
    </body>
</html>
```

Дополнительные примеры и сведения об использовании Acrobat 8 для добавления JavaScript в PDF-файл см. в разделе «Перекрестное выполнение сценариев PDF-содержимого в Adobe AIR».

## Сценарии передачи PDF из ActionScript

**Adobe AIR 1.0 и более поздних версий**

Код ActionScript (в SWF-содержимом) не может напрямую обращаться к JavaScript в PDF-файле. Тем не менее ActionScript может обращаться к коду JavaScript на HTML-странице, загруженной в объект HTMLLoader, который загружает PDF, а этот код JavaScript — к коду JavaScript в загруженном PDF-файле. Дополнительные сведения см. в разделе ««Программирование HTML и JavaScript в AIR» на странице 1054».

# Известные ограничения PDF в AIR

**Adobe AIR 1.0 и более поздних версий**

С PDF-содержимым в Adobe AIR связаны некоторые ограничения:

- PDF-содержимое не отображается в окне (объекте NativeWindow), если последнее прозрачно (т. е. свойство `transparent` равно `true`).

- Порядок отображения PDF-файла работает иначе, чем для других экранных объектов в приложении AIR. Хотя усечение PDF-содержимого выполняется корректно, согласно порядку отображения HTML, это содержимое всегда будет стоять первым в порядке отображения приложения AIR.
- Если изменяются определенные свойства визуализации объекта HTMLLoader, который содержит PDF-документ, этот PDF-документ становится невидимым. К числу этих свойств относятся: `filters`, `alpha`, `rotation` и `scaling`. Изменение этих свойств делает PDF-содержимое невидимым до тех пор, пока эти свойства не будут переустановлены. То же самое происходит и при изменении этих свойств для контейнеров экранных объектов, содержащих объект HTMLLoader.
- PDF-содержимое видно только в том случае, если для свойства `scaleMode` объекта Stage объекта NativeWindow, в котором находится это PDF-содержимое, установлено значение `StageScaleMode.NO_SCALE`. Если установлено какое-либо другое значение, PDF-содержимое не видно.
- При щелчке по ссылке на содержимое внутри PDF-файла положение прокрутки в PDF обновляется. При щелчке по ссылке на содержимое вне PDF-файла объект HTMLLoader, содержащий PDF, перенаправляется (даже если ссылка открывается в новом окне).
- Комментарии PDF не поддерживаются в AIR.

# Глава 29. Основы взаимодействия пользователя с системой

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для ответной реакции на активность пользователя ваше приложение при помощи ActionScript 3.0 может создавать интерактивность. Обратите внимание, что для понимания этого раздела вы должны быть знакомы с моделью событий ActionScript 3.0. Дополнительные сведения см. в разделе ««Обработка событий» на странице 134».

## Захват действий пользователя

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В основе интерактивности лежит взаимодействие с пользователем через клавиатуру, мышь, камеру или все вместе. В ActionScript 3.0 определение и реагирование на действия пользователя прежде всего подразумевает прослушивание событий.

Класс InteractiveObject, подкласс класса DisplayObject, обеспечивает общую структуру событий и функциональность, необходимые для взаимодействия с пользователем. Экземпляр класса InteractiveObject не создается напрямую. Вместо этого такие экранные объекты, как SimpleButton, Sprite, TextField и различные компоненты инструмента разработки Flash и Flex, наследуют модели взаимодействия с пользователем от этого класса и поэтому имеют одинаковую структуру. Это означает, что изученные вами техники и код, который вы напишете для обработки взаимодействия с пользователем объектов, созданных на основе InteractiveObject, можно будет применять и ко всем остальным объектам.

### Важные понятия и термины

Прежде чем продолжать, ознакомьтесь с ключевыми терминами, касающимися взаимодействия с пользователем.

**Код символа** Числовой код, обозначающий символ в текущей раскладке (связанный с клавишей, нажатой на клавиатуре). Например, буквы «D» и «d» в латинской раскладке имеют разные коды, хотя они и набираются с помощью одной и той же клавиши.

**Контекстное меню** Меню, которое появляется по щелчку правой кнопкой мыши или при использовании определенной комбинации клавиш на клавиатуре и мыши. Команды контекстного меню обычно относятся исключительно к тому элементу, на который нажал пользователь. Например, в контекстном меню для изображения могут содержаться команды «Показать изображение» или «Открыть изображение в новом окне».

**Фокус** Индикатор того, что выбранный элемент активен и является целью взаимодействия с помощью мыши или клавиатуры.

**Код клавиши** Числовой код, соответствующий физической клавише на клавиатуре.

### Дополнительные разделы справки

InteractiveObject

Keyboard

Mouse

ContextMenu

# Управление фокусом

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Интерактивный объект может получать фокус программными средствами или вследствие действия пользователя. Кроме того, если свойство `tabEnabled` имеет значение `true`, пользователь может передавать фокус от одного объекта другому клавишей табуляции. Обратите внимание, что по умолчанию значение `tabEnabled = false`, кроме следующих случаев.

- Для объекта SimpleButton используется значение `true`.
- Для поля ввода текста используется значение `true`.
- Для объекта Sprite или MovieClip с `buttonMode = true` используется значение `true`.

В каждом из этих случаев можно добавить прослушиватель для событий `FocusEvent.FOCUS_IN` или `FocusEvent.FOCUS_OUT`, чтобы добавить дополнительные виды поведения при смене фокуса. Это особенно полезно для полей ввода текста и форм, но также может пригодиться для спрайтов, фрагментов роликов и любых других объектов, наследуемых от класса InteractiveObject. В примере ниже показано, как включить зацикливание фокуса с помощью клавиши табуляции и как реагировать на последующие события фокуса. В этом случае каждый квадрат при получении фокуса меняет цвет.

***Примечание.*** *Среда разработки Flash Professional управляет фокусом с помощью клавиатуры, поэтому для правильной симуляции управления фокусом SWF-файлы необходимо тестировать в обозревателе или среде AIR, а не в средстве разработки Flash.*

```
var rows:uint = 10;
var cols:uint = 10;
var rowSpacing:uint = 25;
var colSpacing:uint = 25;
var i:uint;
var j:uint;
for (i = 0; i < rows; i++)
{
    for (j = 0; j < cols; j++)
    {
        createSquare(j * colSpacing, i * rowSpacing, (i * cols) + j);
    }
}

function createSquare(startX:Number, startY:Number, tabNumber:uint):void
{
    var square:Sprite = new Sprite();
    square.graphics.beginFill(0x000000);
    square.graphics.drawRect(0, 0, colSpacing, rowSpacing);
    square.graphics.endFill();
```

```
    square.x = startX;
    square.y = startY;
    square.tabEnabled = true;
    square.tabIndex = tabNumber;
    square.addEventListener(FocusEvent.FOCUS_IN, changeColor);
    addChild(square);
}
function changeColor(event:FocusEvent):void
{
    event.target.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

# Обнаружение типов ввода

**Flash Player 10.1 и более поздних версий, Adobe AIR 2 и более поздних версий**

В выпусках Flash Player 10.1 и Adobe AIR 2 представлена возможность тестирования среды выполнения на наличие поддержки определенных типов ввода. ActionScript можно использовать для тестирования устройства, в котором в настоящее время развернута среда выполнения, на наличие следующих видов поддержки:

- Поддержка ввода с помощью пера или пальца (или полное отсутствие поддержки ввода с помощью прикосновения).
- Наличие виртуальной или физической клавиатуры для пользователя (или полное отсутствие клавиатуры).
- Отображение курсора (если не отображается, функции, которые зависят от наведения курсора на объект, не работают).

API-интерфейсы обнаружения ввода в среде ActionScript включают:

- Свойство flash.system.Capabilities.touchScreenType: значение, предоставляемое во время выполнения и показывающее тип ввода, поддерживаемый в текущей среде.
- Класс flash.system.TouchScreenType: класс констант значений перечисления для свойства Capabilities.touchScreenType.
- Свойство flash.ui.Mouse.supportsCursor: значение, предоставляемое во время выполнения и показывающее доступность постоянного курсора.
- Свойство flash.ui.Keyboard.physicalKeyboardType: значение, предоставляемое во время выполнения и показывающее доступность полной физической клавиатуры или только числовой клавиатуры либо полное отсутствие клавиатуры.

- Класс flash.ui.KeyboardType: класс констант значений перечисления для свойства flash.ui.Keyboard.physicalKeyboardType.
- Свойство flash.ui.Keyboard.hasVirtualKeyboard: значение, предоставляемое во время выполнения и показывающее доступность пользователю виртуальной клавиатуры (вместо физической клавиатуры или в добавление к ней).

API-интерфейсы обнаружения ввода позволяют эффективно использовать возможности пользовательского устройства или предоставлять альтернативные варианты, если эти возможности отсутствуют. Эти API-интерфейсы особенно полезны при разработке мобильных приложений или приложений с поддержкой сенсорного ввода. Например, при наличии интерфейса для мобильного устройства, в котором предусмотрено небольшое число кнопок для нажатия пером, можно предоставить альтернативный интерфейс с большим числом кнопок, чтобы пользователь мог осуществлять ввод прикосновением пальца. Следующий код предназначен для приложения, в котором используется функция createStylusUI(), назначающая один набор элементов пользовательского интерфейса соответствующему взаимодействию с помощью пера. Другая функция, createTouchUI(), назначает другой набор элементов пользовательского интерфейса соответствующему взаимодействию с помощью прикосновения пальцем:

```
if(Capabilities.touchscreenType == TouchscreenType.STYLUS ){
    //Construct the user interface using small buttons for a stylus
    //and allow more screen space for other visual content
    createStylusUI();
} else if(Capabilities.touchscreenType = TouchscreenType.FINGER){
    //Construct the user interface using larger buttons
    //to capture a larger point of contact with the device
    createTouchUI();
}
```

При разработке приложений для различных сред ввода учитывайте следующую таблицу совместимости:

| Среда | supportsCursor | touchscreenType == FINGER | touchscreenType == STYLUS | touchscreenType == NONE |
|---|---|---|---|---|
| Традиционный персональный компьютер | true | false | false | true |

| Среда | supportsCursor | touchscreenType == FINGER | touchscreenType == STYLUS | touchscreenType == NONE |
|---|---|---|---|---|
| Устройства с емкостным сенсорным экраном (планшеты, КПК и телефоны, обнаруживающие легкое прикосновение, такие как Apple iPhone или Palm Pre) | false | true | false | false |
| Устройства с резистивным сенсорным экраном (планшеты, КПК и телефоны, обнаруживающие точное сильное нажатие, такие как HTC Fuze) | false | false | true | false |
| Устройства без сенсорного экрана (популярные телефоны и устройства, в которых выполняются приложения, но отсутствует экран, обнаруживающий нажатие) | false | false | false | true |

***Примечание.*** *Различные платформы устройств могут поддерживать многие комбинации типов ввода. Используйте эту таблицу в качестве общего руководства.*

# Глава 30. Ввод с клавиатуры

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Ваше приложение может перехватывать ввод с клавиатуры и реагировать на него, а также манипулировать IME, чтобы дать возможность пользователям набирать текстовые символы, отличные от символов ASCII, в многобайтовых языках. Обратите внимание, что для понимания этого раздела вы должны быть знакомы с моделью событий ActionScript 3.0. Дополнительные сведения см. в разделе ««Обработка событий» на странице 134».

Сведения об обнаружении доступности поддержки клавиатуры (например, физической, виртуальной, буквенно-цифровой или 12-кнопочной числовой) во время выполнения см. в разделе ««Обнаружение типов ввода» на странице 599».

Редактор методов ввода (англ. Input Method Editor, IME) дает пользователям возможность вводить сложные символы и знаки с использованием стандартной клавиатуры. С помощью классов IME в приложении можно обеспечить поддержку системного редактора методов ввода.

### Дополнительные разделы справки

flash.events.KeyboardEvent

flash.system.IME

## Захват действий клавиатуры

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Экранные объекты, которые наследуют модели взаимодействия от класса InteractiveObject, могут реагировать на события клавиатуры. Для этого используются прослушиватели событий. Например, можно поместить прослушиватель событий в рабочую область для прослушивания действий клавиатуры и реагирования на них. Ниже приведен код, в котором прослушиватель событий захватывает нажатие клавиши, и отображаются название и код клавиши:

```
function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (character code: " +
event.charCode + ")");
}
stage.addEventListener(KeyboardEvent.KEY_DOWN, reportKeyDown);
```

Некоторые клавиши, например Ctrl, генерируют события даже несмотря на то, что им не соответствуют никакие глифы.

В предыдущем примере прослушиватель событий захватывает действия клавиатуры во всей рабочей области. Вы также можете написать прослушиватель событий для конкретного экранного объекта в рабочей области. Этот прослушиватель событий будет вызываться, когда фокус перейдет к объекту.

В примере ниже нажатия клавиш отображаются на панели «Вывод» только тогда, когда они совершаются внутри экземпляра TextField. Удерживание клавиши Shift временно меняет цвет рамки объекта TextField на красный.

В данном коде в рабочей области присутствует экземпляр TextField с именем `tf`.

```
tf.border = true;
tf.type = "input";
tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
tf.addEventListener(KeyboardEvent.KEY_UP,reportKeyUp);

function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (key code: " +
event.keyCode + " character code: " + event.charCode + ")");
    if (event.keyCode == Keyboard.SHIFT) tf.borderColor = 0xFF0000;
}

function reportKeyUp(event:KeyboardEvent):void
{
    trace("Key Released: " + String.fromCharCode(event.charCode) + " (key code: " +
event.keyCode + " character code: " + event.charCode + ")");
    if (event.keyCode == Keyboard.SHIFT)
    {
        tf.borderColor = 0x000000;
    }
}
```

Класс TextField также реагирует на событие `textInput`, которое можно прослушивать при вводе пользователем текста. Дополнительные сведения см. в разделе ««Захват введенного текста» на странице 409».

***Примечание.*** *В среде выполнения AIR события клавиатуры можно отменять. В среде Flash Player события клавиатуры отменять нельзя.*

## Коды клавиш и символов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для определения того, какая клавиша нажимается, и выполнения соответствующих действий используются свойства `keyCode` и `charCode` события клавиатуры. Свойство `keyCode` — это числовое значение, соответствующее значению клавиши на клавиатуре. Свойство `charCode` — это числовое значение данной клавиши в текущей раскладке клавиатуры. (Набор символов по умолчанию — UTF-8 с поддержкой ASCII.)

Основная разница между кодом клавиши и кодом символа в том, что значение кода клавиши представляет конкретную клавишу на клавиатуре («1» на боковой клавиатуре отличается от «1» в верхнем ряду, но клавиша, с помощью которой вводятся «1» и «!», одна и та же), а значение кода символа — это конкретный символ («R» и «r» — разные символы).

***Примечание.*** *Соответствия между клавишами и кодами их символов в ASCII см. в описании класса flash.ui.Keyboard в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.*

Связь клавиш с кодами клавиш зависит от устройства и операционной системы. По этой причине не следует использовать назначения клавиш для вызова действий. Вместо этого лучше использовать предопределенные значения констант класса Keyboard для ссылки на нужные свойства `keyCode`. Например, вместо того чтобы использовать «горячую» клавишу Shift, используйте константу `Keyboard.SHIFT` (как показано в предыдущем примере).

## Приоритет событий KeyboardEvent

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Как и в случае с другими событиями, последовательность событий клавиатуры определяется иерархией экранных объектов, а не порядком, в котором методы `addEventListener()` назначаются в коде.

Предположим, вы поместили текстовое поле `tf` во фрагмент ролика с именем `container` и добавили прослушиватель событий для событий клавиатуры к обоим экземплярам:

```
container.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
container.tf.border = true;
container.tf.type = "input";
container.tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);

function reportKeyDown(event:KeyboardEvent):void
{
    trace(event.currentTarget.name + " hears key press: " + String.fromCharCode(event.charCode)
+ " (key code: " + event.keyCode + " character code: " + event.charCode + ")");
}
```

Так как прослушиватель установлен и на текстовое поле, и на его родительский контейнер, функция `reportKeyDown()` вызывается дважды для каждого нажатия клавиши в текстовом поле TextField. Обратите внимание, что каждый раз при нажатии клавиши текстовое поле отправляет событие прежде, чем это делает фрагмент ролика `container`.

Операционная система и веб-обозреватель обрабатывают события прежде, чем это делает проигрыватель Adobe Flash Player или AIR. Например, если нажать в обозревателе Internet Explorer сочетание клавиш Ctrl+W, окно обозревателя закроется прежде, чем любой из содержащихся в нем SWF-файлов отправит событие клавиатуры.

# Использование класса IME

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс IME позволяет манипулировать редактором методов ввода (IME) в проигрывателе Flash Player или Adobe AIR.

С помощью ActionScript можно определить следующее:

- установлен ли редактор IME на компьютере пользователя (`Capabilities.hasIME`);
- включен или отключен редактор IME на компьютере пользователя (`IME.enabled`);
- текущий режим преобразования в редакторе IME (`IME.conversionMode`).

Можно связать поле вводимого текста с определенным контекстом редактора IME. Переключаясь между полями ввода, можно также переключать редактор IME между хираганой (японский), полноширинными цифрами, полуширинными цифрами, прямым вводом и т.д.

Редакторы IME дают пользователям возможность вводить тексты с многобайтовыми символами, отличными от ASCII, на таких языках, как китайский, японский и корейский.

Дополнительные сведения о работе с редактором IME см. в документации по операционной системе, для которой разрабатывается приложение. Дополнительные ресурсы доступны на следующих веб-сайтах:

- http://www.msdn.microsoft.com/goglobal/
- http://developer.apple.com/library/mac/navigation/
- http://www.java.sun.com/

***Примечание.*** *Если редактор IME не активен на компьютере пользователя, то обращения к методам или свойствам IME, отличным от* `Capabilities.hasIME`*, будут завершаться ошибками. Как только редактор IME будет активирован вручную, последующие обращения ActionScript к методам и свойствам IME будут срабатывать ожидаемым образом. Например, если используется редактор IME для ввода японских иероглифов, его необходимо активировать перед тем, как вызывать методы и свойства класса IME.*

## Проверка доступности редактора IME

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Перед тем как вызывать методы и свойства класса IME, необходимо всегда проверять, установлен ли на клиентском компьютере редактор IME и включен ли он. Следующий код демонстрирует, как можно убедиться, что пользователь установил и активировал редактор IME, прежде чем вызывать методы.

```
if (Capabilities.hasIME)
{
    if (IME.enabled)
    {
        trace("IME is installed and enabled.");
    }
    else
    {
        trace("IME is installed but not enabled. Please enable your IME and try again.");
    }
}
else
{
    trace("IME is not installed. Please install an IME and try again.");
}
```

Предыдущий код сначала проверяет, установлен ли редактор IME, с помощью свойства `Capabilities.hasIME`. Если это свойство имеет значение `true`, то код проверяет, включен ли редактор IME в данный момент, с помощью свойства `IME.enabled`.

## Определение текущего режима преобразования в редакторе IME

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В процессе создания многоязычных приложений может потребоваться определить, какой режим преобразования в данный момент активен в редакторе IME. Следующий код демонстрирует, как можно проверить, установлен ли редактор IME и, если да, какой режим преобразования активен в данный момент.

```
if (Capabilities.hasIME)
{
    switch (IME.conversionMode)
    {
        case IMEConversionMode.ALPHANUMERIC_FULL:
            tf.text = "Current conversion mode is alphanumeric (full-width).";
            break;
        case IMEConversionMode.ALPHANUMERIC_HALF:
            tf.text = "Current conversion mode is alphanumeric (half-width).";
            break;
        case IMEConversionMode.CHINESE:
            tf.text = "Current conversion mode is Chinese.";
            break;
        case IMEConversionMode.JAPANESE_HIRAGANA:
            tf.text = "Current conversion mode is Japananese Hiragana.";
            break;
        case IMEConversionMode.JAPANESE_KATAKANA_FULL:
            tf.text = "Current conversion mode is Japanese Katakana (full-width).";
            break;
        case IMEConversionMode.JAPANESE_KATAKANA_HALF:
            tf.text = "Current conversion mode is Japanese Katakana (half-width).";
            break;
        case IMEConversionMode.KOREAN:
            tf.text = "Current conversion mode is Korean.";
            break;
        default:
            tf.text = "Current conversion mode is " + IME.conversionMode + ".";
            break;
    }
}
else
{
    tf.text = "Please install an IME and try again.";
}
```

Предыдущий код сначала проверяет, установлен ли редактор IME. Затем он проверяет текущий режим преобразования IME сравнивая свойство `IME.conversionMode` с константами класса IMEConversionMode.

## Установка режима преобразования в редакторе IME

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В процессе изменения режима преобразования в клиентском редакторе IME необходимо поместить код в блок `try..catch`, так как установка режима преобразования с помощью свойства `conversionMode` может вызвать ошибку, если редактор IME не сможет перейти в заданный режим. Следующий код демонстрирует использование блока `try..catch` при определении свойства `IME.conversionMode`.

```
var statusText:TextField = new TextField;
statusText.autoSize = TextFieldAutoSize.LEFT;
addChild(statusText);
if (Capabilities.hasIME)
{
    try
    {
        IME.enabled = true;
        IME.conversionMode = IMEConversionMode.KOREAN;
        statusText.text = "Conversion mode is " + IME.conversionMode + ".";
    }
    catch (error:Error)
    {
        statusText.text = "Unable to set conversion mode.\n" + error.message;
    }
}
```

Предыдущий код сначала создает текстовое поле с целью отображения сообщения о состоянии для пользователя. После этого, если редактор IME установлен, код включает его и устанавливает режим корейского языка. Если в системе пользователя не установлен редактор IME, проигрыватель Flash Player или AIR выдает ошибку, которую перехватывает блок `try..catch`. Блок `try..catch` отображает сообщение об ошибке в текстовом поле, созданном ранее.

## Отключение редактора IME для определенных текстовых полей

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В некоторых случаях требуется отличать редактор IME пользователя для полей ввода. Например, если есть текстовое поле, принимающее только цифры, нежелательно, чтобы редактор IME вмешивался и замедлял ввод данных.

В следующем примере показано, как можно прослушивать события `FocusEvent.FOCUS_IN` и `FocusEvent.FOCUS_OUT` и выключать редактор IME требуемым образом.

```
var phoneTxt:TextField = new TextField();
var nameTxt:TextField = new TextField();

phoneTxt.type = TextFieldType.INPUT;
phoneTxt.addEventListener(FocusEvent.FOCUS_IN, focusInHandler);
phoneTxt.addEventListener(FocusEvent.FOCUS_OUT, focusOutHandler);
phoneTxt.restrict = "0-9";
phoneTxt.width = 100;
phoneTxt.height = 18;
phoneTxt.background = true;
phoneTxt.border = true;
addChild(phoneTxt);

nameField.type = TextFieldType.INPUT;
nameField.x = 120;
nameField.width = 100;
nameField.height = 18;
nameField.background = true;
nameField.border = true;
addChild(nameField);

function focusInHandler(event:FocusEvent):void
{
    if (Capabilities.hasIME)
    {
        IME.enabled = false;
    }
}
function focusOutHandler(event:FocusEvent):void
{
    if (Capabilities.hasIME)
    {
        IME.enabled = true;
    }
}
```

В этом примере создается два текстовых поля ввода, `phoneTxt` и `nameTxt`, а затем добавляется два прослушивателя событий для поля `phoneTxt`. Когда пользователь переводит фокус на текстовое поле `phoneTxt`, отправляется событие `FocusEvent.FOCUS_IN`, и редактор IME отключается. Когда текстовое поле `phoneTxt` теряет фокус, отправляется событие `FocusEvent.FOCUS_OUT`, и редактор IME включается снова.

## Прослушивание событий композиции IME

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

События композиции IME отправляются в момент задания строки композиции. Например, если пользователь включил и активировал редактор IME и вводит строку на японском, событие `IMEEvent.IME_COMPOSITION` будет отправлено, как только он выберет строку композиции. Чтобы прослушивать событие `IMEEvent.IME_COMPOSITION`, необходимо добавить прослушиватель событий для статического свойства `ime` в классе System (`flash.system.System.ime.addEventListener(...)`), как показано в следующем примере.

```
var inputTxt:TextField;
var outputTxt:TextField;

inputTxt = new TextField();
inputTxt.type = TextFieldType.INPUT;
inputTxt.width = 200;
inputTxt.height = 18;
inputTxt.border = true;
inputTxt.background = true;
addChild(inputTxt);

outputTxt = new TextField();
outputTxt.autoSize = TextFieldAutoSize.LEFT;
outputTxt.y = 20;
addChild(outputTxt);

if (Capabilities.hasIME)
{
    IME.enabled = true;
    try
    {
        IME.conversionMode = IMEConversionMode.JAPANESE_HIRAGANA;
    }
    catch (error:Error)
    {
        outputTxt.text = "Unable to change IME.";
    }
    System.ime.addEventListener(IMEEvent.IME_COMPOSITION, imeCompositionHandler);
}
else
{
    outputTxt.text = "Please install IME and try again.";
}

function imeCompositionHandler(event:IMEEvent):void
{
    outputTxt.text = "you typed: " + event.text;
}
```

Предыдущий код создает два текстовых поля и добавляет их в список отображения. Первое текстовое поле, `inputTxt`, является полем ввода, в которое пользователь может ввести японский текст. Второе текстовое поле, `outputTxt`, является динамическим текстовым полем, в котором отображаются сообщения об ошибке и дублируется японская строка, введенная пользователем в поле `inputTxt`.

# Виртуальная клавиатура

**Flash Player 10.2 и более поздних версий, AIR 2.6 и более поздних версий**

Мобильные устройства, такие как телефоны и планшетные ПК, часто предусматривают виртуальную клавиатуру вместо физической. Классы в прикладном API-интерфейсе Flash позволяют:

- определять событие вызова и закрытия виртуальной клавиатуры;
- предотвращать вызов клавиатуры;

- определять часть в рабочей области, закрываемую виртуальной клавиатурой;
- создавать интерактивные объекты, которые вызывают клавиатуру, когда получают фокус. (Не поддерживается приложениями AIR в ОС iOS.)
- (Только для AIR.) Выключите поведение автоматического панорамирования, чтобы приложение могло изменять собственный экран при вызове клавиатуры.

## Управление поведением виртуальной клавиатуры

Среда выполнения автоматически открывает виртуальную клавиатуру, когда пользователь касается текстового поля или специально настроенного интерактивного объекта. При открытии клавиатуры среда выполнения следует стандартам платформы для панорамирования и изменения размера содержимого приложения, чтобы пользователь мог видеть вводимый им текст.

При открытии клавиатуры объект с фокусом последовательно отправляет следующие события:

Событие `softKeyboardActivating` — отправляется непосредственно перед появлением клавиатуры в рабочей области. Если вызвать метод `preventDefault()` из отправленного объекта события, виртуальная клавиатура не откроется.

Событие `softKeyboardActivate` — отправляется после завершения обработки события `softKeyboardActivating`. Когда объект с фокус отправляет это событие, в свойстве `softKeyboardRect` объекта Stage обновляются сведения о части рабочей области, занятой виртуальной клавиатурой. Это событие нельзя отменить.

***Примечание.*** *При изменении размера клавиатуры, например, когда пользователь изменяет ее тип, объект с фокусом отправляет второе событие softKeyboardActivate.*

Событие `softKeyboardDeactivate` — отправляется при закрытии виртуальной клавиатуры по любой причине. Это событие нельзя отменить.

В примере ниже в рабочую область добавляются два объекта TextField. Верхний объект TextField предотвращает появление клавиатуры при касании поля и закрывает клавиатуру, если она уже открыта. Нижний объект TextField демонстрирует поведение по умолчанию. Пример выводит отчёт о событиях экранной клавиатуры, отправленных обоими текстовыми полями.

```
package
{
import flash.display.Sprite;
import flash.text.TextField;
import flash.text.TextFieldType;
import flash.events.SoftKeyboardEvent;
public class SoftKeyboardEventExample extends Sprite
{
    private var tf1:TextField = new TextField();
    private var tf2:TextField = new TextField();

    public function SoftKeyboardEventExample()
    {
        tf1.width = this.stage.stageWidth;
        tf1.type = TextFieldType.INPUT;
        tf1.border = true;
        this.addChild( tf1 );

        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATING, preventSoftKe
yboard );
```

```
        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, preventSoftKe
yboard );
        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, preventSoftKeyboard
);

        tf2.border = true;
        tf2.type = TextFieldType.INPUT;
        tf2.width = this.stage.stageWidth;
        tf2.y = tf1.y + tf1.height + 30;
        this.addChild( tf2 );

        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATING, allowSoftKeyboard );
        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, allowSoftKeyboard );
        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, allowSoftKeyboard);
    }

    private function preventSoftKeyboard( event:SoftKeyboardEvent ):void
    {
            event.preventDefault();
            this.stage.focus = null; //close the keyboard, if raised
            trace( "tf1 dispatched: " + event.type + " -- " + event.triggerType );
    }
    private function allowSoftKeyboard( event:SoftKeyboardEvent )    :void
    {
            trace( "tf2 dispatched: " + event.type + " -- " + event.triggerType );
    }
}
}
```

## Добавление поддержки виртуальной клавиатуры для интерактивных объектов

**Flash Player 10.2 и более поздних версий, AIR 2.6 и более поздних версий (но не поддерживается для iOS)**

Обычно виртуальная клавиатура открывается только при касании объекта TextField. Можно настроить экземпляр класса InteractiveObject так, чтобы при получении им фокуса открывалась виртуальная клавиатура.

Чтобы настроить экземпляр InteractiveObject для вызова экранной клавиатуры, задайте его свойству `needsSoftKeyboard` значение `true`. Каждый раз при назначении объекта свойству focus рабочей области автоматически открывается виртуальная клавиатура. Кроме того, открыть клавиатуру можно вызовом метода `requestSoftKeyboard()` объекта InteractiveObject.

В следующем примере демонстрируется, как запрограммировать InteractiveObject в качестве текстового поля ввода. Класс TextInput, показанный в примере, задает свойство `needsSoftKeyboard` таким образом, чтобы при необходимости открывалась виртуальная клавиатура. Затем объект прослушивает события `keyDown`и вставляет введенный символ в поле.

В примере используется текстовый модуль Flash для добавления и отображения любого введенного текста и обрабатываются некоторые важные события. Для простоты в этом примере не реализуется полнофункциональное текстовое поле.

```
package  {
    import flash.geom.Rectangle;
    import flash.display.Sprite;
    import flash.text.engine.TextElement;
    import flash.text.engine.TextBlock;
    import flash.events.MouseEvent;
    import flash.events.FocusEvent;
    import flash.events.KeyboardEvent;
    import flash.text.engine.TextLine;
    import flash.text.engine.ElementFormat;
    import flash.events.Event;

    public class TextInput extends Sprite
    {

        public var text:String = " ";
        public  var textSize:Number = 24;
        public var textColor:uint = 0x000000;
        private var _bounds:Rectangle = new Rectangle( 0, 0, 100, textSize );
        private var textElement: TextElement;
        private var textBlock:TextBlock = new  TextBlock();

        public function TextInput( text:String = "" )
        {
            this.text = text;
            this.scrollRect = _bounds;
            this.focusRect= false;

            //Enable keyboard support
            this.needsSoftKeyboard = true;
            this.addEventListener(MouseEvent.MOUSE_DOWN, onSelect);
            this.addEventListener(FocusEvent.FOCUS_IN, onFocusIn);
            this.addEventListener(FocusEvent.FOCUS_OUT, onFocusOut);

            //Setup text engine
            textElement = new TextElement( text, new ElementFormat( null, textSize, textColor ) );
            textBlock.content = textElement;
            var firstLine:TextLine = textBlock.createTextLine( null, _bounds.width - 8 );
            firstLine.x = 4;
            firstLine.y = 4 + firstLine.totalHeight;
            this.addChild( firstLine );

        }

        private function onSelect( event:MouseEvent ):void
        {
            stage.focus = this;
        }
        private function onFocusIn( event:FocusEvent ):void
        {
            this.addEventListener( KeyboardEvent.KEY_DOWN, onKey );
        }

        private function onFocusOut( event:FocusEvent ):void
        {
            this.removeEventListener( KeyboardEvent.KEY_UP, onKey );
        }
```

```
        private function onKey( event:KeyboardEvent ):void
        {
            textElement.replaceText( textElement.text.length, textElement.text.length,
String.fromCharCode( event.charCode ) );
            updateText();
        }
        public function set bounds( newBounds:Rectangle ):void
        {
            _bounds = newBounds.clone();
            drawBackground();
            updateText();
            this.scrollRect = _bounds;

            //force update to focus rect, if needed
            if( this.stage!= null && this.focusRect && this.stage.focus == this )
                this.stage.focus = this;
        }

        private function updateText():void
        {
            //clear text lines
            while( this.numChildren > 0 ) this.removeChildAt( 0 );

            //and recreate them
            var textLine:TextLine = textBlock.createTextLine( null, _bounds.width - 8);
            while ( textLine)
            {
                textLine.x = 4;
                if( textLine.previousLine != null )
                {
                    textLine.y = textLine.previousLine.y +
                                 textLine.previousLine.totalHeight + 2;
                }
                              else
                {
                    textLine.y = 4 + textLine.totalHeight;
                }
                this.addChild(textLine);
                textLine = textBlock.createTextLine(textLine, _bounds.width - 8 );
            }
        }

        private function drawBackground():void
        {
            //draw background and border for the field
            this.graphics.clear();
            this.graphics.beginFill( 0xededed );
            this.graphics.lineStyle( 1, 0x000000 );
            this.graphics.drawRect( _bounds.x + 2, _bounds.y + 2, _bounds.width - 4,
_bounds.height - 4);
            this.graphics.endFill();
        }
    }
}
```

Приведенный ниже основной класс приложения демонстрирует использование класса TextInput и управление макетом приложения при вызове клавиатуры или изменении ориентации устройства. Основной класс создает объект TextInput и устанавливает его границы для заполнения рабочей области. Класс корректирует размер объекта TextInput при вызове экранной клавиатуры или изменения размера рабочей области. Класс прослушивает события экранной клавиатуры, отправляемые объектом TextInput, и события изменения размера, отправляемые рабочей областью. Независимо от причины события приложение определяет видимую часть рабочей области и изменяет размер элемента управления вводом, чтобы заполнить ее. Естественно, в реальном приложении потребуется более сложный алгоритм макета.

```
package  {

    import flash.display.MovieClip;
    import flash.events.SoftKeyboardEvent;
    import flash.geom.Rectangle;
    import flash.events.Event;
    import flash.display.StageScaleMode;
    import flash.display.StageAlign;

    public class CustomTextField extends MovieClip {

        private var customField:TextInput = new TextInput("Input text: ");

        public function CustomTextField() {
            this.stage.scaleMode = StageScaleMode.NO_SCALE;
            this.stage.align = StageAlign.TOP_LEFT;
            this.addChild( customField );
            customField.bounds = new Rectangle( 0, 0, this.stage.stageWidth,
this.stage.stageHeight );

            //track soft keyboard and stage resize events
            customField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE,
onDisplayAreaChange );
            customField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE,
onDisplayAreaChange );
            this.stage.addEventListener( Event.RESIZE, onDisplayAreaChange );
        }

        private function onDisplayAreaChange( event:Event ):void
        {
            //Fill the stage if possible, but avoid the area covered by a keyboard
            var desiredBounds = new Rectangle( 0, 0, this.stage.stageWidth,
this.stage.stageHeight );
            if( this.stage.stageHeight - this.stage.softKeyboardRect.height <
desiredBounds.height )
                desiredBounds.height = this.stage.stageHeight -
this.stage.softKeyboardRect.height;

            customField.bounds = desiredBounds;
        }
    }
}
```

***Примечание.*** *Рабочая область отправляет события изменения размеров только в ответ на изменение ориентации, когда значение свойства* `scaleMode` *получает значение* `noScale`*. В других режимах размеры рабочей области не изменяются, а содержимое масштабируется с целью компенсации.*

## Обработка изменения отображения приложения

**AIR 2.6 и более поздних версий**

В AIR можно отключить поведение панорамирования и изменения размеров по умолчанию, связанное с вызовом экранной клавиатуры, задав элементу `softKeyboardBehavior` в дескрипторе приложения значение `none`.

```
<softKeyboardBehavior>none</softKeyboardBehavior>
```

При выключении автоматического поведения приложение должно будет соответствующим образом корректировать свой экран. Событие softKeyboardActivate отправляется, когда открывается клавиатура. При отправке события `softKeyboardActivate` свойство `softKeyboardRect` рабочей области содержит размеры области, занятой открытой клавиатурой. Используйте эти размеры для перемещения или изменения размера содержимого, чтобы оно правильно отображалось, когда открывается клавиатура для ввода текста. (После закрытия клавиатуры все размеры прямоугольника softKeyboardRect равны нулю.)

При закрытии клавиатуры отправляется событие `softKeyboardDeactivate`, и можно восстановить обычное отображение приложения.

```
package  {
    import flash.display.MovieClip;
    import flash.events.SoftKeyboardEvent;
    import flash.events.Event;
    import flash.display.StageScaleMode;
    import flash.display.StageAlign;
    import flash.display.InteractiveObject;
    import flash.text.TextFieldType;
    import flash.text.TextField;

    public class PanningExample extends MovieClip {

        private var textField:TextField = new TextField();

        public function PanningExample() {
            this.stage.scaleMode = StageScaleMode.NO_SCALE;
            this.stage.align = StageAlign.TOP_LEFT;

            textField.y = this.stage.stageHeight - 201;
            textField.width = this.stage.stageWidth;
            textField.height = 200;
            textField.type = TextFieldType.INPUT;
            textField.border = true;
            textField.wordWrap = true;
            textField.multiline = true;

            this.addChild( textField );

            //track soft keyboard and stage resize events
            textField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE,
onKeyboardChange );
            textField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE,
onKeyboardChange );
            this.stage.addEventListener( Event.RESIZE, onDisplayAreaChange );
        }

        private function onDisplayAreaChange( event:Event ):void
```

```
        {
            textField.y = this.stage.stageHeight - 201;
            textField.width = this.stage.stageWidth;
        }

        private function onKeyboardChange( event:SoftKeyboardEvent ):void
        {
            var field:InteractiveObject = textField;
            var offset:int = 0;

            //if the softkeyboard is open and the field is at least partially covered
            if( (this.stage.softKeyboardRect.y != 0) && (field.y + field.height >
this.stage.softKeyboardRect.y) )
                offset = field.y + field.height - this.stage.softKeyboardRect.y;

            //but don't push the top of the field above the top of the screen
            if( field.y - offset < 0 ) offset += field.y - offset;

            this.y = -offset;
        }
    }
}
```

***Примечание.*** *В Android возможны условия, в том числе при работе в полноэкранном режиме, когда точные размеры клавиатуры невозможно получить из операционной системы. В этих случаях используется предполагаемый размер. Кроме того, в альбомной ориентации для ввода текста используется собственная полноэкранная клавиатура IME (редактора метода ввода). Эта клавиатура IME имеет встроенное текстовое поле ввода и закрывает всю рабочую область. Нет возможности отображения альбомной клавиатуры, которая занимает не весь экран.*

# Глава 31. Действия мыши

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Посредством захвата действий мыши и реагирования на них ваше приложение может создавать интерактивность. Обратите внимание, что для понимания этого раздела вы должны быть знакомы с моделью событий ActionScript 3.0. Дополнительные сведения см. в разделе ««Обработка событий» на странице 134».

Сведения об обнаружении доступности поддержки мыши (например, постоянный курсор, перо или сенсорный ввод) во время выполнения см. в разделе ««Обнаружение типов ввода» на странице 599».

### Дополнительные разделы справки

flash.ui.Mouse

flash.events.MouseEvent

«Ввод с помощью прикосновения, нескольких прикосновений или жестов» на странице 624

## Захват действий мыши

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При щелчках мыши создаются события мыши, которые можно использовать для вызова интерактивной функциональности. Можно добавить в рабочую область прослушиватель событий, чтобы улавливать события, происходящие в любом месте SWF-файла. Также можно добавить прослушиватели событий к объектам в рабочей области, которые являются наследниками InteractiveObject (например, Sprite или MovieClip). Эти прослушиватели вызываются при щелчке по объекту.

Как и в случае с событиями клавиатуры, события мыши передаются вверх по иерархии. В следующем примере объект `square` является дочерним объектом объекта Stage, поэтому при щелчке мышью событие отправляется как спрайтом `square`, так и объектом Stage:

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0xFF0000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.addEventListener(MouseEvent.CLICK, reportClick);
square.x =
square.y = 50;
addChild(square);

stage.addEventListener(MouseEvent.CLICK, reportClick);

function reportClick(event:MouseEvent):void
{
    trace(event.currentTarget.toString() + " dispatches MouseEvent. Local coords [" +
event.localX + "," + event.localY + "] Stage coords [" + event.stageX + "," + event.stageY + "]");
}
```

В предыдущем примере событие мыши содержит информацию о положении щелчка. Свойства `localX` и `localY` содержат положение щелчка на низшем дочернем объекте в цепи отображения. Например, при щелчке в левом верхнем углу объекта `square` возвращаются локальные координаты [0,0], потому что это регистрационная точка `square`. Также свойства `stageX` и `stageY` могут указывать на глобальные координаты щелчка в рабочей области. Тот же щелчок возвратит для этих координат значение [50,50], потому что объект `square` переместился. Могут пригодиться обе пары координат в зависимости от того, каким образом вы хотите отвечать на действия пользователя.

***Примечание.*** *В полноэкранном режиме можно сделать такие настройки, чтобы в приложении использовалась блокировка мыши. При блокировке мыши курсор отключается, и включается несвязанное движение мыши. Дополнительные сведения см. в разделе ««Работа в полноэкранном режиме» на странице 179».*

Объект MouseEvent также содержит логические свойства `altKey`, `ctrlKey` и `shiftKey`. Эти свойства можно использовать для проверки того, нажаты ли во время щелчка клавиши Alt, Ctrl или Shift.

## Перетаскивание спрайтов в рабочей области

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Можно предоставить пользователям возможность перетаскивать объекты Sprite по рабочей области с помощью метода `startDrag()` класса Sprite. Следующий код иллюстрирует это:

```
import flash.display.Sprite;
import flash.events.MouseEvent;

var circle:Sprite = new Sprite();
circle.graphics.beginFill(0xFFCC00);
circle.graphics.drawCircle(0, 0, 40);

var target1:Sprite = new Sprite();
target1.graphics.beginFill(0xCCFF00);
target1.graphics.drawRect(0, 0, 100, 100);
target1.name = "target1";

var target2:Sprite = new Sprite();
target2.graphics.beginFill(0xCCFF00);
target2.graphics.drawRect(0, 200, 100, 100);
target2.name = "target2";

addChild(target1);
addChild(target2);
addChild(circle);

circle.addEventListener(MouseEvent.MOUSE_DOWN, mouseDown)

function mouseDown(event:MouseEvent):void
{
    circle.startDrag();
}
circle.addEventListener(MouseEvent.MOUSE_UP, mouseReleased);

function mouseReleased(event:MouseEvent):void
{
    circle.stopDrag();
    trace(circle.dropTarget.name);
}
```

Дополнительные сведения см. в разделе о создании взаимодействия путем перетаскивания с помощью мыши в разделе ««Изменение положения» на странице 187».

**Перетаскивание в AIR**

В Adobe AIR можно включить поддержку перетаскивания, чтобы пользователи могли перетаскивать данные в приложение и за его пределы. Дополнительные сведения см. в разделе ««Перетаскивание в AIR» на странице 654».

---

## Персонализация указателя мыши

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Указатель мыши (курсор) для любого объекта в рабочей области можно скрыть или заменить. Чтобы скрыть указатель мыши, нужно вызвать метод `Mouse.hide()`. Можно изменить курсор, вызвав метод `Mouse.hide()`, прослушивающий рабочую область на предмет события `MouseEvent.MOUSE_MOVE`, и указав в качестве координат экранного объекта (вашего собственного курсора) свойства события `stageX` и `stageY`. В примере ниже показано выполнение этой задачи:

```
var cursor:Sprite = new Sprite();
cursor.graphics.beginFill(0x000000);
cursor.graphics.drawCircle(0,0,20);
cursor.graphics.endFill();
addChild(cursor);

stage.addEventListener(MouseEvent.MOUSE_MOVE,redrawCursor);
Mouse.hide();

function redrawCursor(event:MouseEvent):void
{
    cursor.x = event.stageX;
    cursor.y = event.stageY;
}
```

# Пример действий мыши: WordSearch

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В этом примере показано взаимодействие с пользователем с помощью событий мыши. У пользователя есть расчерченное поле со случайным набором букв. Задача — найти как можно больше слов, записанных по горизонтали или по вертикали, при этом нельзя дважды использовать одну букву. В этом примере демонстрируются следующие функции языка ActionScript 3.0:

- динамическое построение сетки компонентов;
- реагирование на события мыши;
- ведение счета на основе взаимодействия с пользователем.

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения WordSearch находятся в папке Samples/WordSearch. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| WordSearch.as | Класс, обеспечивающий основную функциональность приложения. |
| WordSearch.fla<br>или<br>WordSearch.mxml | Основной файл приложения для Flex (MXML) или Flash (FLA). |
| dictionary.txt | Файл, используемый для проверки орфографии и «стоимости» каждого слова. |

## Загрузка словаря

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для создания игры на нахождение слов необходим словарь. В примере используется текстовый файл dictionary.txt, содержащий список слов, разделенных переводами каретки. После создания массива `words` функция `loadDictionary()` запрашивает этот файл, который после успешной загрузки становится длинной строкой. Эту строку можно разобрать на массив слов с помощью метода `split()`, отделяя слова друг от друга по переводу каретки (код символа 10) или новой строке (код символа 13). Разбор осуществляется в функции `dictionaryLoaded()`:

```
words = dictionaryText.split(String.fromCharCode(13, 10));
```

## Создание пользовательского интерфейса

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

После сохранения слов можно создать пользовательский интерфейс. Создайте два экземпляра Button: один для подтверждения слова, а другой — для очистки слова, над которым пользователь в данный момент работает. В каждом случае нужно отвечать на команды пользователя путем прослушивания события `MouseEvent.CLICK`, которое транслирует кнопка, и вызова функции. В функции `setupUI()` этот код создает прослушиватели на двух кнопках:

```
submitWordButton.addEventListener(MouseEvent.CLICK,submitWord);
clearWordButton.addEventListener(MouseEvent.CLICK,clearWord);
```

## Генерация игрового поля

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Игровое поле — это сетка со случайными буквами. Функция `generateBoard()` создает двухмерную сетку, размещая один цикл внутри другого. Первый цикл приращивает ряды, а второй — общее количество столбцов в ряду. Каждая ячейка, формируемая этими рядами и столбцами, содержит кнопку с буквой.

```
private function generateBoard(startX:Number, startY:Number, totalRows:Number,
totalCols:Number, buttonSize:Number):void
{
    buttons = new Array();
    var colCounter:uint;
    var rowCounter:uint;
    for (rowCounter = 0; rowCounter < totalRows; rowCounter++)
    {
        for (colCounter = 0; colCounter < totalCols; colCounter++)
        {
            var b:Button = new Button();
            b.x = startX + (colCounter*buttonSize);
            b.y = startY + (rowCounter*buttonSize);
            b.addEventListener(MouseEvent.CLICK, letterClicked);
            b.label = getRandomLetter().toUpperCase();
            b.setSize(buttonSize,buttonSize);
            b.name = "buttonRow"+rowCounter+"Col"+colCounter;
            addChild(b);

            buttons.push(b);
        }
    }
}
```

Хотя прослушиватель для событий `MouseEvent.CLICK` добавляется только на одну строку, так как он находится в цикле `for`, он все равно назначается каждому экземпляру Button. Кроме того, каждой кнопке назначается имя на основе ее положения в столбце и в ряду, что облегчает работу с каждой из кнопок в коде.

## Составление слов в ответ на действия пользователя

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Слова можно составлять, выбирая буквы, стоящие рядом по вертикали и по горизонтали, но одну и ту же букву нельзя использовать дважды. Каждый щелчок вызывает событие мыши: в этот момент программа проверяет, чтобы слово, которое составил пользователь, должным образом продолжало буквы, на которых пользователь щелкнул ранее. Если это не так, предыдущее слово удаляется и начинается новое. Эта проверка осуществляется методом `isLegalContinuation()`.

```
private function isLegalContinuation(prevButton:Button, currButton:Button):Boolean
{
    var currButtonRow:Number = Number(currButton.name.charAt(currButton.name. indexOf("Row") +
3));
    var currButtonCol:Number = Number(currButton.name.charAt(currButton.name.indexOf("Col") +
3));
    var prevButtonRow:Number = Number(prevButton.name.charAt(prevButton.name.indexOf("Row") +
3));
    var prevButtonCol:Number = Number(prevButton.name.charAt(prevButton.name.indexOf("Col") +
3));

    return ((prevButtonCol == currButtonCol && Math.abs(prevButtonRow - currButtonRow) <= 1) ||
             (prevButtonRow == currButtonRow && Math.abs(prevButtonCol - currButtonCol) <= 1));
}
```

Методы `charAt()` и `indexOf()` класса String возвращают значения рядов и столбцов текущей и предыдущей букв. Метод `isLegalContinuation()` возвращает значение `true`, если текущая буква находится в том же ряду или столбце, что и предыдущая, при этом положение в столбце или ряду увеличивается на единицу. Если вы хотите, чтобы пользователь мог составлять слова и по диагонали, удалите проверку последовательности рядов и столбцов. Итоговая строка будет выглядеть так:

```
return (Math.abs(prevButtonRow - currButtonRow) <= 1) && Math.abs(prevButtonCol -
currButtonCol) <= 1));
```

## Проверка подтверждения слов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Остался последний шаг в написании кода к игре: необходим механизм для проверки подтверждения слов и ведения счета. Метод `searchForWord()` помогает решить обе эти задачи:

```
private function searchForWord(str:String):Number
{
    if (words && str)
    {
        var i:uint = 0
        for (i = 0; i < words.length; i++)
        {
            var thisWord:String = words[i];
            if (str == words[i])
            {
                return i;
            }
        }
        return -1;
    }
    else
    {
        trace("WARNING: cannot find words, or string supplied is null");
    }
    return -1;
}
```

Функция проходит по всем словам в словаре, при этом цикл непрерывно повторяется. Если слово, составленное пользователем, совпадает со словом из словаря, возвращается его положение в словаре. Метод `submitWord()` проверяет ответ и обновляет счет, если положение действительно.

## Персонализация

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В начале класса есть несколько констант. Можно модифицировать игру, изменив эти переменные. Например, можно изменить количество времени, которое дается пользователю, исправив переменную `TOTAL_TIME`. Также можно немного увеличить значение переменной `PERCENT_VOWELS`, чтобы пользователю было проще находить слова.

# Глава 32. Ввод с помощью прикосновения, нескольких прикосновений или жестов

**Flash Player 10.1 и более поздних версий, Adobe AIR 2 и более поздних версий**

Функции обработки события прикосновения Flash Platform включают ввод в одной точке или нескольких точках в устройствах, поддерживающих технологию сенсорного ввода. Кроме того, среды выполнения Flash обрабатывают события, объединяющие несколько точек прикосновения для создания жеста. Другими словами, среды выполнения Flash интерпретируют два типа ввода:

**Прикосновение** ввод с помощью инструмента ввода в одной точке, например пальца, пера или другого инструмента, в устройстве, поддерживающем сенсорный ввод. В некоторых устройствах поддерживается несколько точек прикосновения пальцем или пером.

**Multitouch** ввод путем прикосновения одновременно в нескольких точках.

**Жест** Ввод, интерпретируемый устройством или операционной системой в ответ на одно или несколько событий прикосновения. Например, пользователь выполняет вращательное движение одновременно двумя пальцами, а устройство или операционная система интерпретирует этот ввод прикосновением как жест поворота. Некоторые жесты выполняются одним пальцем или в одной точке прикосновения, а для других требуется несколько точек прикосновения. Тип жеста, назначаемый вводу, устанавливается устройством или операционной системой.

Ввод с помощью прикосновения и жеста может выполняться в нескольких точках в зависимости от пользовательского устройства. ActionScript предоставляет API-интерфейс для обработки событий прикосновения, событий жестов и отдельных событий прикосновения в случае ввода в нескольких точках.

***Примечание.*** *В зависимости от компьютерного устройства и операционной системы прослушивание событий прикосновения и жестов может потреблять значительное количество вычислительных ресурсов (аналогично визуализации нескольких кадров в секунду). Зачастую лучше использовать события мыши, даже если дополнительная функциональность, предоставляемая прикосновениями или жестами, не требуется. При использовании событий прикосновения или жестов рассмотрите возможность снижения количества происходящих графических изменений, особенно когда такие события отправляются быстро, например при выполнении операций панорамирования, поворота и масштабирования. Например, можно остановить анимацию для компонента, когда пользователь изменяет его размер с помощью жеста масштабирования.*

### Дополнительные разделы справки

flash.ui.Multitouch

flash.events.TouchEvent

flash.events.GestureEvent

flash.events.TransformGestureEvent

flash.events.GesturePhase

flash.events.PressAndTapGestureEvent

Пол Трани (Paul Trani): События прикосновения и жесты на мобильных устройствах

Майк Джонс (Mike Jones): Виртуальные игровые контроллеры

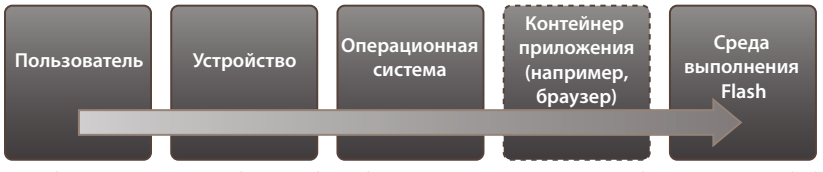
# Основные сведения о вводе прикосновением

**Flash Player 10.1 и более поздних версий, Adobe AIR 2 и более поздних версий**

Когда Flash Platform выполняется в среде, поддерживающей ввод прикосновением, экземпляры InteractiveObject могут прослушивать события прикосновения и вызывать обработчики. Как правило, события прикосновения, нескольких прикосновений и жестов обрабатываются так же, как и другие события в ActionScript (основные сведения об обработке событий в коде ActionScript см. в разделе ««Обработка событий» на странице 134»).

Однако, чтобы среда выполнения Flash могла интерпретировать прикосновение или жест, она должна выполняться в аппаратной и программной средах, поддерживающих ввод прикосновением или несколькими прикосновениями. См. раздел ««Обнаружение типов ввода» на странице 599» с таблицей, в которой сравниваются различные типы сенсорных экранов. Помимо этого, если среда выполнения работает в приложении контейнера (например, браузере), контейнер передает ввод в среду выполнения. В некоторых случаях текущая среда оборудования и операционной системы поддерживают технологию Multitouch, но браузер, содержащий среду выполнения Flash, интерпретирует ввод и не передает его в среду выполнения. Или он может полностью игнорировать ввод.

На следующей диаграмме показана последовательность передачи входных данных от пользователя в среду выполнения:




*Последовательность передачи входных данных от пользователя в среду выполнения Flash Platform*

К счастью, API-интерфейс ActionScript для разработки приложений, поддерживающих сенсорный ввод, включает классы, методы и свойства для определения наличия поддержки ввода прикосновением или несколькими прикосновениями в среде выполнения. API-интерфейс, используемый для определения наличия поддержки сенсорного ввода, является «API-интерфейсом обнаружения» для обработки событий прикосновения.

### Важные понятия и термины

В следующем справочном списке содержатся важные термины, необходимые для написания приложений, поддерживающих обработку событий прикосновения:

**API-интерфейс обнаружения** Методы и свойства, используемые для проверки наличия в среде выполнения поддержки событий прикосновения и различных режимов ввода.

**Событие прикосновения** Действие ввода, выполняемое в устройстве, поддерживающем сенсорный ввод, с использованием одной точки контакта.

**Точка прикосновения** Точка контакта для события одного прикосновения. Даже если устройство не поддерживает ввод с помощью жеста, оно может поддерживать одновременно несколько точек прикосновения.

**Последовательность прикосновения** Набор событий, представляющий жизненный цикл одного прикосновения. Эти события включают одно начальное событие, событие отсутствия перемещений или нескольких перемещений и одно завершающее событие.

**Событие Multitouch** Действие ввода, выполняемое в устройстве, поддерживающем сенсорный ввод, с использованием нескольких точек контакта (например, несколькими пальцами).

**Событие жеста** Действие ввода, выполняемое в устройстве, поддерживающем сенсорный ввод и отслеживающем набор сложных движений. Например, одним из жестов является прикосновение к экрану двумя пальцами и одновременное перемещение пальцев по периметру абстрактной окружности для указания поворота.

**Фазы** Отдельные точки времени в потоке событий (например, начало и конец).

**Перо** Инструмент для взаимодействия с сенсорным экраном. Перо является более точным инструментом указания, чем палец. Некоторые устройства распознают ввод только с помощью определенных типов перьев. Устройства, распознающие ввод пером, могут не распознавать несколько одновременных точек контакта или прикосновение пальцем.

**Удерживание и нажатие** Определенный тип жеста для ввода несколькими прикосновениями, когда пользователь нажимает пальцем сенсорный экран устройства, а затем другим пальцем или инструментом указания нажимает кнопки. Этот жест часто используется для имитации щелчка правой кнопкой мыши в приложениях Multitouch.

## Структура API-интерфейса для ввода прикосновением

API-интерфейс для ввода прикосновением в среде ActionScript разработан с учетом того факта, что обработка ввода прикосновением зависит от аппаратной и программной среды, в которой выполняется Flash. API-интерфейс ввода прикосновением в основном удовлетворяет три потребности при разработке приложений, поддерживающих сенсорный ввод: обнаружения, события и фазы. Управляйте этим API-интерфейсом для создания приложений с предсказуемым приспосабливаемым поведением; даже если целевое устройство известно при разработке приложения.

### Обнаружение

API-интерфейс обнаружения позволяет тестировать аппаратную и программную среду во время выполнения. Значения, заполненные средой выполнения, определяют ввод прикосновением, доступный среде выполнения Flash в текущем контексте. Кроме того, используйте набор свойств и методов обнаружения для задания реакции приложения на события мыши (вместо событий прикосновения в случае, если ввод прикосновением не поддерживается средой). Дополнительные сведения см. в разделе ««Обнаружение поддержки прикосновения» на странице 627».

### События

ActionScript управляет событиями ввода прикосновением с помощью прослушивателей и обработчиков событий, как и в случае с другими событиями. Однако при этом также должна учитываться обработка событий ввода прикосновением:

- Прикосновение может интерпретироваться устройством или операционной системой несколькими способами: как последовательность прикосновений или совместно как жест.

- При одном прикосновении (пальцем, пером или инструментом указания) к устройству, поддерживающему сенсорный ввод, всегда также отправляется событие мыши. Событие мыши можно обработать с использованием типов событий в классе MouseEvent. Или можно создать приложение так, чтобы оно реагировало только на события прикосновения. Либо можно создать приложение, реагирующее на оба типа событий.
- Приложение может реагировать на несколько одновременных событий прикосновения и обрабатывать каждое событие отдельно.

Обычно API-интерфейс обнаружения используется для условной обработки событий в приложении и управления способом обработки. Как только приложение получит данные о среде выполнения, оно может вызвать подходящий обработчик или создать подходящий объект события при взаимодействии пользователя с приложением. Или приложение может передать сведения о том, что обработка определенного типа ввода невозможна в текущей среде, и вывести для пользователя альтернативное решение или инструкции. Дополнительные сведения см. в разделах ««Обработка событий нажатия» на странице 628» и ««Обработка событий жеста» на странице 633».

### Фазы

Для приложений, поддерживающих ввод одним или несколькими прикосновениями, в объектах событий прикосновения содержатся свойства для отслеживания фаз взаимодействия с пользователем. Напишите код ActionScript для обработки фаз как начальную фазу, фазу обновления или конечную фазу пользовательского ввода, чтобы предоставить пользователю отзыв. Реакция на фазы событий должна быть такой, чтобы визуальные объекты изменялись, когда пользователь прикасается к точке и перемещает точку прикосновения на экране. Или используйте фазы для отслеживания определенных свойств жеста при его выполнении.

Для событий с точкой прикосновения отслеживайте продолжительность нажатия пользователем определенного интерактивного объекта. Приложение может отслеживать фазы нескольких одновременных точек прикосновения по отдельности и обрабатывать их соответственно.

Для жеста интерпретируйте определенную информацию об изменении. Отслеживайте координаты одной или нескольких точек контакта при их перемещении на экране.

## Обнаружение поддержки прикосновения

**Flash Player 10.1 и более поздних версий, Adobe AIR 2 и более поздних версий**

Используйте свойства класса Multitouch для задания области ввода прикосновением, обрабатываемой приложением. Затем протестируйте среду, чтобы убедиться в наличии поддержки событий, обрабатываемых кодом ActionScript. В частности, сначала установите тип ввода прикосновением для приложения. Возможные варианты: точка прикосновения, жест или отсутствие поддержки (все события ввода прикосновением интерпретируются как нажатия кнопки мыши и используются только обработчики событий мыши). Затем используйте свойства и методы класса Multitouch, чтобы убедиться в том, что среда выполнения работает в среде, поддерживающей сенсорный ввод, который требуется в приложении. Проверьте в среде наличие поддержки трех типов сенсорного ввода (например, поддержку интерпретации жестов) и действуйте соответственно.

***Примечание.*** *Свойства класса Multitouch являются статическими и не принадлежат экземплярам какого-либо класса. Используйте их с синтаксисом «Multitouch.свойство», например:*

```
var touchSupport:Boolean = Multitouch.supportsTouchEvents;
```

## Задание типа ввода

В среде выполнения Flash должен быть известен тип интерпретируемого сенсорного ввода, поскольку событие прикосновение может иметь множество элементов или фаз. Отправляет ли среда выполнения событие прикосновения, если палец просто касается сенсорного экрана? Или она ждет жеста? Или отслеживает ли среда выполнения прикосновение как событие нажатия кнопки мыши? Приложение, поддерживающее сенсорный ввод, должно устанавливать тип события прикосновения, обрабатываемого ею для среды выполнения Flash. Используйте свойство `Multitouch.inputMode` для установки типа сенсорного ввода для среды выполнения. Возможен один из трех вариантов режима ввода:

**Нет** Специальная обработка для события прикосновения не предоставляется. Задайте: `Multitouch.inputMode=MultitouchInputMode.NONE` и используйте класс MouseEvent для обработки ввода.

**Точки одного прикосновения** Все события сенсорного ввода интерпретируются по отдельности, и все точки прикосновения могут быть отслежены и обработаны. Задайте: `Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT` и используйте класс TouchEvent для обработки ввода.

**Ввод жестом** Устройство или операционная система интерпретирует ввод как сложную форму движения пальца по экрану. Устройство или операционная система обобщенно назначает движение одному событию ввода жестом. Задайте: `Multitouch.inputMode=MultitouchInputMode.GESTURE` и используйте классы TransformGestureEvent, PressAndTapGestureEvent или GestureEvent для обработки ввода.

См. раздел ««Обработка событий нажатия» на странице 628» с примером, в котором свойство `Multitouch.inputMode` используется для задания типа ввода перед обработкой события прикосновения.

## Проверка наличия поддержки сенсорного ввода

Другие свойства класса Multitouch предоставляют значения для точной настройки приложения в соответствии с текущей поддержкой прикосновений в среде. Среда выполнения Flash заполняет значения для ряда допустимых точек одновременного прикосновения или доступных жестов. Если среда выполнения является средой, в которой не поддерживается обработка событий прикосновения, необходимая в приложении, предоставьте пользователю альтернативное решение. Например, обеспечьте обработку событий мыши или предоставьте информацию о функциях, доступных в текущей среде.

Можно также использовать API-интерфейс для поддержки клавиатуры, прикосновения и мыши. См. раздел ««Обнаружение типов ввода» на странице 599».

Дополнительные сведения о проверке совместимости см. в разделе ««Устранение неполадок» на странице 637».

# Обработка событий нажатия

**Flash Player 10.1 и более поздних версий, Adobe AIR 2 и более поздних версий**

В среде ActionScript основные события прикосновения обрабатываются так же, как и другие события, например события мыши. Можно прослушать набор событий прикосновения, заданных константами типов событий в классе TouchEvent.

***Примечание.*** *При вводе с несколькими точками прикосновения (например, касание устройства несколькими пальцами) первая точка контакта отправляет событие мыши и событие касания.*

Обработка основного события прикосновения:

1 Задайте в приложении обработку событий прикосновения, установив для свойства `flash.ui.Multitouch.inputMode` значение `MultitouchInputMode.TOUCH_POINT`.

2 Назначьте прослушиватель событий экземпляру класса, который наследует свойства класса InteractiveObject, например Sprite или TextField.

3 Укажите тип обрабатываемого события прикосновения.

4 Вызовите функцию обработчика события для выполнения действия в ответ на событие.

Например, следующий код позволяет отобразить сообщение, когда пользователь прикасается к квадрату, нарисованному в объекте mySprite, на сенсорном экране:

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taphandler);

function taphandler(evt:TouchEvent): void {
    myTextField.text = "I've been tapped";
    myTextField.y = 50;
    addChild(myTextField);
}
```

## Свойства событий нажатия

Когда происходит событие, создается объект события. Объект TouchEvent содержит информацию о расположении и условиях события прикосновения. Свойства объекта события можно использовать для получения этой информации.

Например, следующий код создает объект TouchEvent *evt*, а затем отображает свойство `stageX` объекта события (координату X точки рабочей области, в которой произошло прикосновение) в текстовом поле:

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taphandler);

function taphandler(evt:TouchEvent): void {
myTextField.text = evt.stageX.toString;
myTextField.y = 50;
addChild(myTextField);
}
```

См. описание класса TouchEvent для получения свойств, доступных в объекте события.

***Примечание.*** *Не все свойства TouchEvent поддерживаются в средах выполнения. Например, не все устройства с сенсорным экраном поддерживают измерение силы нажатия на сенсорный экран. Поэтому свойство* `TouchEvent.pressure` *не поддерживается на этих устройствах. Попробуйте проверить наличие поддержки определенного свойства для обеспечения правильной работы приложения и см. раздел ««Устранение неполадок» на странице 637» для получения дополнительных сведений.*

## Фазы события прикосновения

Отслеживайте различные этапы событий прикосновения в объекте InteractiveObject и за его пределами, как и в случае с событиями мыши. Кроме того, отслеживайте начальную, промежуточную и конечную фазу события прикосновения. Класс TouchEvent предоставляет значения для обработки событий `touchBegin`, `touchMove` и `touchEnd`.

Например, события `touchBegin`, `touchMove` и `touchEnd` можно использовать для предоставления пользователю визуального отклика при нажатии и перемещении экранного объекта:

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);
var myTextField:TextField = new TextField();
myTextField.width = 200;
myTextField.height = 20;
addChild(myTextField);

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
stage.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
stage.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);
function onTouchBegin(event:TouchEvent) {
    myTextField.text = "touch begin" + event.touchPointID;
}
function onTouchMove(event:TouchEvent) {
    myTextField.text = "touch move" + event.touchPointID;
}
function onTouchEnd(event:TouchEvent) {
    myTextField.text = "touch end" + event.touchPointID;
}
```

***Примечание.*** *Начальный прослушиватель событий прикосновения подключается к объекту mySprite, но прослушиватели для перемещения и завершения прикосновения не подключаются. Если палец или инструмент указания перемещается перед экранным объектом, в рабочей области продолжается прослушивание события прикосновения.*

### Идентификатор точки прикосновения

Свойство `TouchEvent.touchPointID` является обязательным компонентом при написании приложений. Оно обеспечивает реакцию на сенсорный ввод. Среда выполнения Flash назначает каждой точке прикосновения уникальное значение `touchPointID`. Когда приложение реагирует на фазы или перемещение при сенсорном вводе, проверьте значение `touchPointID` перед обработкой события. В методах перетаскивания при сенсорном вводе в классе Sprite свойство `touchPointID` используется в качестве параметра, поэтому

обрабатывается подходящий экземпляр ввода. Свойство `touchPointID` обеспечивает реакцию обработчика событий на правильную точку ввода. В противном случае обработчик события реагирует на любые экземпляры типов событий прикосновения (например, все события `touchMove`) в устройстве, создавая непредвиденное поведение. Свойство играет особенно важную роль, когда пользователь перетаскивает объекты.

Используйте свойство `touchPointID` для управления всей последовательностью прикосновений. Последовательность прикосновений включает одно событие `touchBegin`, ни одного или несколько событий `touchMove` и одно событие `touchEnd`, для каждого из которых задано одинаковое значение `touchPointID`.

В следующем примере вводится переменная *touchMoveID* для проверки правильного значения `touchPointID` перед ответом на событие перемещения точки касания. В противном случае другой ввод с помощью касания также запускает обработчика события. Обратите внимание, что прослушиватели фаз перемещения и завершения закреплены за рабочей областью, а не экранным объектом. Рабочая область прослушивает фазы перемещения и завершения, когда прикосновение пользователя перемещается за границы экранного объекта.

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);
var myTextField:TextField = new TextField();
addChild(myTextField);
myTextField.width = 200;
myTextField.height = 20;
var touchMoveID:int = 0;

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
function onTouchBegin(event:TouchEvent) {
    if(touchMoveID != 0) {
        myTextField.text = "already moving. ignoring new touch";
        return;
    }
    touchMoveID = event.touchPointID;

    myTextField.text = "touch begin" + event.touchPointID;
    stage.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
    stage.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);
```

```
}
function onTouchMove(event:TouchEvent) {
    if(event.touchPointID != touchMoveID) {
        myTextField.text = "ignoring unrelated touch";
        return;
    }
    mySprite.x = event.stageX;
    mySprite.y = event.stageY;
    myTextField.text = "touch move" + event.touchPointID;
}
function onTouchEnd(event:TouchEvent) {
    if(event.touchPointID != touchMoveID) {
        myTextField.text = "ignoring unrelated touch end";
        return;
    }
    touchMoveID = 0;
    stage.removeEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
    stage.removeEventListener(TouchEvent.TOUCH_END, onTouchEnd);
    myTextField.text = "touch end" + event.touchPointID;
}
```

# Прикосновение и перетаскивание

**Flash Player 10.1 и более поздних версий, Adobe AIR 2 и более поздних версий**

Два метода добавлены в класс Sprite для обеспечения дополнительной поддержки для приложений, поддерживающих сенсорный ввод: `Sprite.startTouchDrag()` и `Sprite.stopTouchDrag()`. Поведение этих методов аналогично поведению методов `Sprite.startDrag()` и `Sprite.stopDrag()` для событий мыши. Однако обратите внимание, что в методах `Sprite.startTouchDrag()` и `Sprite.stopTouchDrag()` значения `touchPointID` используются в качестве параметров.

Среда выполнения назначает значение `touchPointID` объекту события для события прикосновения. Используйте это значение для создания реакции на определенную точку прикосновения, если среда поддерживает несколько одновременных точек прикосновения (даже если она не обрабатывает жесты). Дополнительные сведения о свойстве `touchPointID` см. в разделе ««Идентификатор точки прикосновения» на странице 630».

Следующий код показывает простой обработчик события начала перетаскивания и обработчик события окончания перетаскивания для события прикосновения. Переменная *bg* является экранным объектом, содержащим объект *mySprite*:

```
mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
mySprite.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);

function onTouchBegin(e:TouchEvent) {
    e.target.startTouchDrag(e.touchPointID, false, bg.getRect(this));
    trace("touch begin");
}

function onTouchEnd(e:TouchEvent) {
    e.target.stopTouchDrag(e.touchPointID);
    trace("touch end");
}
```

А далее показан более сложный пример, в котором перетаскивание сочетается с фазами события прикосновения:

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
mySprite.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
mySprite.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);

function onTouchBegin(evt:TouchEvent) {
    evt.target.startTouchDrag(evt.touchPointID);
    evt.target.scaleX *= 1.5;
    evt.target.scaleY *= 1.5;
}

function onTouchMove(evt:TouchEvent) {
    evt.target.alpha = 0.5;
}

function onTouchEnd(evt:TouchEvent) {
    evt.target.stopTouchDrag(evt.touchPointID);
    evt.target.width = 40;
    evt.target.height = 40;
    evt.target.alpha = 1;
}
```

# Обработка событий жеста

**Flash Player 10.1 и более поздних версий, Adobe AIR 2 и более поздних версий**

Обработка событий жестов осуществляется так же, как и обработка основных событий прикосновения. Можно прослушать последовательность событий жестов, заданных константами типов событий в классах TransformGestureEvent, GestureEvent и PressAndTapGestureEvent.

Обработка события жеста:

1. Задайте в приложении обработку ввода жестом, установив для свойства `flash.ui.Multitouch.inputMode` значение `MultitouchInputMode.GESTURE`.
2. Назначьте прослушиватель событий экземпляру класса, который наследует свойства класса InteractiveObject, например Sprite или TextField.
3. Укажите тип обрабатываемого события жеста.
4. Вызовите функцию обработчика события для выполнения действия в ответ на событие.

Например, следующий код позволяет отобразить сообщение, когда пользователь проводит по квадрату, нарисованному в объекте *mySprite*, на сенсорном экране:

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TransformGestureEvent.GESTURE_SWIPE, swipehandler);

function swipehandler(evt:TransformGestureEvent): void {
myTextField.text = "I've been swiped";
myTextField.y = 50;
addChild(myTextField);
}
```

События прикосновения двумя пальцами обрабатываются так же, но с использованием класса GestureEvent:

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(GestureEvent.GESTURE_TWO_FINGER_TAP, taphandler);

function taphandler(evt:GestureEvent): void {
myTextField.text = "I've been two-finger tapped";
myTextField.y = 50;
addChild(myTextField);
}
```

События удерживания и нажатия также обрабатываются аналогичным образом, но с использованием класса PressAndTapGestureEvent:

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(PressAndTapGestureEvent.GESTURE_PRESS_AND_TAP, taphandler);

function taphandler(evt:PressAndTapGestureEvent): void {
myTextField.text = "I've been press-and-tapped";
myTextField.y = 50;
addChild(myTextField);
}
```

***Примечание.*** *Все типы событий GestureEvent, TransformGestureEvent и PressAndTapGestureEvent могут не поддерживаться во всех средах. Например, не все устройства с сенсорным экраном поддерживают обнаружение перемещения нескольких пальцев. Поэтому события* `gestureSwipe` *объекта InteractiveObject не поддерживаются в этих устройствах. Попробуйте проверить наличие поддержки определенного события для обеспечения правильной работы приложения и см. раздел ««Устранение неполадок» на странице 637» для получения дополнительных сведений.*

## Свойства событий жеста

События жестов имеют меньший диапазон свойств событий, чем основные события прикосновения. Доступ к ним осуществляется так же: через объект события в функции обработчика события.

Например, в следующем коде выполняется поворот объекта *mySprite*, когда пользователь выполняет в нем жест поворота. В текстовом поле показан угол поворота, выполненного с помощью последнего жеста (при тестировании кода поверните объект несколько раз для просмотра изменения значения):

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var mySpriteCon:Sprite = new Sprite();
var myTextField:TextField = new TextField();
myTextField.y = 50;
addChild(myTextField);

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(-20,-20,40,40);
mySpriteCon.addChild(mySprite);
mySprite.x = 20;
mySprite.y = 20;
addChild(mySpriteCon);

mySprite.addEventListener(TransformGestureEvent.GESTURE_ROTATE, rothandler);

function rothandler(evt:TransformGestureEvent): void {
evt.target.parent.rotationZ += evt.target.rotation;
myTextField.text = evt.target.parent.rotation.toString();
}
```

***Примечание.*** *Все свойства TransformGestureEvent могут поддерживаться не во всех средах. Например, не все устройства с сенсорным экраном поддерживают обнаружение жеста поворота на экране. Поэтому свойство* `TransformGestureEvent.rotation` *не поддерживается в этих устройствах. Попробуйте проверить наличие поддержки определенного свойства для обеспечения правильной работы приложения и см. раздел ««Устранение неполадок» на странице 637» для получения дополнительных сведений.*

## Фазы жеста

Кроме того, события жестов можно отслеживать с помощью фаз, поэтому можно отслеживать свойства при выполнении жеста. Например, можно отслеживать координаты X, когда объект перемещается жестом перемещения. Используйте эти значения для рисования линии, проходящей через все точки на пути перемещения, по завершении выполнения жеста перемещения. Или визуально измените экранный объект при его перетаскивании по экрану с помощью жеста панорамирования. Снова измените экранный объект по завершении выполнения жеста панорамирования.

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;
var mySprite = new Sprite();
mySprite.addEventListener(TransformGestureEvent.GESTURE_PAN , onPan);
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0, 0, 40, 40);
var myTextField = new TextField();
myTextField.y = 200;
addChild(mySprite);
addChild(myTextField);

function onPan(evt:TransformGestureEvent):void {

    evt.target.localX++;

if (evt.phase==GesturePhase.BEGIN) {
    myTextField.text = "Begin";
    evt.target.scaleX *= 1.5;
    evt.target.scaleY *= 1.5;
}
if (evt.phase==GesturePhase.UPDATE) {
    myTextField.text = "Update";
    evt.target.alpha = 0.5;
}
if (evt.phase==GesturePhase.END) {
    myTextField.text = "End";
    evt.target.width = 40;
    evt.target.height = 40;
    evt.target.alpha = 1;
}
}
```

***Примечание.*** *Частота фазы обновления зависит от среды, в которой работает среда выполнения. При некоторых комбинациях операционной системы и оборудования обновления не срабатывают вовсе.*

## Фаза жеста — основа простых событий жестов

Некоторые объекты события жеста не отслеживают отдельные фазы события жеста и указывают значение all в свойстве фазы объекта события. При простых жестах перемещения пальцев или прикосновения двумя пальцами событие не отслеживается по нескольким фазам. Свойство `phase` объекта события для прослушивания объектом InteractiveObject listening событий `gestureSwipe` или `gestureTwoFingerTap` всегда имеет значение `all` при отправке события:

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;
var mySprite = new Sprite();
mySprite.addEventListener(TransformGestureEvent.GESTURE_SWIPE, onSwipe);
mySprite.addEventListener(GestureEvent.GESTURE_TWO_FINGER_TAP, onTwoTap);
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0, 0, 40, 40);
var myTextField = new TextField();
myTextField.y = 200;
addChild(mySprite);
addChild(myTextField);

function onSwipe(swipeEvt:TransformGestureEvent):void {
    myTextField.text = swipeEvt.phase // Output is "all"
}
function onTwoTap(tapEvt:GestureEvent):void {
    myTextField.text = tapEvt.phase // Output is "all"
}
```

# Устранение неполадок

**Flash Player 10.1 и более поздних версий, Adobe AIR 2 и более поздних версий**

Сведения об аппаратной и программной поддержке сенсорного ввода быстро изменяются. В этом справочном описании не содержится список всех комбинаций устройств, операционных систем и программного обеспечения, в которых поддерживается технология Multitouch. Однако в нем приводятся рекомендации по использованию API-интерфейса обнаружения для определения, развернуто ли приложение в устройстве, поддерживающем технологию Multitouch, и советы по устранению ошибок в коде ActionScript.

Среды выполнения Flash реагируют на события прикосновения на основе информации, которую устройство, операционная система или приложение-контейнер (например, браузер) передает в среду выполнения. Эта зависимость от программной среды усложняет создание документации по совместимости Multitouch. В разных устройствах интерпретация жеста или прикосновения может отличаться. Задан ли поворот как одновременный поворот двумя пальцами? Будет ли при совершении вращательного движения одного пальца рисоваться окружность на экране? В зависимости от аппаратной и программной среды жест поворота может быть любым из этих жестов или полностью отличаться. Поэтому устройство передает данные в операционную систему о пользовательском вводе, а операционная система в свою очередь передает информацию в среду выполнения. Если среда выполнения работает в браузере, программное обеспечение браузера иногда интерпретирует жест или событие прикосновения и не передает данные о вводе в среду выполнения. Это поведение аналогично поведению «горячих клавиш»: вы пытаетесь использовать определенную комбинацию клавиш для выполнения действия в проигрывателе Flash Player, работающем в браузере, а вместо этого в браузере открывается меню.

Отдельные API-интерфейс и классы выдают информацию, если они не совместимы с определенной операционной системой. Здесь можно ознакомиться с отдельными элементами API-интерфейса, начиная с класса Multitouch: http://help.adobe.com/ru_RU/FlashPlatform/reference/actionscript/3/flash/ui/Multitouch.html.

Здесь приводятся некоторые общие описания жестов и прикосновений:

**Панорамирование** Перемещайте палец слева направо или справа налево. В некоторых устройствах панорамирование выполняется двумя пальцами.

**Вращать** Прикоснитесь двумя пальцами, затем перемещайте их по кругу (как если бы оба пальца перемещались по воображаемой окружности на поверхности). Точка поворота задана посередине между точками прикосновения двух пальцев.

**Пролистывание** Быстро переместите три пальца слева направо, справа налево, сверху вниз или снизу вверх.

**Масштабирование** Прикоснитесь двумя пальцами, затем раздвиньте их для увеличения масштаба или сдвиньте для уменьшения масштаба.

**Удерживание и нажатие** Выполните перемещение или нажатие одним пальцем, затем коснитесь поверхности другим пальцем.

Для каждого устройства предусмотрена собственная документация по поддерживаемым жестам и выполнению каждого жеста в устройстве. Как правило, в зависимости от операционной системы пользователь не должен касаться другими пальцами устройства при выполнении жеста.

Если приложение не реагирует на события прикосновения или жесты, проверьте следующее:

1. Существуют ли прослушиватели событий, подключенные к классу объектов, наследуемому от класса InteractiveObject? Только экземпляры InteractiveObject могут прослушивать события прикосновения или мыши.
2. Выполняется ли тестирование приложения в среде выполнения Flash Professional CS5? Если да, попробуйте опубликовать и протестировать приложение, поскольку среда Flash Professional может препятствовать взаимодействию.
3. Начните с простого и проверьте работающий код (следующий пример кода взят из записи API-интерфейса для `Multitouch.inputMode`:

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField()

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taplistener);

function taplistener(e:TouchEvent): void {
    myTextField.text = "I've been tapped";
    myTextField.y = 50;
    addChild(myTextField);
}
```

   Нажмите на прямоугольник. Если этот пример работает, то известно, что среда поддерживает простое прикосновение. После этого можно попробовать выполнить более сложную обработку.

   Протестировать поддержку жестов сложнее. Определенное устройство или операционная система поддерживает любую комбинацию ввода с помощью жестов или ни одну из них.

   В следующем примере проверяется наличие поддержки для жеста увеличения.

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;

stage.addEventListener(TransformGestureEvent.GESTURE_ZOOM , onZoom);
var myTextField = new TextField();
myTextField.y = 200;
myTextField.text = "Perform a zoom gesture";
addChild(myTextField);

function onZoom(evt:TransformGestureEvent):void {
    myTextField.text = "Zoom is supported";
}
```

Выполните жест увеличения на устройстве и проверьте, появится ли в текстовом поле сообщение «`Zoom is supported`» (Увеличение поддерживается). Для рабочей области добавляется прослушиватель событий, чтобы жест можно было выполнить в любой части тестового приложения.

В следующем примере проверяется наличие поддержки для жеста панорамирования.

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;

stage.addEventListener(TransformGestureEvent.GESTURE_PAN , onPan);
var myTextField = new TextField();
myTextField.y = 200;
myTextField.text = "Perform a pan gesture";
addChild(myTextField);

function onPan(evt:TransformGestureEvent):void {
    myTextField.text = "Pan is supported";
}
```

Выполните жест панорамирования на устройстве и проверьте, появится ли в текстовом поле сообщение «`Pan is supported`» (Панорамирование поддерживается). Для рабочей области добавляется прослушиватель событий, чтобы жест можно было выполнить в любой части тестового приложения.

Для некоторых комбинаций устройства и операционной системы поддерживаются оба жеста, для других — только один из них, а для третьих — ни один. Для уверенности следует протестировать среду развертывания приложения.

## Известные проблемы

С сенсорным вводом связаны следующие выявленные неполадки:

1 Mobile Internet Explorer в операционной системе Windows Mobile автоматически масштабирует содержимое SWF-файла:

   Это поведение масштабирования Internet Explorer переопределяется путем добавления следующей страницы HTML, на которой размещен SWF-файл:

```
<head>
<meta name="viewport" content="width=device-width, height=device-height, initial-
scale=1.0">
</head>
```

2 В Windows 7 (и, возможно, других операционных системах) пользователь должен убирать инструмент указания (или пальцы) с экрана между жестами. Например, для поворота и масштабирования изображения необходимо выполнить следующее:

   - Выполнить жест поворота.

- Убрать пальцы с экрана.
- Снова коснуться пальцами экрана и выполнить жест масштабирования.

3 В Windows 7 (и, возможно, в других ОС) жесты поворота и масштабирования могут не генерировать фазу обновления, если пользователь выполняет их слишком быстро.

4 В ОС Windows 7 Starter Edition не поддерживается ввод Multitouch. См. форум AIR Labs Forum для получения сведений: http://forums.adobe.com/thread/579180?tstart=0

5 Для ОС Mac OS 10.5.3 и более поздних версий значением `Multitouch.supportsGestureEvents` всегда является `true`, даже если отсутствует аппаратная поддержка жестов.

# Глава 33. Копирование и вставка

**Flash Player 10 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для копирования информации в буфер обмена и из него используются классы API-интерфейса буфера обмена. Следующие форматы данных можно переносить в приложение или извлекать из приложения, выполняемого в среде Adobe® Flash® Player или Adobe® AIR®.

- Текст
- Текст в формате HTML
- Данные в формате RTF (Rich Text Format)
- Сериализованные объекты
- Ссылки на объекты (действует только в пределах исходного приложения)
- Растровые изображения (только для AIR)
- Файлы (только для AIR)
- Строки URL (только для AIR)

## Основы копирования и вставки

**Flash Player 10 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Ниже перечислены классы API-интерфейса копирования и вставки.

| Пакет | Классы |
|---|---|
| flash.desktop | • Clipboard<br>• ClipboardFormats<br>• ClipboardTransferMode |

Статическое свойство `Clipboard.generalClipboard` представляет системный буфер обмена. Класс Clipboard предлагает методы для чтения и записи данных в объекты буфера обмена.

Класс HTMLLoader (в среде AIR) и класс TextField реализуют поведение по умолчанию для обычных сочетаний клавиш для копирования и вставки. Для реализации поведения клавиатурных сокращений копирования и вставки для заказных компонентов можно прослушивать непосредственно нажатия клавиш. Также для реакции на нажатия клавиш, помимо клавишных эквивалентов, могут использоваться команды исходного меню.

В одном объекте Clipboard могут содержаться разные представления одной и той же информации, что расширяет круг приложений, которые могут распознать и использовать эти данные. Например, изображение может храниться в виде данных изображения, сериализованного объекта и файла. Визуализация данных в тот или иной формат может происходить с задержкой (отложенная визуализация), таким образом, формат не будет создан, пока данные в этом формате не будут считаны.

# Чтение из системного буфера обмена и запись в него

**Flash Player 10 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для чтения буфера обмена операционной системы нужно вызвать метод `getData()` объекта `Clipboard.generalClipbooard` и передать имя формата для чтения:

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

if(Clipboard.generalClipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
    var text:String = Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT);
}
```

***Примечание.*** *Содержимое, выполняемое в проигрывателе Flash Player или в AIR за пределами изолированной программной среды приложения, может вызывать метод `getData()` только в обработчике события `paste`. Другими словами,только код, выполняемый в изолированной программной среде приложения AIR, может вызывать метод `getData()` за пределами обработчика события `paste`.*

Для записи в буфер обмена добавьте данные в объект `Clipboard.generalClipboard` в одном или нескольких форматах. Все существующие данные в таком же формате автоматически перезаписываются. Однако перед записью новых данных системный буфер обмена рекомендуется очищать, чтобы удалить нерелевантные данные в других форматах.

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

var textToCopy:String = "Copy to clipboard.";
Clipboard.generalClipboard.clear();
Clipboard.generalClipboard.setData(ClipboardFormats.TEXT_FORMAT, textToCopy, false);
```

***Примечание.*** *Содержимое, выполняемое в проигрывателе Flash Player или в AIR за пределами изолированной программной среды приложения, может вызывать метод `setData()` только в обработчике событий пользователя, таких как события клавиатуры, мыши, `copy` или `cut`. Другими словами,только код, выполняемый в изолированной программной среде приложения AIR, может вызывать метод `getData()` за пределами обработчика события пользователя.*

# Копирование и вставка HTML в AIR

**Adobe AIR 1.0 и более поздних версий**

Среда HTML в Adobe AIR имеет собственный набор событий и обеспечивает поведение по умолчанию для копирования и вставки. Доступ к системному буферу обмена имеет только код, выполняемый в изолированной программной среде. Доступ осуществляется напрямую через объект AIR `Clipboard.generalClipboard`. Код JavaScript в изолированной программной среде вне приложения получает доступ к буферу обмена через объект события, отправляемый в ответ на одно из событий копирования или вставки, которые, в свою очередь, отправляются элементом HTML-документа.

Существуют следующие события копирования и вставки: `copy`, `cut` и `paste`. Объект, отправляемый в ответ на эти события, обеспечивает доступ к буферу обмена через свойство `clipboardData`.

## Поведение по умолчанию

**Adobe AIR 1.0 и более поздних версий**

По умолчанию среда AIR копирует выделенные элементы в ответ на команду copy, которая может отправляться с клавиатуры или из контекстного меню. В редактируемых областях в ответ на команду cut AIR вырезает текст, а в ответ на команду paste — вставляет текст в точку расположения курсора или вместо выделенного текста.

Для отмены поведения по умолчанию обработчик события может вызывать метод `preventDefault()` отправляемого объекта события.

## Использования свойства clipboardData объекта события

**Adobe AIR 1.0 и более поздних версий**

Свойство `clipboardData` объекта события отправляется в результате события copy или paste и позволяет считывать и записывать данные в буфер обмена.

Для записи данных в буфер обмена при обработке события copy или cut используйте метод `setData()` объекта `clipboardData`, передающего копируемые данные и MIME-тип:

```
function customCopy(event){
    event.clipboardData.setData("text/plain", "A copied string.");
}
```

Для доступа к вставляемым данным может использоваться метод `getData()` объекта `clipboardData`, передающий MIME-тип формата данных. Доступные форматы содержатся в свойстве `types`.

```
function customPaste(event){
    var pastedData = event.clipboardData("text/plain");
}
```

Доступ к методу `getData()` и свойству `types` возможен только из объекта события, отправляемого событием `paste`.

В примере ниже показано, как переопределить поведение копирования и вставки по умолчанию на HTML-странице. Обработчик события `copy` выделяет копируемый текст курсивом и копирует его в буфер обмена в виде HTML-текста. Обработчик события `cut` копирует выделенные данные в буфер обмена и удаляет их из документа. Обработчик события `paste` вставляет содержимое буфера обмена в виде HTML и выделяет его полужирным шрифтом.

```
 <html>
<head>
    <title>Copy and Paste</title>
    <script language="javascript" type="text/javascript">
        function onCopy(event){
            var selection = window.getSelection();
            event.clipboardData.setData("text/html","<i>" + selection + "</i>");
            event.preventDefault();
        }

        function onCut(event){
             var selection = window.getSelection();
             event.clipboardData.setData("text/html","<i>" + selection + "</i>");
             var range = selection.getRangeAt(0);
             range.extractContents();

            event.preventDefault();
        }

        function onPaste(event){
            var insertion = document.createElement("b");
            insertion.innerHTML = event.clipboardData.getData("text/html");
             var selection = window.getSelection();
             var range = selection.getRangeAt(0);
             range.insertNode(insertion);
            event.preventDefault();
        }
    </script>
</head>
<body onCopy="onCopy(event)"
     onPaste="onPaste(event)"
     onCut="onCut(event)">
<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore
veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam
voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur
magni dolores eos qui ratione voluptatem sequi nesciunt.</p>
</body>
</html>
```

# Форматы данных Clipboard

**Flash Player 10 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Форматы Clipboard описывают данные, помещенные в объект Clipboard. Проигрыватель Flash Player и среда AIR автоматически переводят типы данных ActionScript и форматы данных буфера обмена в стандартные форматы данных. Кроме того, объекты приложения можно переносить между приложениями, созданными на базе ActionScript, и внутри них с помощью форматов, определенных в приложении.

Объект Clipboard может содержать представления одной и той же информации в разных форматах. Например, объект Clipboard, представляющий экземпляр Sprite, может включать формат ссылки для использования в том же приложении, сериализованный формат для использования другим приложением, выполняемым в проигрывателе Flash Player или среде AIR, формат растрового изображения для использования графическим редактором и формат списка файлов, возможно, с отложенной визуализацией для кодирования файла PNG, чтобы копировать или перетаскивать представление объекта Sprite в файловой системе.

## Стандартные форматы данных

**Flash Player 10 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Константы, определяющие названия стандартных форматов, содержатся в классе ClipboardFormats:

| Константа | Описание |
|---|---|
| TEXT_FORMAT | Данные в текстовом формате переводятся в класс String языка ActionScript и обратно. |
| HTML_FORMAT | Текст с HTML-разметкой. |
| RICH_TEXT_FORMAT | Данные в формате RTF переводятся в класс ByteArray языка ActionScript и обратно. RTF-разметка не прерывается и не переводится. |
| BITMAP_FORMAT | (только для AIR) Данные в формате растрового изображения преобразуются в класс ActionScript BitmapData и обратно. |
| FILE_LIST_FORMAT | (только для AIR) Данные в формате списка файлов преобразуются в массив объектов ActionScript File и обратно. |
| URL_FORMAT | (только для AIR) Данные в формате URL преобразуются в класс ActionScript String и обратно. |

При копировании и вставке данных в ответ на событие `copy`, `cut` или `paste` в HTML-содержимом, размещенном в приложении AIR, вместо строк ClipboardFormat следует использовать MIME-типы. Ниже перечислены поддерживаемые MIME-типы данных:

| Тип MIME | Описание |
|---|---|
| Текст | «text/plain»; |
| URL-адрес | «text/uri-list»; |
| Растровое изображение | «image/x-vnd.adobe.air.bitmap»; |
| Список файлов | «application/x-vnd.adobe.air.file-list». |

***Примечание.*** *Доступ к данным в формате RTF нельзя получить из свойства `clipboardData` объекта события в результате события `paste` в HTML-содержимом.*

## Заказные форматы данных

**Flash Player 10 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Пользовательские форматы приложения могут использоваться для передачи объектов в виде ссылок или сериализованных копий. Ссылки действуют только в пределах одного и того же приложения. Сериализованные объекты могут передаваться между приложениями, но использовать их можно только с объектами, которые остаются действительными в результате сериализации и десериализации. Как правило, объекты поддаются сериализации, если их свойства либо относятся к простому типу, либо являются сериализованными объектами.

Для добавления сериализованного объекта к объекту Clipboard при вызове метода `Clipboard.setData()` параметр *serializable* должен быть равен `true`. Именем формата может быть одно из стандартных названий или строка, произвольно заданная приложением.

### Режимы передачи

**Flash Player 10 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Когда объект записывается в буфер обмена с использованием пользовательского формата данных, данные объекта могут считываться из буфера как ссылка или как сериализованная копия исходного объекта. Определено четыре режима передачи, в соответствии с которыми объекты переносятся как ссылки либо как сериализованные копии.

| Режим передачи | Описание |
|---|---|
| ClipboardTransferModes.ORIGINAL_ONLY | Возвращается только ссылка. Если ссылки нет, возвращается значение null. |
| ClipboardTransferModes.ORIGINAL_PREFFERED | Возвращается ссылка (при наличии). В остальных случаях возвращается сериализованная копия. |
| ClipboardTransferModes.CLONE_ONLY | Возвращается только сериализованная копия. Если сериализованная копия недоступна, возвращается нулевое значение. |
| ClipboardTransferModes.CLONE_PREFFERED | Возвращается сериализованная копия (при наличии). В остальных случаях возвращается ссылка. |

### Чтение и запись данных в заказных форматах

**Flash Player 10 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При записи объекта в буфер обмена можно использовать любую строку, которая не начинается с зарезервированных префиксов `air:` или `flash:` для параметра *format*. Для чтения объекта используйте в качестве формата ту же строку. В примерах ниже показано, как считывать и записывать объекты в буфер обмена.

```
public function createClipboardObject(object:Object):Clipboard{
    var transfer:Clipboard = Clipboard.generalClipboard;
    transfer.setData("object", object, true);
}
```

Чтобы извлечь сериализованный объект из объекта Clipboard (после операции перетаскивания или вставки), используйте то же имя формата и режимы передачи `CLONE_ONLY` или `CLONE_PREFFERED`.

```
var transfer:Object = clipboard.getData("object", ClipboardTransferMode.CLONE_ONLY);
```

К объекту Clipboard всегда добавляется ссылка. Для извлечения из объекта буфера обмена ссылки, а не сериализованной копии (после операции drop или paste) используйте режимы передачи `ORIGINAL_ONLY` или `ORIGINAL_PREFFERED`:

```
var transferredObject:Object =
    clipboard.getData("object", ClipboardTransferMode.ORIGINAL_ONLY);
```

Ссылки действительны, только если источником объекта Clipboard является текущее приложение. Используйте режим передачи `ORIGINAL_PREFFERED`, чтобы получить ссылку, если она имеется, или сериализованную копию, если она недоступна.

## Отложенная визуализация

**Flash Player 10 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если создание формата затратно с точки зрения вычислительных ресурсов, можно использовать отложенную визуализацию: для этого необходима функция, предоставляющая данные по запросу. Эта функция вызывается, только если получатель операции перетаскивания или вставки запрашивает данные в отложенном формате.

Функция визуализации добавляется к объекту Clipboard с помощью метода `setDataHandler()`. Функция должна возвращать данные в подходящем формате. Например, при вызове метода `setDataHandler(ClipboardFormat.TEXT_FORMAT, writeText)` функция `writeText()` должна возвращать строку.

Если к объекту Clipboard с помощью метода `setData()` добавляются данные в таком же формате, эти данные будут иметь приоритет перед отложенными данными (функция визуализации не будет вызвана). При повторном доступе к данным буфера обмена функция визуализации может быть вызвана, а может и не быть вызвана.

***Примечание.*** *В ОС Mac OS X отложенная визуализация поддерживается только для пользовательских форматов данных. Для стандартных форматов данных функция визуализации вызывается немедленно.*

### Вставка текста с помощью функции отложенной визуализации

**Flash Player 10 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В примере ниже показано, как выполнять отложенную визуализацию.

Когда пользователь нажимает кнопку Copy (Копировать), приложение освобождает системный буфер обмена, чтобы удалить данные, оставшиеся после предыдущих операций с буфером обмена. Затем метод `setDataHandler()` задает функцию `renderData()` в качестве визуализатора буфера обмена.

Когда пользователь выбирает команду Paste (Вставить) из контекстного меню целевого текстового поля, приложение обращается к буферу обмена и задает целевой текст. Так как текстовый формат данных в буфере обмена задан с использованием функции, а не строки, буфер обмена вызывает функцию `renderData()`. Функция `renderData()` возвращает текст в исходном тексте, который впоследствии назначается целевому.

Обратите внимание, что если исходный текст редактируется до нажатия кнопки вставки, то эта правка отразится во вставляемом тексте, даже если текст редактировался уже после нажатия кнопки копирования. Это происходит из-за того, что функция визуализации не копирует текст из источника, пока не будет нажата кнопка вставки. (При использовании отложенной визуализации в рабочем приложении для предотвращения такой ситуации можно сохранить или защитить исходные данные.)

### Пример Flash

```
package {
   import flash.desktop.Clipboard;
   import flash.desktop.ClipboardFormats;
   import flash.desktop.ClipboardTransferMode;
   import flash.display.Sprite;
   import flash.text.TextField;
   import flash.text.TextFormat;
   import flash.text.TextFieldType;
   import flash.events.MouseEvent;
   import flash.events.Event;
   public class DeferredRenderingExample extends Sprite
   {
       private var sourceTextField:TextField;
       private var destination:TextField;
       private var copyText:TextField;
       public function DeferredRenderingExample():void
       {
           sourceTextField = createTextField(10, 10, 380, 90);
           sourceTextField.text = "Neque porro quisquam est qui dolorem "
               + "ipsum quia dolor sit amet, consectetur, adipisci velit.";

           copyText = createTextField(10, 110, 35, 20);
           copyText.htmlText = "<a href='#'>Copy</a>";
           copyText.addEventListener(MouseEvent.CLICK, onCopy);

           destination = createTextField(10, 145, 380, 90);
           destination.addEventListener(Event.PASTE, onPaste);
       }
       private function createTextField(x:Number, y:Number, width:Number,
                       height:Number):TextField
       {
           var newTxt:TextField = new TextField();
           newTxt.x = x;
           newTxt.y = y;
           newTxt.height = height;
           newTxt.width = width;
           newTxt.border = true;
           newTxt.multiline = true;
           newTxt.wordWrap = true;
           newTxt.type = TextFieldType.INPUT;
           addChild(newTxt);
           return newTxt;
       }
       public function onCopy(event:MouseEvent):void
       {
           Clipboard.generalClipboard.clear();
           Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT,
                           renderData);
       }
       public function onPaste(event:Event):void
       {
```

```
            sourceTextField.text =
            Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
        }
        public function renderData():String
        {
            trace("Rendering data");
            var sourceStr:String = sourceTextField.text;
            if (sourceTextField.selectionEndIndex >
                     sourceTextField.selectionBeginIndex)
            {
                return sourceStr.substring(sourceTextField.selectionBeginIndex,
                                    sourceTextField.selectionEndIndex);
            }
            else
            {
                return sourceStr;
            }
        }
    }
}
```

### Пример Flex

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" width="326"
height="330" applicationComplete="init()">
    <mx:Script>
    <![CDATA[
    import flash.desktop.Clipboard;
    import flash.desktop.ClipboardFormats;

    public function init():void
    {
        destination.addEventListener("paste", doPaste);
    }

    public function doCopy():void
    {
        Clipboard.generalClipboard.clear();
        Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT, renderData);
    }
    public function doPaste(event:Event):void
    {
        destination.text =
Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
    }

    public function renderData():String{
        trace("Rendering data");
        return source.text;
    }
    ]]>
    </mx:Script>
    <mx:Label x="10" y="10" text="Source"/>
    <mx:TextArea id="source" x="10" y="36" width="300" height="100">
        <mx:text>Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur,
adipisci velit.</mx:text>
    </mx:TextArea>
    <mx:Label x="10" y="181" text="Destination"/>
    <mx:TextArea id="destination"  x="12" y="207" width="300" height="100"/>
    <mx:Button click="doCopy();" x="91" y="156" label="Copy"/>
</mx:Application>
```

# Глава 34. Применение акселерометра

**Flash Player 10.1 и более поздних версий, Adobe AIR 2 и более поздних версий**

Класс Accelerometer отправляет события на основании активности, обнаруживаемой датчиком движения устройства. Эти данные представляют местоположение устройства или движение вдоль трехмерной оси. Когда устройство перемещается, датчик обнаруживает это движение и возвращает координаты ускорения устройства. Класс Accelerometer предоставляет методы, позволяющие сделать запрос, поддерживается ли акселерометр, а также установить частоту отправки событий ускорения.

Оси акселерометра нормализуются в соответствии с ориентацией экрана, а не физической ориентацией устройства. При изменении ориентации экрана ориентация осей акселерометра изменяется соответствующим образом. Таким образом, ось y всегда расположена примерно вертикально, когда телефон находится в обычном вертикальном положении для просмотра независимо от того, каким образом повернут телефон. Если автоматическая ориентация отключена, например когда SWF-содержимое в браузере отображается в полноэкранном режиме, изменение ориентации осей акселерометра при повороте устройства не выполняется.

### Дополнительные разделы справки

flash.sensors.Accelerometer

flash.events.AccelerometerEvent

## Проверка наличия поддержки акселерометра

Используйте свойство `Accelerometer.isSupported`, чтобы проверить наличие поддержки этой функции в среде выполнения:

```
if (Accelerometer.isSupported)
{
    // Set up Accelerometer event listeners and code.
}
```

Класс Accelerometer и его члены доступны для версий среды выполнения, перечисленных для каждой записи API-интерфейса. Однако текущая среда во время выполнения определяет доступность этой функции. Например, можно скомпилировать код с использованием свойств класса Accelerometer для Flash Player 10.1, однако необходимо использовать свойство `Accelerometer.isSupported` для проверки доступности функции Accelerometer в устройстве пользователя. Если свойство `Accelerometer.isSupported` возвращает значение `true` во время выполнения, то текущая платформа поддерживает класс Accelerometer.

# Обнаружение изменений акселерометра

Чтобы использовать датчик акселерометра, создайте экземпляр объекта Accelerometer и зарегистрируйте отправляемое им событие `update`. Событие `update` является объектом события Accelerometer. Событие имеет четыре свойства, каждое из которых является числом:

- `accelerationX`: ускорение вдоль оси X, измеряется в Галах (Г). Ось X направлена слева направо относительно устройства, когда оно находится в вертикальном положении. (Устройство находится в вертикальном положении, когда верхняя часть устройства направлена вверх.) Ускорение имеет положительное значение, если устройство перемещается вправо.
- `accelerationY`: ускорение вдоль оси Y, измеряется в Галах (Г). Ось Y направлена снизу вверх относительно устройства, когда оно находится в вертикальном положении. (Устройство находится в вертикальном положении, когда верхняя часть устройства направлена вверх.) Ускорение имеет положительное значение, если устройство перемещается вверх относительно этой оси.
- `accelerationZ`: ускорение вдоль оси Z, измеряется в Галах (Г). Ось Z направлена перпендикулярно лицевой стороне устройства. Ускорение имеет положительное значение, если при перемещении устройства его лицевая сторона направлена вверх. Ускорение имеет отрицательное значение, если лицевая сторона устройства направлена в сторону земли.
- `timestamp`: число миллисекунд на момент события, начиная с инициализации среды выполнения.

1 Г — это стандартное ускорение в следствие гравитации, которое приближенно равно 9,8 м/с$^2$.

Здесь приводится основной пример отображения данных акселерометра в текстовом поле:

```
var accl:Accelerometer;
if (Accelerometer.isSupported)
{
    accl = new Accelerometer();
    accl.addEventListener(AccelerometerEvent.UPDATE, updateHandler);
}
else
{
    accTextField.text = "Accelerometer feature not supported";
}
function updateHandler(evt:AccelerometerEvent):void
{
    accTextField.text = "acceleration X: " + evt.accelerationX.toString() + "\n"
            + "acceleration Y: " + evt.accelerationY.toString() + "\n"
            + "acceleration Z: " + evt.accelerationZ.toString()
}
```

Чтобы воспользоваться этим примером, перед использованием кода создайте текстовое поле `accTextField` и добавьте его в список отображения.

Можно настроить необходимый интервал времени для событий акселерометра, вызвав метод `setRequestedUpdateInterval()` объекта Accelerometer. В этом методе используется один параметр `interval`, который представляет собой запрошенный интервал обновления в миллисекундах:

```
var accl:Accelerometer;
accl = new Accelerometer();
accl.setRequestedUpdateInterval(1000);
```

Фактическое время между обновлениями акселерометра может быть больше или меньше этого значения. Любое изменение в интервале обновления затрагивает все зарегистрированные прослушиватели. При отсутствии вызова метода `setRequestedUpdateInterval()` приложение получает обновления на основе интервала, заданного в устройстве по умолчанию.

Данные акселерометра имеют определенную погрешность. Можно использовать скользящее среднее значение по последним данным для сглаживания данных. В следующем примере коэффициент, рассчитанный на основе последних показателей акселерометра, умножается на текущие показания для получения округленного результата:

```
var accl:Accelerometer;
var rollingX:Number = 0;
var rollingY:Number = 0;
var rollingZ:Number = 0;
const FACTOR:Number = 0.25;

if (Accelerometer.isSupported)
{
    accl = new Accelerometer();
    accl.setRequestedUpdateInterval(200);
    accl.addEventListener(AccelerometerEvent.UPDATE, updateHandler);
}
else
{
    accTextField.text = "Accelerometer feature not supported";
}
function updateHandler(event:AccelerometerEvent):void
{
    accelRollingAvg(event);
    accTextField.text = rollingX + "\n" +  rollingY + "\n" + rollingZ + "\n";
}

function accelRollingAvg(event:AccelerometerEvent):void
{
    rollingX = (event.accelerationX * FACTOR) + (rollingX * (1 - FACTOR));
    rollingY = (event.accelerationY * FACTOR) + (rollingY * (1 - FACTOR));
    rollingZ = (event.accelerationZ * FACTOR) + (rollingZ * (1 - FACTOR));
}
```

Однако это скользящее среднее желательно использовать только при небольшом интервале обновления акселерометра.

# Глава 35. Перетаскивание в AIR

**Adobe AIR 1.0 и более поздних версий**

Жесты перетаскивания в пользовательском интерфейсе реализуются с помощью классов API Adobe® AIR™. *Жест* в данном смысле — это действие пользователя, распознаваемое операционной системой и приложением, по копированию, перемещению или созданию ссылки на данные. Жест «*перетаскивания из*» означает, что пользователь начинает перетаскивать объект из какого-либо компонента или приложения. Жест «*перетаскивания в*» означает, что пользователь начинает перетаскивать объект в какой-либо компонент или приложение извне.

С помощью API-интерфейса перетаскивания пользователь может перемещать данные между разными приложениями и между компонентами одного приложения. Ниже приведен список поддерживаемых форматов:

- Растровые изображения
- Файлы
- Текст в формате HTML
- Текст
- Данные в формате RTF (Rich Text Format)
- URL-адреса
- Предварительные образы файла
- Сериализованные объекты
- Ссылки на объекты (только в границах исходного приложения)

## Основы перетаскивания в AIR

**Adobe AIR 1.0 и более поздних версий**

Краткое разъяснение и примеры кодов с перетаскиванием в приложении AIR см. в следующих статьях краткого руководства на Adobe Developer Connection:

- Поддерживание операций перетаскивания, а также копирования и вставки (Flex)
- Поддерживание операций перетаскивания, а также копирования и вставки (Flash)

API-интерфейс перетаскивания содержит следующие классы.

| Пакет | Классы |
|---|---|
| flash.desktop | • NativeDragManager<br>• NativeDragOptions<br>• Clipboard<br>• URLFilePromise<br>• IFilePromise<br>Константы, используемые с API-интерфейсом перетаскивания, определены в следующих классах:<br>• NativeDragActions<br>• ClipboardFormat<br>• ClipboardTransferModes |
| flash.events | NativeDragEvent |

### Этапы перетаскивания

Жест перетаскивания включает три этапа:

**Начало** *Пользователь начинает операцию перетаскивания, удерживая кнопку мыши и перетаскивая объект из компонента или элемента компонента.* Компонент-источник обычно выступает в качестве инициатора перетаскивания и отправляет события `nativeDragStart` и `nativeDragComplete`. Приложение Adobe AIR начинает операцию перетаскивания вызовом метода `NativeDragManager.doDrag()` в ответ на событие `mouseDown` или `mouseMove`.

Если операция перетаскивания начинается извне приложения AIR, отсутствует объект-инициатор для обработки событий `nativeDragStart` или `nativeDragComplete`.

**Перетаскивание** *Удерживая кнопку мыши, пользователь переводит курсор на другой компонент, приложение или рабочий стол.* Во время выполнения перетаскивания его инициатор отправляет события `nativeDragUpdate`. (Однако это событие не отправляется в AIR for Linux.) Когда пользователь наводит указатель мыши на возможную цель перетаскивания в приложении AIR, целевой элемент отправляет событие `nativeDragEnter`. Обработчик событий может исследовать объект события, чтобы определить, принимает ли целевой элемент формат перетаскиваемых данных, и, если это так, разрешить пользователю оставить там данные, вызвав метод `NativeDragManager.acceptDragDrop()`.

Пока жест перетаскивания выполняется над интерактивным объектом, этот объект непрерывно отправляет события `nativeDragOver`. Когда жест выходит за границы интерактивного объекта, тот отправляет событие `nativeDragExit`.

**Отпускание** *Пользователь отпускает кнопку мыши в границах возможного целевого элемента.* Если целью является приложение или компонент AIR, то целевой объект отправляет событие `nativeDragDrop`. Обработчик событий имеет доступ к данным, переданным от объекта события. Если цель лежит за пределами AIR, то отпускание обрабатывается операционной системой или другим приложением. В обоих случаях объект-инициатор отправляет событие `nativeDragComplete` (если перетаскивание начинается в границах AIR).

Класс NativeDragManager отвечает как за жест «перетаскивания из», так и за жест «перетаскивания в». Все члены класса NativeDragManager являются статическими, поэтому экземпляры этого класса создавать не нужно.

**Объект Clipboard**

Данные, перетаскиваемые из приложения или компонента или обратно, сохраняются в объекте Clipboard. Один объект Clipboard может поддерживать различные представления одних и тех же данных, тем самым расширяя диапазон приложений, которые смогут распознавать и использовать эти данные. Например, изображение может храниться в виде данных изображения, сериализованного объекта и файла. Визуализация данных в определенном формате может задерживаться до вызова функции визуализации, а та не вызывается, пока не будут считаны данные.

После начала жеста перетаскивания доступ к объекту Clipboard может быть получен только из обработчика событий `nativeDragEnter`, `nativeDragOver` и `nativeDragDrop`. После окончания перетаскивания объект Clipboard не может быть считан или повторно использован.

Объект приложения может быть передан в качестве ссылки и в качестве сериализованного объекта. Ссылки действительны только в границах исходного приложения. Сериализованные объекты могут передаваться от одного приложения AIR к другому, но использовать их можно только с объектами, которые остаются действительными в результате сериализации и десериализации. Сериализованные объекты преобразуются в формат ActionScript 3 (AMF3) — формат передачи строковых данных.

**Работа с инфраструктурой Flex**

В большинстве случаев при создании приложений Flex рекомендуется использовать API-интерфейс перетаскивания Adobe® Flex™. Инфраструктура Flex предлагает идентичный набор функций, когда приложение Flex выполняется в среде AIR (с использованием AIR NativeDragManager). Flex поддерживает меньший набор функций, когда компонент или приложение выполняется в среде обозревателя с более строгими ограничениями. Классы AIR нельзя использовать в компонентах или приложениях, выполняющихся за пределами рабочей среды AIR.

# Поддержка жеста «перетаскивания из»

**Adobe AIR 1.0 и более поздних версий**

Для поддержки жеста перетаскивания необходимо создать объект Clipboard в ответ на событие `mouseDown` и отправить его методу `NativeDragManager.doDrag()`. Затем приложение может прослушивать событие `nativeDragComplete` на объекте-инициаторе, чтобы определить, какое действие необходимо выполнить, когда пользователь завершит или остановит действие.

## Подготовка данных для передачи

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы подготовить данные к перетаскиванию, создайте объект Clipboard и добавьте информацию, которую необходимо передать в одном или нескольких форматах. Для передачи данных можно использовать стандартные форматы, которые автоматически переводятся в исходные форматы буфера обмена, или форматы, заданные приложениями для передачи объектов.

Если перевод в определенный формат требует слишком много вычислительной мощности, то для выполнения преобразования можно передать имя функции обработчика. Функция вызывается только в том случае, если компонент или приложение-получатель распознают связанный формат.

Дополнительные сведения о форматах буфера обмена см. в разделе ««Форматы данных Clipboard» на странице 644».

В примере ниже показано, как создавать объект Clipboard, содержащий растровое изображение в нескольких форматах: объект Bitmap, исходный формат растрового изображения и формат списка файлов, содержащего файл, из которого изображение изначально загружается:

```
import flash.desktop.Clipboard;
import flash.display.Bitmap;
import flash.filesystem.File;
public function createClipboard(image:Bitmap, sourceFile:File):Clipboard{
    var transfer:Clipboard = new Clipboard();
    transfer.setData("CUSTOM_BITMAP", image, true); //Flash object by value and by reference
    transfer.setData(ClipboardFormats.BITMAP_FORMAT, image.bitmapData, false);
    transfer.setData(ClipboardFormats.FILE_LIST_FORMAT, new Array(sourceFile), false);
    return transfer;
}
```

## Начало операции «перетаскивания из»

**Adobe AIR 1.0 и более поздних версий**

Для начала операции перетаскивания нужно вызвать метод `NativeDragManager.doDrag()` в ответ на нажатие кнопки мыши. Метод `doDrag()` является статическим и принимает ряд параметров:

| Параметр | Описание |
|---|---|
| initiator | Объект-инициатор перетаскивания, отправляющий события `dragStart` и `dragComplete`. Инициатором всегда является интерактивный объект. |
| clipboard | Объект Clipboard, содержащий передаваемые данные. На объект Clipboard ссылается объект NativeDragEvent, отправляемый в ходе последовательности захвата и перетаскивания. |
| dragImage | Объект BitmapData, отображаемый во время перетаскивания (не обязательно). Для изображения может быть задано значение `alpha`. (Примечание. Microsoft Windows всегда применяет к перетаскиваемым изображениям предопределенное обесцвечивание альфа-канала.) |
| offset | Объект Point, задающий смещение изображения перетаскивания относительно указателя мыши (не обязательно). При отрицательных координатах изображение будет смещено вверх и влево относительно курсора. Если значение смещения не задано, то левый угол изображения перетаскивания располагается у кончика указателя мыши. |
| actionsAllowed | Объект NativeDragOptions, указывающий, какие действия (копирование, перемещение или создание ссылки) действительны для операции перетаскивания (не обязательно). Если ни один аргумент не передается, то разрешены все действия. На объект DragOptions ссылаются объекты NativeDragEvent, позволяющие проверить, совместима ли потенциальная цель перетаскивания с выполняемыми действиями. Например, компонент trash может принимать только перетаскивания типа «перемещение». |

В примере ниже показано, как начинать операцию перетаскивания для растрового изображения, загруженного из файла. В примере изображение загружается, а по событию `mouseDown` начинается его перетаскивание.

```
package
{
import flash.desktop.NativeDragManager;
import mx.core.UIComponent;
import flash.display.Sprite;
import flash.display.Loader;
import flash.system.LoaderContext;
import flash.net.URLRequest;
import flash.geom.Point;
import flash.desktop.Clipboard;
import flash.display.Bitmap;
import flash.filesystem.File;
import flash.events.Event;
import flash.events.MouseEvent;

public class DragOutExample extends UIComponent Sprite {
    protected var fileURL:String = "app:/image.jpg";
    protected var display:Bitmap;

    private function init():void {
        loadImage();
    }
    private function onMouseDown(event:MouseEvent):void {
        var bitmapFile:File = new File(fileURL);
        var transferObject:Clipboard = createClipboard(display, bitmapFile);
        NativeDragManager.doDrag(this,
                            transferObject,
                            display.bitmapData,
                            new Point(-mouseX,-mouseY));
    }
    public function createClipboard(image:Bitmap, sourceFile:File):Clipboard {
        var transfer:Clipboard = new Clipboard();
        transfer.setData("bitmap",
                            image,
                            true);
                            // ActionScript 3 Bitmap object by value and by reference
        transfer.setData(ClipboardFormats.BITMAP_FORMAT,
                            image.bitmapData,
                            false);
                            // Standard BitmapData format
        transfer.setData(ClipboardFormats.FILE_LIST_FORMAT,
```

```
                            new Array(sourceFile),
                            false);
                            // Standard file list format
        return transfer;
    }
    private function loadImage():void {
        var url:URLRequest = new URLRequest(fileURL);
        var loader:Loader = new Loader();
        loader.load(url,new LoaderContext());
        loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onLoadComplete);
    }
    private function onLoadComplete(event:Event):void {
        display = event.target.loader.content;
        var flexWrapper:UIComponent = new UIComponent();
        flexWrapper.addChild(event.target.loader.content);
        addChild(flexWrapper);
        flexWrapper.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
    }
}
}
```

## Завершение «перетаскивания из»

**Adobe AIR 1.0 и более поздних версий**

Когда пользователь отпускает перетаскиваемый объект, отпуская кнопку мыши, объект-инициатор отправляет событие `nativeDragComplete`. Можно проверить свойство `dropAction` объекта события, а затем выполнить нужное действие. Например, при действии `NativeDragAction.MOVE` исходный элемент может быть удален из своего начального местоположения. Пользователь может прекратить перетаскивание, отпустив кнопку мыши за пределами целевой области. Диспетчер перетаскивания задает свойству `dropAction` действия прекращения значение `NativeDragAction.NONE`.

# Поддержка жеста «перетаскивания в»

**Adobe AIR 1.0 и более поздних версий**

Поддержка жеста «перетаскивания в» возможна, если приложение (или, что более распространено, визуальный компонент приложения) отвечает на события `nativeDragEnter` или `nativeDragOver`.

## Этапы типичной операции отпускания

**Adobe AIR 1.0 и более поздних версий**

Для операции отпускания характерна определенная последовательность событий:

1. Пользователь перетаскивает объект буфера обмена на компонент.
2. Компонент отправляет событие `nativeDragEnter`.
3. Обработчик события `nativeDragEnter` исследует объект события на предмет доступных форматов данных и разрешенных действий. Если компонент может обработать отпускание, он вызывает метод `NativeDragManager.acceptDragDrop()`.

4 NativeDragManager изменяет вид курсора мыши, чтобы показать, что объект можно отпустить.

5 Пользователь отпускает объект на компоненте.

6 Компонент-получатель отправляет событие `nativeDragDrop`.

7 Компонент-получатель считывает из объекта Clipboard в объекте события данные в нужном формате.

8 Если жест перетаскивания начался в приложении AIR, то интерактивный объект-инициатор отправляет событие `nativeDragComplete`. Если жест начался вне приложения AIR, обратной связи нет.

## Распознавание жеста «перетаскивания в»

**Adobe AIR 1.0 и более поздних версий**

Когда пользователь перетаскивает элемент буфера обмена в границы визуального компонента, тот отправляет события `nativeDragEnter` и `nativeDragOver`. Чтобы определить, может ли компонент принять этот элемент, обработчики этих событий проверяют свойства `clipboard` и `allowedActions` объекта события. Для подтверждения возможности перетаскивания обработчик события должен вызвать метод `NativeDragManager.acceptDragDrop()` и передать компоненту-получателю ссылку. Если метод `acceptDragDrop()` вызывается сразу несколькими обработчиками, то преимущество имеет последний в списке. Вызов метода `acceptDragDrop()` остается действительным, пока курсор мыши не будет выведен за границы объекта-получателя и не будет отправлено событие `nativeDragExit`.

Если параметром `allowedActions`, передаваемым методу `doDrag()`, разрешено более одного действия, пользователь может указать желаемое действие с помощью клавиши-модификатора. Диспетчер перетаскивания изменяет изображение курсора, показывая пользователю, какое действие будет совершено при отпускании. О планируемом действии сообщает свойство `dropAction` объекта NativeDragEvent. Действие, заданное для жеста перетаскивания, приводится исключительно для информации. Поведение компонентов, задействованных в передаче, должно быть соответствующим. Для завершения, например, действия перемещения инициатор перетаскивания может удалять перетаскиваемый объект, а целевой элемент — добавлять его.

Цель перетаскивания может ограничивать число действий отпускания до трех возможных с помощью свойства `dropAction` класса NativeDragManager. Если пользователь пытается выбрать другое действие с помощью клавиатуры, NativeDragManager отобразит курсор вида *unavailable*. В обработчиках событий `nativeDragEnter` и `nativeDragOver` следует задать свойство `dropAction`.

В примере ниже рассматривается обработчик событий `nativeDragEnter` и `nativeDragOver`. Этот обработчик принимает жест «перетаскивания в», только если перетаскиваемый объект буфера обмена содержит данные в текстовом формате.

```
import flash.desktop.NativeDragManager;
import flash.events.NativeDragEvent;

public function onDragIn(event:NativeDragEvent):void{
    NativeDragManager.dropAction = NativeDragActions.MOVE;
    if(event.clipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
        NativeDragManager.acceptDragDrop(this); //'this' is the receiving component
    }
}
```

## Завершение перетаскивания

**Adobe AIR 1.0 и более поздних версий**

Когда пользователь отпускает перетаскиваемый элемент на интерактивном объекте, принимающем этот жест, тот отправляет событие `nativeDragDrop`. Обработчик этого события может извлекать данные из свойства `clipboard` объекта события.

Когда в буфере обмена содержится поддерживаемый приложением формат, параметр `transferMode`, передаваемый методу `getData()` объекта Clipboard, определяет, что должен возвращать диспетчер перетаскивания: ссылку или сериализованную версию объекта.

В примере ниже рассматривается обработчик события `nativeDragDrop`:

```
 import flash.desktop.Clipboard;
import flash.events.NativeDragEvent;

public function onDrop(event:NativeDragEvent):void {
    if (event.clipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)) {
    var text:String =
        String(event.clipboard.getData(ClipboardFormats.TEXT_FORMAT,
                                    ClipboardTransferMode.ORIGINAL_PREFERRED));
}
```

При появлении обработчика события объект Clipboard перестает действовать. Любая попытка доступа к этому объекту или его данным возвращает ошибку.

## Обновление внешнего вида компонента

**Adobe AIR 1.0 и более поздних версий**

Внешний вид компонента может обновляться на основании событий NativeDragEvent. В таблице ниже показаны типы изменений, которые могут происходить с типичным компонентом в ответ на различные события.

| Событие | Описание |
|---|---|
| nativeDragStart | Интерактивный объект-инициатор может использовать событие `nativeDragStart` для обозначения того, что интерактивный объект инициировал перетаскивание. |
| nativeDragUpdate | Интерактивный объект-инициатор может использовать событие nativeDragUpdate для обновления своего состояния во время этого жеста. (Это событие не существует в AIR for Linux.) |
| nativeDragEnter | Потенциальный интерактивный объект-получатель может использовать это событие для принятия фокуса или визуального обозначения того, что перетащить на него данные нельзя. |
| nativeDragOver | Потенциальный интерактивный объект-получатель может использовать это событие для ответа на движение мыши в границах интерактивного объекта, например когда курсор мыши входит в границы «горячей» области сложного компонента, такого как карта города. |
| nativeDragExit | Потенциальный интерактивный объект-получатель может использовать это событие для обозначения того, что курсор, перетаскивающий данные, покидает его границы. |
| nativeDragComplete | Потенциальный интерактивный объект-получатель может использовать это событие для обновления связанной с ним модели данных (например, путем удаления элемента из списка) и для восстановления своего визуального состояния. |

## Отслеживание положения мыши во время жеста «перетаскивания в»

**Adobe AIR 1.0 и более поздних версий**

Пока жест перетаскивания выполняется в границах компонента, отправляются события `nativeDragOver`. Эти события отправляются каждые несколько миллисекунд и при любом движении мыши. Объект события `nativeDragOver` может использоваться для определения положения мыши в границах компонентом. Доступ к положению мыши полезен в тех ситуациях, когда компонент-получатель является сложным, но при этом не состоит из нескольких компонентов. Например, если в приложении отображается растровое изображение карты города и на нем при перетаскивании информации требуется выделить некоторые области, то данные о координатах мыши можно получить с помощью события `nativeDragOver`, позволяющего отслеживать положение мыши на карте.

# Перетаскивание в HTML

**Adobe AIR 1.0 и более поздних версий**

Для перетаскивания данных в HTML-приложение и из него (или в HTML-файл, отображаемый в HTMLLoader, и из него), могут использоваться события drag HTML. API-интерфейс перетаскивания в HTML позволяет осуществлять перетаскивание в элементы модели DOM и обратно в HTML-содержимом.

***Примечание.*** *Также могут использоваться API-интерфейсы классов AIR NativeDragEvent и NativeDragManager. Для этого нужно прослушивать события на объекте HTMLLoader, в котором содержится HTML-файл. Тем не менее API-интерфейс HTML больше подходит для работы с HTML-моделью DOM и позволяет контролировать поведение по умолчанию.*

## Поведение по умолчанию при перетаскивании

**Adobe AIR 1.0 и более поздних версий**

В среде HTML задано поведение по умолчанию для жестов перетаскивания текста, изображений и URL-адресов. С помощью поведения по умолчанию всегда можно перетащить эти типы данных из одного элемента в другой. Однако текст можно перетаскивать только между редактируемыми областями страницы. При перетаскивании текста из одной редактируемой области в другую по умолчанию выполняется перемещение. При перетаскивании текста в редактируемую область страницы из нередактируемой или из другого приложения по умолчанию выполняется копирование.

Поведение по умолчанию можно переопределить. Для этого необходимо самостоятельно обрабатывать события перетаскивания. Для отмены поведения по умолчанию необходимо вызвать методы `preventDefault()` объектов, отправленных в ответ на события перетаскивания. При необходимости для выполнения нужного действия можно вставить данные в целевой элемент перетаскивания и удалить их из источника перетаскивания.

По умолчанию пользователь может выделить и перетащить любой текст, изображение или ссылку. Для управления поведением при выделении HTML-элемента можно использовать свойство CSS WebKit `-webkit-user-select`. Например, если значение свойства `-webkit-user-select` равно `none`, то содержимое элемента нельзя выбрать и перетащить. Также для указания того, можно ли перетащить элемент целиком, может использоваться свойство CSS `-webkit-user-drag`. Однако разные компоненты элемента будут обрабатываться по-разному. Пользователь по-прежнему сможет перетащить отдельный фрагмент текста. Дополнительные сведения см. в разделе ««CSS в AIR» на странице 1049».

## События перетаскивания в HTML

**Adobe AIR 1.0 и более поздних версий**

Ниже перечислены события, отправляемые элементом-инициатором, начинающим перетаскивание:

| Событие | Описание |
|---|---|
| dragstart | Отправляется, когда пользователь начинает перетаскивание. Обработчик этого события при необходимости может предотвратить перетаскивание, вызвав метод preventDefault() объекта события. Для управления возможностью копирования, создания ссылки или перемещения перетаскиваемого объекта необходимо задать свойство effectAllowed. Выделенный текст, изображения и ссылки по умолчанию помещаются в буфер обмена, но для жеста перетаскивания можно задать разные данные с помощью свойства dataTransfer объекта события. |
| drag | Непрерывно отправляется при выполнении перетаскивания. |
| dragend | Отправляется, когда пользователь отпускает кнопку мыши для завершения перетаскивания. |

Целевой элемент перетаскивания отправляет следующие события:

| Событие | Описание |
|---|---|
| dragover | Отправляется непрерывно, пока перетаскивание осуществляется в границах элемента. Обработчик этого события должен задавать свойство dataTransfer.dropEffect для указания действия, которое необходимо выполнить по окончании перетаскивания: копирование, создание ссылки или перемещение. |
| dragenter | Отправляется, когда указатель мыши, перетаскивающий данные, входит в границы элемента.<br>Если изменить какие-либо из свойств объекта dataTransfer в обработчике события dragenter, эти изменения будут отменены следующим же событием dragover. С другой стороны, между событиями dragenter и первым событием dragover, которое может заставить курсор мигать, если заданы разные свойства, существует небольшая задержка. Часто для обоих событий может использоваться один и тот же обработчик. |
| dragleave | Отправляется, когда указатель мыши, перетаскивающий данные, покидает границы элемента. |
| drop | Отправляется, когда пользователь отпускает данные над элементом. Доступ к перетаскиваемым данным возможен только в рамках обработчика этого события. |

Объект события, отправляемый в ответ на эти события, похож на событие mouse. Свойства события mouse, такие как (`clientX`, `clientY`) и (`screenX`, `screenY`), могут использоваться для определения положения мыши.

Самое важное свойство объекта события drag — это `dataTransfer`: в нем содержатся перетаскиваемые данные. Сам объект `dataTransfer` включает ряд свойств и методов:

| Свойство или метод | Описание |
|---|---|
| effectAllowed | Эффект, поддерживаемый целевым элементом перетаскивания. Как правило, это значение задается обработчиком события dragstart. См. раздел «Эффекты перетаскивания в HTML» на странице 665. |
| dropEffect | Эффект, выбранный пользователем или целевым элементом. Если задать свойство `dropEffect` в обработчике событий `dragover` или `dragenter`, то AIR обновляет вид курсора мыши, указывая, что произойдет после отпускания кнопки мыши. Если свойство `dropEffect` не соответствует ни одному из поддерживаемых эффектов, то завершить перетаскивание данных невозможно и курсор мыши приобретает вид *unavailable*. Если в ответ на последнее событие `dragover` или `dragenter` не задано свойство `dropEffect`, то пользователь может выбрать один из поддерживаемых эффектов с помощью стандартных системных клавиш-модификаторов.<br><br>Конечный эффект записывается в свойстве `dropEffect` объекта, отправляемого в ответ на `dragend`. Если пользователь отменяет перетаскивание, отпустив кнопку мыши вне границ целевого элемента, свойство `dropEffect` принимает значение `none`. |
| types | Массив, содержащий строковые значения типов MIME для каждого формата данных в объекте `dataTransfer`. |
| getData(mimeType) | Получает данные в формате, заданном параметром `mimeType`.<br><br>Метод `getData()` может быть вызван только в ответ на событие `drop`. |
| setData(mimeType) | Добавляет данные в `dataTransfer` в формате, заданном параметром mimeType. Могут быть добавлены данные в разных форматах. Для этого для каждого типа MIME вызывается метод `setData()`. Любые данные, помещенные в объект `dataTransfer` в результате поведения перетаскивания по умолчанию, стираются.<br><br>Метод `setData()` может быть вызван только в ответ на событие `dragstart`. |
| clearData(mimeType) | Стирает все данные в формате, заданном параметром `mimeType`. |
| setDragImage(image, offsetX, offsetY) | Задает заказное изображение для перетаскивания. Метод `setDragImage()` может быть вызван только в ответ на событие dragstart, и только когда весь элемент HTML перетаскивается путем задания значения `element` для его стиля CSS `-webkit-user-drag`. Параметр `image` может быть элементом сценария JavaScript или объектом Image. |

## Типы MIME при перетаскивании в HTML

**Adobe AIR 1.0 и более поздних версий**

С объектом `dataTransfer` события перетаскивания в HTML может использоваться ряд типов MIME:

| Формат данных | Тип MIME |
|---|---|
| Текст | «text/plain»; |
| HTML | «text/html»; |
| URL-адрес | «text/uri-list»; |
| Растровое изображение | «image/x-vnd.adobe.air.bitmap»; |
| Список файлов | «application/x-vnd.adobe.air.file-list». |

Также можно использовать и другие строки MIME, включая строки, определенные приложением. Однако другие приложения могут не распознать или не суметь использовать эти данные. Следите, чтобы данные, добавляемые к объекту `dataTransfer`, были нужного формата.

***Важная информация.*** *Доступ к отпущенным файлам имеет только код, выполняемый в изолированной программной среде. Попытка чтения или задания любого свойства объекта File вне изолированной программной среды приложения приводит к ошибке безопасности. Дополнительную информацию см. в разделе «Обработка перемещения файлов в изолированной программной среде HTML вне приложений» на странице 669.*

## Эффекты перетаскивания в HTML

**Adobe AIR 1.0 и более поздних версий**

Инициатор жеста перетаскивания может ограничивать эффекты путем задания свойства `dataTransfer.effectAllowed` в обработчике события `dragstart`. Ниже перечислены используемые строковые значения:

| Строковое значение | Описание |
|---|---|
| "none" | Операция перетаскивания запрещена. |
| "copy" | Данные копируются в целевой элемент, оригинал остается на месте. |
| "link" | Данные совместно используются исходным и целевым элементами, для чего применяется ссылка на оригинал. |
| "move" | Данные копируются в целевое место и удаляются из исходного. |
| "copyLink" | Данные можно копировать или создавать ссылку на них. |
| "copyMove" | Данные можно открывать или перемещать. |
| "linkMove" | Данные можно перемещать или создавать на них ссылку. |
| "all" | Данные можно копировать, перемещать или создавать на них ссылку. Эффект *All* выставляется автоматически, если отменяется поведение по умолчанию. |

Целевой элемент перетаскивания может задавать свойство `dataTransfer.dropEffect`, чтобы указать действие, выполняемое по завершении перетаскивания. Если одним из разрешенных действий является отпускание, то система отображает соответствующий курсор, означающий копирование, перемещение или создание ссылки. В противном случае отображается курсор *unavailable*. Если эффект отпускания не задан целевым элементом, пользователь может выбрать любое из разрешенных действий с помощью клавиш-модификаторов.

Значение `dropEffect` может быть задано в обработчиках событий `dragover` и `dragenter`:

```
function doDragStart(event) {
    event.dataTransfer.setData("text/plain","Text to drag");
    event.dataTransfer.effectAllowed = "copyMove";
}

function doDragOver(event) {
    event.dataTransfer.dropEffect = "copy";
}

function doDragEnter(event) {
    event.dataTransfer.dropEffect = "copy";
}
```

***Примечание.*** *Хотя свойство* `dropEffect` *всегда должно задаваться в обработчике события* `dragenter`*, помните, что следующее событие* `dragover` *сбрасывает значения свойства до исходных (по умолчанию). В ответ на оба событие необходимо задать* `dropEffect`*.*

# Перетаскивание данных из HTML-элемента

**Adobe AIR 1.0 и более поздних версий**

Поведение по умолчанию позволяет копировать большую часть HTML-содержимого перетаскиванием. Содержимое, которое можно перетаскивать, контролируется свойствами CSS `-webkit-user-select` и `-webkit-user-drag`.

Переопределите поведение жеста «перетаскивания из» по умолчанию в обработчике события `dragstart`. Вызовите метод `setData()` свойства `dataTransfer` объекта события для включения собственных данных в перетаскивание.

Если поведение по умолчанию не используется, то для указания поддерживаемых объектом-источником эффектов перетаскивания необходимо задать свойство `dataTransfer.effectAllowed` объекта события, отправляемого в ответ на `dragstart`. Может быть выбрано любое сочетание эффектов. Например, если элемент-источник поддерживает и эффект *copy*, и эффект *link*, задайте свойству значение `"copyLink"`.

## Настройка перетаскиваемых данных

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Добавьте данные для жеста перетаскивания в обработчик события `dragstart` с помощью свойства `dataTransfer`. С помощью метода `dataTransfer.setData()` данные помещаются в буфер обмена, при этом передается тип MIME и перемещаемые данные.

Например, если в приложении присутствует некое изображение с идентификатором *imageOfGeorge*, можно использовать показанный ниже обработчик события dragstart. В этом примере добавляются представления изображения в разных форматах данных, что увеличивает список приложений, которые смогут использовать перетаскиваемые данные.

```
function dragStartHandler(event){
    event.dataTransfer.effectAllowed = "copy";

    var dragImage = document.getElementById("imageOfGeorge");
    var dragFile = new air.File(dragImage.src);
    event.dataTransfer.setData("text/plain","A picture of George");
    event.dataTransfer.setData("image/x-vnd.adobe.air.bitmap", dragImage);
    event.dataTransfer.setData("application/x-vnd.adobe.air.file-list",
                               new Array(dragFile));
}
```

***Примечание.*** *При вызове метода* `setData()` *объекта* `dataTransfer` *поведение перетаскивания по умолчанию не подразумевает добавления каких-либо данных.*

# Перетаскивание данных в HTML-элемент

**Adobe AIR 1.0 и более поздних версий**

По умолчанию текст разрешается перетаскивать только в редактируемые области страницы. Можно указать, что элемент и его дочерние элементы являются редактируемыми, включив в открывающий тег элемента атрибут `contenteditable`. Можно сделать редактируемым весь документ. Для этого установите для свойства объекта документа `designMode` значение `"on"`.

Альтернативное поведение при перетаскивании на странице может быть задано с помощью обработки событий `dragenter`, `dragover` и `drop` для любых элементов, принимающих перетаскиваемые данные.

## Разрешение жеста «перетаскивания в»

**Adobe AIR 1.0 и более поздних версий**

Для обработки жеста «перетаскивания в» необходимо сначала отменить поведение по умолчанию. На любых HTML-элементах, которые используются в качестве целевых при перетаскивании, необходимо прослушивать события `dragenter` и `dragover`. В обработчиках этих событий вызывается метод `preventDefault()` отправляемого объекта события. Отмена поведения по умолчанию позволяет перетаскивать данные и в нередактируемые области.

## Получение перетаскиваемых данных

**Adobe AIR 1.0 и более поздних версий**

Перетаскиваемые данные могут использоваться в обработчике события `ondrop`:

```
 function doDrop(event){
    droppedText = event.dataTransfer.getData("text/plain");
}
```

С помощью метода `dataTransfer.getData()` данные могут считываться в буфер обмена путем передачи типа MIME формата считываемых данных. Узнать, какие форматы данных доступны, можно с помощью свойства `types` объекта `dataTransfer`. Массив `types` содержит строку типа MIME для каждого из доступных форматов.

Если поведение по умолчанию в событиях dragenter и dragover отменяется, разработчику необходимо самостоятельно вставить все перетаскиваемые данные в нужное место документа. Не существует API-интерфейса, который позволял бы преобразовать положение мыши в точку вставки внутри элемента. Это ограничение может осложнить реализацию жестов перетаскивания со вставкой.

# Пример: переопределение поведения по умолчанию при перетаскивании в HTML

**Adobe AIR 1.0 и более поздних версий**

В этом примере показана реализация целевого элемента перетаскивания, выводящего таблицу, в которой перечислены все поддерживаемые перетаскиваемым элементом форматы данных.

Поведение по умолчанию разрешает перетаскивать текст, ссылки и изображения в границах приложения. В примере поведение по умолчанию при перетаскивании переопределяется для элемента div, который служит целевым элементом перетаскивания. При разрешении нередактируемым областям принимать жесты «перетаскивания в» ключевым шагом является вызов метода `preventDefault()` объекта события, отправляемого в ответ на события `dragenter` и `dragover`. В ответ на событие `drop` обработчик преобразует передаваемые данные в ряд в формате HTML и вставляет этот ряд (строку) в отображаемую таблицу.

```
 <html>
<head>
<title>Drag-and-drop</title>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>
<script language="javascript">
    function init(){
        var target = document.getElementById('target');
        target.addEventListener("dragenter", dragEnterOverHandler);
        target.addEventListener("dragover", dragEnterOverHandler);
        target.addEventListener("drop", dropHandler);

        var source = document.getElementById('source');
        source.addEventListener("dragstart", dragStartHandler);
        source.addEventListener("dragend", dragEndHandler);

        emptyRow = document.getElementById("emptyTargetRow");
    }

    function dragStartHandler(event){
        event.dataTransfer.effectAllowed = "copy";
    }

    function dragEndHandler(event){
        air.trace(event.type + ": " + event.dataTransfer.dropEffect);
    }

    function dragEnterOverHandler(event){
        event.preventDefault();
    }

    var emptyRow;
    function dropHandler(event){
        for(var prop in event){
            air.trace(prop + " = " + event[prop]);
        }
        var row = document.createElement('tr');
        row.innerHTML = "<td>" + event.dataTransfer.getData("text/plain") + "</td>" +
            "<td>" + event.dataTransfer.getData("text/html") + "</td>" +
            "<td>" + event.dataTransfer.getData("text/uri-list") + "</td>" +
            "<td>" + event.dataTransfer.getData("application/x-vnd.adobe.air.file-list") +
            "</td>";

        var imageCell = document.createElement('td');
        if((event.dataTransfer.types.toString()).search("image/x-vnd.adobe.air.bitmap") > -
1){
            imageCell.appendChild(event.dataTransfer.getData("image/x-
vnd.adobe.air.bitmap"));
        }
        row.appendChild(imageCell);
```

```
        var parent = emptyRow.parentNode;
        parent.insertBefore(row, emptyRow);
    }
</script>
</head>
<body onLoad="init()" style="padding:5px">
<div>
    <h1>Source</h1>
       <p>Items to drag:</p>
    <ul id="source">
        <li>Plain text.</li>
           <li>HTML <b>formatted</b> text.</li>
           <li>A <a href="http://www.adobe.com">URL.</a></li>
           <li><img src="icons/AIRApp_16.png" alt="An image"/></li>
           <li style="-webkit-user-drag:none;">
        Uses "-webkit-user-drag:none" style.
           </li>
           <li style="-webkit-user-select:none;">
        Uses "-webkit-user-select:none" style.
           </li>

       </ul>
</div>
<div id="target" style="border-style:dashed;">
    <h1 >Target</h1>
       <p>Drag items from the source list (or elsewhere).</p>
       <table id="displayTable" border="1">
        <tr><th>Plain text</th><th>Html text</th><th>URL</th><th>File list</th><th>Bitmap
Data</th></tr>
           <tr
id="emptyTargetRow"><td> </td><td> </td><td> </td><td> </td><td> </
td></tr>
       </table>
       </div>
</div>
</body>
</html>
```

# Обработка перемещения файлов в изолированной программной среде HTML вне приложений

**Adobe AIR 1.0 и более поздних версий**

Содержимое извне приложения не имеет доступа к объектам File, получаемым при перетаскивании файлов в приложение AIR. Нельзя и передать один из этих объектов File содержимому приложения через мост изолированной программной среды. (В процессе сериализации доступ к свойствам объекта должен быть открыт.) Тем не менее файлы по-прежнему можно перетащить в приложение путем прослушивания событий nativeDragDrop в AIR на объекте HTMLLoader.

Обычно, если пользователь перетаскивает файл во фрейм с содержимым, не относящимся к приложению, событие drop не переносится с дочернего элемента на родительский. Однако, так как события, отправляемые HTMLLoader (который является контейнером для всего HTML-содержимого в приложении AIR), не являются частью потока событий HTML, событие drop по-прежнему можно получать в содержимом приложения.

Для получения события отпускания файла родительский документ добавляет прослушиватель события к объекту HTMLLoader, используя ссылку `window.htmlLoader`:

```
 window.htmlLoader.addEventListener("nativeDragDrop",function(event){
    var filelist = event.clipboard.getData(air.ClipboardFormats.FILE_LIST_FORMAT);
    air.trace(filelist[0].url);
});
```

В примере ниже используется родительский документ, загружающий дочернюю страницу в удаленную изолированную программную среду (http://localhost/). Родительский документ прослушивает объект HTMLLoader на предмет события `nativeDragDrop` и получает URL-адрес файла.

```
 <html>
<head>
<title>Drag-and-drop in a remote sandbox</title>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>
<script language="javascript">
    window.htmlLoader.addEventListener("nativeDragDrop",function(event){
        var filelist = event.clipboard.getData(air.ClipboardFormats.FILE_LIST_FORMAT);
        air.trace(filelist[0].url);
    });
</script>
</head>
<body>
       <iframe src="child.html"
               sandboxRoot="http://localhost/"
               documentRoot="app:/"
               frameBorder="0"  width="100%" height="100%">
       </iframe>
</body>
</html>
```

Дочерний документ должен представлять действительный целевой элемент перетаскивания путем вызова метода `preventDefault()` объекта Event в обработчиках событий HTML `dragenter` и `dragover`. В противном случае событие перетаскивания может никогда не произойти.

```
 <html>
<head>
    <title>Drag and drop target</title>
    <script language="javascript" type="text/javascript">
        function preventDefault(event){
            event.preventDefault();
        }
    </script>
</head>
<body ondragenter="preventDefault(event)" ondragover="preventDefault(event)">
<div>
<h1>Drop Files Here</h1>
</div>
</body>
</html>
```

# Перетаскивание предварительных образов файла

**Adobe AIR 2 и более поздних версий**

Предварительный образ файла представляет собой формат буфера перетаскивания, который дает пользователю возможность перетащить пока не существующий файл за пределы программы AIR. Например, с помощью предварительных образов файла ваша программа может дать пользователю возможность перетащить прокси-значок в папку на рабочем столе. Прокси-значок представляет файл или данные, о которых известно, что они доступны по определенному URL. После отпускания значка пользователем среда выполнения загружает данные и записывает файл в местоположение перетаскивания.

Благодаря классу URLFilePromise в приложении AIR можно перетащить файлы, которые доступны по URL. Реализация URLFilePromise предоставляется в библиотеке aircore как компонент пакета AIR 2 SDK. Используйте файл aircore.swc или aircore.swf, находящийся в каталоге frameworks/libs/air пакета SDK.

Как вариант можно реализовать собственный алгоритм предоставления файла с использованием интерфейса IFilePromise (который задан в пакете flash.desktop среды выполнения).

Предварительные образы файла, в принципе, похожи на отложенную визуализацию с помощью функции обработки данных в буфере обмена. При перетаскивании файлов используйте предварительные образы файла вместо отложенной визуализации. Метод отложенной визуализации может привести к появлению нежелательных пауз в жесте перетаскивания из-за генерирования или загрузки данных. Используйте отложенную визуализацию во время операций копирования и вставки (где не поддерживаются предварительные образы файла).

### Ограничения при использовании предварительных образов файла

По сравнению с другими форматами данных, которые можно поместить в буфер обмена при перетаскивании, предварительные образы файла имеют следующие ограничения:

- Предварительные образы файла можно перетаскивать только из приложения AIR; в приложение AIR их перетаскивать нельзя.
- Предварительные образы файла не поддерживаются во всех операционных системах. Воспользуйтесь свойством `Clipboard.supportsFilePromise`, чтобы проверить, поддерживаются ли предварительные образы файла в системе хоста. В системах, не поддерживающих предварительные образы файла, необходимо предусмотреть альтернативный механизм загрузки или генерирования данных файла.
- Предварительные образы файла нельзя применять в буфере обмена для копирования и вставки (`Clipboard.generalClipboard`).

## Дополнительные разделы справки

flash.desktop.IFilePromise

air.desktop.URLFilePromise

## Перетаскивание удаленных файлов

**Adobe AIR 2 и более поздних версий**

Воспользуйтесь классом URLFilePromise для создания объектов предварительных образов файла, представляющих файлы или данные, которые доступны по URL. Добавьте один или несколько объектов предварительных образов файла с помощью формата буфера данных `FILE_PROMISE_LIST`. В следующем примере единичный файл, который доступен по адресу http://www.example.com/foo.txt, загружается и сохраняется в местоположение перетаскивания под именем bar.txt. Имена удаленного и локального файлов могут не совпадать.

```
if( Clipboard.supportsFilePromise )
{
    var filePromise:URLFilePromise = new URLFilePromise();
    filePromise.request = new URLRequest("http://example.com/foo.txt");
    filePromise.relativePath = "bar.txt";

    var fileList:Array = new Array( filePromise );
    var clipboard:Clipboard = new Clipboard();
    clipboard.setData( ClipboardFormats.FILE_PROMISE_LIST_FORMAT, fileList );
    NativeDragManager.doDrag( dragSource, clipboard );
}
```

Можно разрешить пользователю перетаскивать более одного файла за один раз путем добавления дополнительных объектов предварительных образов файла в массив, присвоенный буферу обмена. Также можно указать подкаталоги в свойстве `relativePath`, чтобы поместить некоторые или все включенные в операцию файлы в подпапку местоположения перетаскивания.

В следующем примере показано, как запустить операцию перетаскивания, включающую несколько предварительных образов файла. В этом примере HTML-страница *article.html* помещается в буфер данных в качестве предварительного образа файла вместе с двумя связанными с ней файлами изображений. Изображения копируются в подпапку *images*, чтобы сохранить относительные ссылки.

```
if( Clipboard.supportsFilePromise )
{   //Create the promise objects
    var filePromise:URLFilePromise = new URLFilePromise();
    filePromise.request = new URLRequest("http://example.com/article.html");
    filePromise.relativePath = "article.html";

    var image1Promise:URLFilePromise = new URLFilePromise();
    image1Promise.request = new URLRequest("http://example.com/images/img_1.jpg");
    image1Promise.relativePath = "images/img_1.html";
    var image2Promise:URLFilePromise = new URLFilePromise();
    image2Promise.request = new URLRequest("http://example.com/images/img_2.jpg");
    image2Promise.relativePath = "images/img_2.jpg";


    //Put the promise objects onto the clipboard inside an array
    var fileList:Array = new Array( filePromise, image1Promise, image2Promise );
    var clipboard:Clipboard = new Clipboard();
    clipboard.setData( ClipboardFormats.FILE_PROMISE_LIST_FORMAT, fileList );
    //Start the drag operation
    NativeDragManager.doDrag( dragSource, clipboard );
}
```

## Реализация интерфейса IFilePromise

**Adobe AIR 2 и более поздних версий**

Чтобы предоставить предварительные образы файла для ресурсов, доступ к которым с помощью объекта URLFilePromise невозможен, можно реализовать интерфейс IFilePromise в пользовательском классе. Интерфейс IFilePromise определяет методы и свойства, используемые средой выполнения AIR для доступа к данным, которые необходимо записать в файл после перетаскивания предварительного образа файла.

Реализация IFilePromise передает другой объект среде выполнения AIR, которая предоставляет данные для предварительного образа файла. Этот объект должен реализовать интерфейс IDataInput, который применяется средой выполнения AIR для чтения данных. Например, класс URLFilePromise, реализующий IFilePromise, использует в качестве поставщика данных объект URLStream.

AIR может синхронно или асинхронно считывать данные. Реализация IFilePromise сообщает о поддерживаемом режиме доступа путем возврата соответствующего значения в свойство `isAsync`. Если предоставляется асинхронный доступ к данным, объект поставщика данных должен реализовать интерфейс IEventDispatcher и отправить необходимые события (такие как `open`, `progress` и `complete`).

Можно воспользоваться пользовательским классом, либо одним из следующих встроенных классов в качестве поставщика данных для предварительного образа файла:

- ByteArray (синхронный)
- FileStream (синхронный или асинхронный)
- Socket (асинхронный)
- URLStream (асинхронный)

Чтобы реализовать интерфейс IFilePromise, необходимо предоставить код для следующих функций и свойств.

- `open():IDataInput` — возвращает объект поставщика данных, из которого считываются данные для файла. Объект должен реализовывать интерфейс IDataInput. Если данные предоставляются асинхронно, объект должен также реализовывать интерфейс IEventDispatcher и отправлять необходимые события (см. раздел «Использование асинхронного поставщика данных в предварительном образе файла» на странице 675).
- `get relativePath():String` — указывает путь создаваемого файла, включая имя файла. Этот путь разрешается относительно местоположения перетаскивания, выбранного пользователем во время операции перетаскивания. Чтобы убедиться в том, что в пути используется правильный символ разделителя для операционной системы хоста, воспользуйтесь константой `File.separator` при указании путей с каталогами. Чтобы дать возможность задавать путь во время выполнения, можно добавить функцию установщика или воспользоваться параметром конструктора.
- `get isAsync():Boolean` — сообщает среде выполнения AIR о синхронном или асинхронном предоставлении данных объектом поставщика данных.
- `close():void` — вызывается средой выполнения, когда данные полностью считаны (или при возникновении ошибки, предотвращающей дальнейшее чтение). Эту функцию можно применять для очистки ресурсов.
- `reportError( e:ErrorEvent ):void` — вызывается средой выполнения при появлении ошибки чтения данных.

Все методы IFilePromise вызываются средой выполнения во время операции перетаскивания, включающей предварительный образ файла. Как правило, ваша логика приложения не должна вызывать какие-либо из этих методов напрямую.

## Использование синхронного поставщика данных в предварительном образе файла

**Adobe AIR 2 и более поздних версий**

Самый простой способ реализации интерфейса IFilePromise состоит в том, чтобы использовать объект синхронного поставщика данных (например, ByteArray или синхронный FileStream). В следующем примере создается заполненный данными объект ByteArray, который возвращается при вызове метода `open()`.

```
package
{
    import flash.desktop.IFilePromise;
    import flash.events.ErrorEvent;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class SynchronousFilePromise implements IFilePromise
    {
        private const fileSize:int = 5000; //size of file data
        private var filePath:String = "SynchronousFile.txt";

        public function get relativePath():String
        {
            return filePath;
        }

        public function get isAsync():Boolean
        {
            return false;
        }

        public function open():IDataInput
        {
            var fileContents:ByteArray = new ByteArray();

            //Create some arbitrary data for the file
            for( var i:int = 0; i < fileSize; i++ )
            {
                fileContents.writeUTFBytes( 'S' );
            }

            //Important: the ByteArray is read from the current position
            fileContents.position = 0;
            return fileContents;
        }

        public function close():void
        {
            //Nothing needs to be closed in this case.
        }

        public function reportError(e:ErrorEvent):void
        {
            trace("Something went wrong: " + e.errorID + " - " + e.type + ", " + e.text );
        }
    }
}
```

На практике синхронные предварительные образы файла имеют ограниченное применение. Если данных мало, можно с легкостью создать файл во временном каталоге и добавить обычный массив со списком файлов в буфер обмена при перетаскивании. С другой стороны, если объем данных велик или генерирование данных требует больших вычислительных ресурсов, требуется продолжительный синхронный процесс. Продолжительные синхронные процессы могут блокировать обновления пользовательского интерфейса в течение заметного периода времени и создавать впечатление «зависания» программы. Во избежание появления этой проблемы можно создать асинхронный поставщик данных, который включается таймером.

## Использование асинхронного поставщика данных в предварительном образе файла

**Adobe AIR 2 и более поздних версий**

При использовании объекта асинхронного поставщика данных свойству `isAsync` интерфейса IFilePromise должно быть присвоено значение `true`, а объект, возвращенный методом `open()`, должен реализовать интерфейс IEventDispatcher. Среда выполнения прослушивает несколько альтернативных событий, чтобы в качестве поставщика данных можно было использовать разные встроенные объекты. Например, события `progress` отправляются объектами FileStream и URLStream, а события `socketData` отправляются объектами Socket. Среда выполнения прослушивает соответствующие события из всех этих объектов.

Процесс считывания данных из объекта поставщика данных зависит от следующих событий.

- Event.OPEN — сообщает среде выполнения о готовности источника данных.
- ProgressEvent.PROGRESS — сообщает среде выполнения о доступности данных. Среда выполнения считает объем доступных данных из объекта поставщика данных.
- ProgressEvent.SOCKET_DATA — сообщает среде выполнения о доступности данных. Событие `socketData` отправляется объектами на базе сокета. Для других типов объектов необходимо отправить событие `progress`. Среда выполнения прослушивает оба события, чтобы определить, когда можно прочесть данные.
- Event.COMPLETE — сообщает среде выполнения о том, что все данные считаны.
- Event.CLOSE — сообщает среде выполнения о том, что все данные считаны. Для этой цели среда выполнения прослушивает как событие `close`, так и событие `complete`.
- IOErrorEvent.IOERROR — сообщает среде выполнения о появлении ошибки чтения данных. Среда выполнения останавливает создание файла и вызывает метод IFilePromise `close()`.
- SecurityErrorEvent.SECURITY_ERROR — сообщает среде выполнения о появлении ошибки безопасности. Среда выполнения останавливает создание файла и вызывает метод IFilePromise `close()`.
- HTTPStatusEvent.HTTP_STATUS — используется средой выполнения вместе с `httpResponseStatus`, чтобы убедиться в том, что доступные данные отражают нужное содержимое, а не сообщение об ошибке (такое как «страница 404»). Это событие должны отправлять объекты, основанные на протоколе HTTP.
- HTTPStatusEvent.HTTP_RESPONSE_STATUS — используется средой выполнения вместе с `httpStatus`, чтобы убедиться в том, что доступные данные отражают нужное содержимое. Это событие должны отправлять объекты, основанные на протоколе HTTP.

Поставщик данных должен отправлять эти события в следующей последовательности:

1. событие `open`
2. события `progress` или `socketData`,
3. событие `complete` или `close`.

***Примечание.*** *Встроенные объекты (FileStream, Socket и URLStream) автоматически отправляют соответствующие события.*

В следующем примере предварительный образ файла создается с помощью пользовательского асинхронного поставщика данных. Класс поставщика данных является расширением ByteArray (для поддержки IDataInput) и реализует интерфейс IEventDispatcher. При каждом событии таймера объект генерирует порцию данных и отправляет событие progress, чтобы сообщить среде выполнения о доступности данных. После получения достаточного объема данных объект отправляет событие complete.

```
package
{
import flash.events.Event;
import flash.events.EventDispatcher;
import flash.events.IEventDispatcher;
import flash.events.ProgressEvent;
import flash.events.TimerEvent;
import flash.utils.ByteArray;
import flash.utils.Timer;

[Event(name="open", type="flash.events.Event.OPEN")]
[Event(name="complete",  type="flash.events.Event.COMPLETE")]
[Event(name="progress", type="flash.events.ProgressEvent")]
[Event(name="ioError", type="flash.events.IOErrorEvent")]
[Event(name="securityError", type="flash.events.SecurityErrorEvent")]
public class AsyncDataProvider extends ByteArray implements IEventDispatcher
{
    private var dispatcher:EventDispatcher = new EventDispatcher();
    public var fileSize:int = 0; //The number of characters in the file
    private const chunkSize:int = 1000; //Amount of data written per event
    private var dispatchDataTimer:Timer = new Timer( 100 );
    private var opened:Boolean = false;

    public function AsyncDataProvider()
    {
        super();
        dispatchDataTimer.addEventListener( TimerEvent.TIMER, generateData );
    }

    public function begin():void{
        dispatchDataTimer.start();
    }

    public function end():void
    {
        dispatchDataTimer.stop();
    }
    private function generateData( event:Event ):void
    {
        if( !opened )
        {
            var open:Event = new Event( Event.OPEN );
            dispatchEvent( open );
            opened = true;
        }
        else if( position + chunkSize < fileSize )
        {
            for( var i:int = 0; i <= chunkSize; i++ )
            {
                writeUTFBytes( 'A' );
```

```
            }
            //Set position back to the start of the new data
            this.position -= chunkSize;
            var progress:ProgressEvent =
                new ProgressEvent( ProgressEvent.PROGRESS, false, false, bytesAvailable,
bytesAvailable + chunkSize);
            dispatchEvent( progress )
        }
        else
        {
            var complete:Event = new Event( Event.COMPLETE );
            dispatchEvent( complete );
        }
    }
     //IEventDispatcher implementation
    public function addEventListener(type:String, listener:Function,
useCapture:Boolean=false, priority:int=0, useWeakReference:Boolean=false):void
    {
        dispatcher.addEventListener( type, listener, useCapture, priority, useWeakReference );
    }

    public function removeEventListener(type:String, listener:Function,
useCapture:Boolean=false):void
    {
        dispatcher.removeEventListener( type, listener, useCapture );
    }

    public function dispatchEvent(event:Event):Boolean
    {
        return dispatcher.dispatchEvent( event );
    }

    public function hasEventListener(type:String):Boolean
    {
        return dispatcher.hasEventListener( type );
    }

    public function willTrigger(type:String):Boolean
    {
        return dispatcher.willTrigger( type );
    }
}
}
```

***Примечание.*** *Поскольку класс AsyncDataProvider в данном примере является расширением ByteArray, он не может также являться расширением EventDispatcher. Чтобы реализовать интерфейс IEventDispatcher, класс использует внутренний объект EventDispatcher и перенаправляет вызовы метода IEventDispatcher в этот внутренний объект. Можно также расширить EventDispatcher и реализовать IDataInput (или реализовать оба интерфейса).*

Асинхронная реализация интерфейса IFilePromise почти идентична синхронной реализации. Основные различия состоят в том, что `isAsync` возвращает `true` и что метод `open()` возвращает объект асинхронных данных:

```
package
{
    import flash.desktop.IFilePromise;
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.utils.IDataInput;

    public class AsynchronousFilePromise extends EventDispatcher implements IFilePromise
    {
        private var fileGenerator:AsyncDataProvider;
        private const fileSize:int = 5000; //size of file data
        private var filePath:String = "AsynchronousFile.txt";

        public function get relativePath():String
        {
            return filePath;
        }

        public function get isAsync():Boolean
        {
            return true;
        }

        public function open():IDataInput
        {
            fileGenerator = new AsyncDataProvider();
            fileGenerator.fileSize = fileSize;
            fileGenerator.begin();
            return fileGenerator;
        }

        public function close():void
        {
            fileGenerator.end();
        }

        public function reportError(e:ErrorEvent):void
        {
            trace("Something went wrong: " + e.errorID + " - " + e.type + ", " + e.text );
        }
    }
}
```

# Глава 36. Работа с меню

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для модификации контекстных меню в веб-приложениях Flex и Flash Player используйте классы в контекстном меню API-интерфейса.

Для определения меню программ, окон, контекстных и всплывающих меню в Adobe® AIR® используйте классы API-интерфейса.

## Основные сведения о меню

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Краткое разъяснение и примеры кодов с созданием собственных меню в приложениях AIR см. в следующих статьях по быстрому началу работы в центре Adobe Developer Connection:

- Добавление встроенных меню в приложение AIR (Flex)
- Добавление встроенных меню в приложение AIR (Flash)

Классы исходных меню обеспечивают доступ к функциям исходных меню операционной системы, в которой выполняется приложение. Объекты NativeMenu могут применяться для создания меню приложения (в Mac OS X), меню окна (в Windows и Linux), контекстных и всплывающих меню.

Вне AIR вы можете использовать классы контекстного меню для преобразования контекстного меню, которое Flash Player автоматически отображает, когда пользователь щелкает правой кнопкой или щелкает с удержанием клавиши «Command» на объекте вашего приложения. (Автоматическое контекстное меню для приложений AIR не отображается.)

### Классы меню

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Классы меню перечислены ниже.

| Пакет | Классы |
|---|---|
| flash.display | • NativeMenu<br>• NativeMenuItem |
| flash.ui | • ContextMenu<br>• ContextMenuItem |
| flash.events | • Event<br>• ContextMenuEvent |

## Разновидности меню

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Ниже перечислены поддерживаемые AIR типы меню.

**Контекстные меню** Контекстные меню открываются по щелчку правой кнопкой или щелчку с удерживанием клавиши «Command» по интерактивному объекту в SWF-содержимом или по элементу HTML-содержимого.

В среде выполнения Flash Player контекстное меню отображается автоматически. Для добавления в меню своих собственных команд можно использовать классы ContextMenu и ContextMenuItem. Можно также удалять некоторые, хотя и не все, встроенные команды.

В среде выполнения AIR контекстные меню можно создавать с помощью классов NativeMenu или ContextMenu. В HTML-содержимом в среде AIR для добавления контекстного меню к элементам можно применять прикладные интерфейсы программирования WebKit HTML и JavaScript.

**Меню программы (только для AIR)** Меню программы — это глобальное меню, которое работает в рамках всего приложения. Меню программы поддерживаются в Mac OS X и не поддерживаются в Windows и Linux. Операционная система Mac OS X автоматически создает меню программы. С помощью API-интерфейса меню AIR в стандартные меню можно добавлять пункты и подменю. Можно также добавить прослушиватели для существующих команд меню. Также можно удалить существующие пункты.

**Меню окон (только для AIR)** Меню окна связано с отдельным окном и отображается под строкой заголовка. Меню добавляется в окно путем создания объекта NativeMenu и его назначения свойству `menu` объекта NativeWindow. Меню окон поддерживаются в ОС Windows и Linux, но не поддерживаются в Mac OS X. Исходные меню можно использовать только с окнами, имеющими системный Chrome.

**Меню значков Dock и области уведомлений (только для AIR)** Эти меню значков схожи с контекстными меню и назначаются значку приложения на панели Dock в Mac OS X или в областях уведомления на панели задач в Windows. Для создания меню значков Dock и области уведомлений используется класс NativeMenu. В Mac OS X элементы меню добавляются над стандартными системными элементами. В Windows или Linux стандартное меню отсутствует.

**Всплывающие меню (только для AIR)** Всплывающие меню AIR напоминают контекстные, но не всегда связаны с конкретным объектом или компонентом приложения. Всплывающие меню могут выводиться в любой части окна в результате вызова метода `display()` любого объекта NativeMenu.

**Заказные меню** Исходные меню отрисовываются исключительно операционной системой и поэтому не входят в модели визуализации Flash и HTML. Вместо исходных меню всегда можно создать свои собственные пользовательские меню, использовав для этого языки MXML, ActionScript или сценарий JavaScript (в AIR). Такие меню должны полностью визуализироваться внутри содержимого программы.

**меню Flex** Инфраструктура Adobe® Flex™ предоставляет набор компонентов меню Flex. Меню Flex рисует среда выполнения, а не операционная система, они не являются *исходными*. Меню Flex может использоваться в окнах, не имеющих системного Chrome. Другим преимуществом меню Flex является то, что их можно декларативно определять в формате MXML. Если используется инфраструктура Flex Framework, создавать меню окон рекомендуется с помощью классов меню Flex, а не исходных классов.

### Меню по умолчанию (только для AIR)

Ниже перечислены меню по умолчанию, предоставляемые операционной системой или встроенным классом AIR:

- Меню программы в Mac OS X
- Меню значка Dock в Mac OS X
- Контекстное меню для выделенного текста или изображения в HTML-содержимом

- Контекстное меню для выделенного текста в объекте TextField (или расширении объекта TextField)

## О контекстных меню

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В SWF-содержимом любой объект-наследник InteractiveObject может иметь контекстное меню. Для этого нужно назначить объект меню его свойству `contextMenu`. Некоторые команды включены по умолчанию, например «Вперед», «Назад», «Печать», «Качество» и «Масштаб». В среде выполнения AIR объект меню, назначенный свойству `contextMenu`, может относиться к типу NativeMenu или ContextMenu. В среде выполнения Flash Player доступен только класс ContextMenu.

Можно выполнять прослушивание либо событий исходных меню, либо событий контекстных меню, если используются классы ContextMenu и ContextMenuItem; при этом будут создаваться и те и другие события. Одним из преимуществ, обеспечиваемых свойствами объекта ContextMenuEvent, является то, что свойство contextMenuOwner идентифицирует объект, к которому прикреплено меню, а свойство`mouseTarget` идентифицирует объект, который щелкнули для открытия меню. Эта информация не доступна из объекта NativeMenuEvent.

В примере ниже показано, как создать спрайт и добавить простое контекстное меню правки:

```
 var sprite:Sprite = new Sprite();
sprite.contextMenu = createContextMenu()
private function createContextMenu():ContextMenu{
    var editContextMenu:ContextMenu = new ContextMenu();
    var cutItem:ContextMenuItem = new ContextMenuItem("Cut")
    cutItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doCutCommand);
    editContextMenu.customItems.push(cutItem);

    var copyItem:ContextMenuItem = new ContextMenuItem("Copy")
    copyItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doCopyCommand);
    editContextMenu.customItems.push(copyItem);

    var pasteItem:ContextMenuItem = new ContextMenuItem("Paste")
    pasteItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doPasteCommand);
    editContextMenu.customItems.push(pasteItem);

    return editContextMenu
}
private function doCutCommand(event:ContextMenuEvent):void{trace("cut");}
private function doCopyCommand(event:ContextMenuEvent):void{trace("copy");}
private function doPasteCommand(event:ContextMenuEvent):void{trace("paste");}
```

***Примечание.*** *В отличие от SWF-содержимого, отображаемого в среде обозревателя, контекстные меню в среде AIR не имеют встроенных команд.*

### Настройка контекстного меню Flash Player

В браузере или проекторе контекстные меню в SWF-содержимом всегда содержат встроенные элементы. Из меню можно удалить все команды по умолчанию, кроме команд «Настройки» и «О программе». Если задать свойству рабочей области `showDefaultContextMenu` значение `false`, эти команды будут удалены из контекстного меню.

Для создания персонализированного контекстного меню для конкретного экранного объекта создайте новый экземпляр класса ContextMenu, вызовите метод `hideBuiltInItems()` и назначьте этот экземпляр свойству `contextMenu` этого экземпляра DisplayObject. В примере ниже показан динамически рисуемый квадрат с командой контекстного меню, позволяющей изменить его цвет:

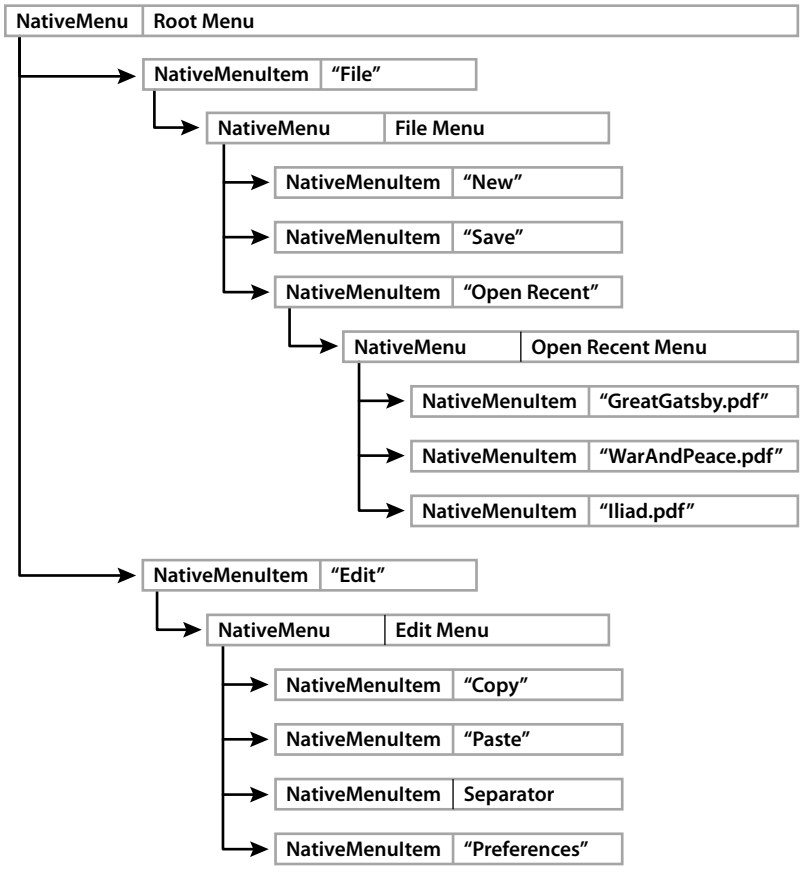
```
var square:Sprite = new Sprite();
square.graphics.beginFill(0x000000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.x =
square.y = 10;
addChild(square);

var menuItem:ContextMenuItem = new ContextMenuItem("Change Color");
menuItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT,changeColor);
var customContextMenu:ContextMenu = new ContextMenu();
customContextMenu.hideBuiltInItems();
customContextMenu.customItems.push(menuItem);
square.contextMenu = customContextMenu;

function changeColor(event:ContextMenuEvent):void
{
    square.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    return new ColorTransform(Math.random(), Math.random(), Math.random(),1,(Math.random() *
512) - 255, (Math.random() * 512) -255, (Math.random() * 512) - 255, 0);
}
```

## Структура исходного меню (AIR)

**Adobe AIR 1.0 и более поздних версий**

Исходные меню по своей природе имеют иерархическую структуру. Объекты NativeMenu содержат дочерние объекты NativeMenuItem. Объекты NativeMenuItem, представляющие подменю, в свою очередь, могут содержать объекты NativeMenu. Объект меню верхнего (корневого) уровня представляет строку меню программы или окна. (У контекстных и всплывающих меню, а также меню значков нет строки меню.)

На схеме ниже показана типичная структура меню. Корневое меню — это строка меню, содержащая два пункта, указывающие на подменю *Файл* и *Правка*. Подменю «Файл» в этой структуре содержит две команды и пункт, ссылающийся на подменю *Открыть последний файл*, которое содержит еще три пункта. Подменю «Правка» содержит три команды и разделитель.




Для определения подменю необходимы объекты NativeMenu и NativeMenuItem. Объект NativeMenuItem определяет текст заголовка, отображаемый в родительском меню, и позволяет пользователю открывать подменю. Объект NativeMenu служит контейнером для элементов подменю. Объект NativeMenuItem ссылается на объект NativeMenu через свойство `submenu` объекта NativeMenuItem.

Пример кода, с помощью которого создается это меню, см. в разделе ««Пример собственного меню: меню окна и приложения (AIR)» на странице 691».

## События меню

**Adobe AIR 1.0 и более поздних версий**

Объекты NativeMenu и NativeMenuItem отправляют события `preparing`, `displaying` и `select`:

**Preparing:** каждый раз при начале взаимодействия объекта с пользователем меню и элементы меню отправляют событие `preparing` всем зарегистрированным прослушивателям. Взаимодействие включает открытие меню и выбор элемента меню с помощью клавиатурных сокращений.

***Примечание.*** *Событие* `preparing` *доступно только в Adobe AIR 2.6 и более поздних версиях.*

**Displaying (отображение):** Непосредственно перед отображением меню оно само и его элементы отправляют событие `displaying` всем зарегистрированным прослушивателям.

События `preparing` и `displaying` дают возможность обновить содержимое меню или вид пункта меню перед его выводом на экран. Например, в прослушивателе события `displaying` меню «Открыть последний файл» можно изменить пункты меню так, чтобы они отображали список недавно просмотренных документов.

Если элемент меню, клавиатурное сокращение которого вызвало событие `preparing`, был удален, взаимодействие меню отменяется, и событие `select` не отправляется.

Свойства события `target` и `currentTarget` являются объектами, для которых зарегистрирован прослушиватель (меню или один из его пунктов).

Событие `preparing` отправляется перед событием `displaying`. Обычно выполняется прослушивание только одного из событий.

**Выделить:** Когда пользователь выбирает в меню пункт-команду, этот пункт отправляет событие `select` всем зарегистрированным прослушивателям. Подменю и разделитель нельзя выбрать, поэтому эти элементы не могут отправлять событие `select`.

Событие `select` передается вверх по иерархии от пункта меню до его родительского меню или до корневого меню. Событие `select` можно прослушивать непосредственно на элементе меню, а также можно прослушивать структуру меню вверх по иерархии. При прослушивании события `select` в меню можно определить выбранный элемент с помощью свойства `target` события. Так как событие передается вверх по иерархии меню, свойство `currentTarget` объекта события определяет текущий объект меню.

***Примечание.*** *Объекты ContextMenu и ContextMenuItem отправляют события* `menuItemSelect` *и* `menuSelect`*, а также события* `select`*,* `preparing` *и* `displaying`*.*

## Клавишные эквиваленты для команд исходного меню (AIR)

**Adobe AIR 1.0 и более поздних версий**

Команде меню можно назначить клавишный эквивалент (иногда называемый ускорителем). Пункт меню отправляет событие `select` всем зарегистрированным прослушивателям, когда пользователь нажимает клавишу или комбинацию клавиш. Чтобы команда сработала, меню, содержащее элемент, должно быть частью меню программы или активного окна.

Клавишные эквиваленты состоят из двух частей: строки, представляющей основную клавишу, и массива клавиш-модификаторов, которые также должны быть нажаты. Чтобы назначить основную клавишу, задайте для свойства `keyEquivalent` пункта меню односимвольную строку. При использовании верхнего регистра клавиша «Shift» автоматически добавляется в массив.

В Mac OS X модификатором по молчанию является клавиша «Command» (`Keyboard.COMMAND`). В Windows или Linux это клавиша Ctrl (`Keyboard.CONTROL`). Эти клавиши по умолчанию добавляются в массив автоматически. Для назначения разных клавиш-модификаторов назначьте новый массив с кодами нужных клавиш в свойство `keyEquivalentModifiers`. Массив по умолчанию будет заменен на новый. Независимо от того, используете вы модификаторы по умолчанию или назначаете собственные, если строка, назначаемая свойству `keyEquivalent`, записана в верхнем регистре, клавиша «Shift» добавляется в массив автоматически. Константы кодов клавиш-модификаторов задаются классом Keyboard.

Строка назначенного клавишного эквивалента автоматически выводится рядом с названием пункта меню. Формат зависит от операционной системы и ее настроек.

***Примечание.*** *Если значение* `Keyboard.COMMAND` *назначено массиву модификаторов в ОС Windows, в меню не отображаются клавишные эквиваленты. Тем не менее для активации команд меню должна использоваться клавиша «Ctrl».*

В примере ниже показано, как назначить пункту меню сочетание клавиш `Ctrl+Shift+G`:

```
var item:NativeMenuItem = new NativeMenuItem("Ungroup");
item.keyEquivalent = "G";
```

В этом примере сочетание клавиш `Ctrl+Shift+G` назначается эквивалентом путем прямого задания массива модификаторов:

```
var item:NativeMenuItem = new NativeMenuItem("Ungroup");
item.keyEquivalent = "G";
item.keyEquivalentModifiers = [Keyboard.CONTROL];
```

***Примечание.*** *Клавишные эквиваленты поддерживаются только в меню программы или окна. Если добавить клавишный эквивалент в контекстное или всплывающее меню, комбинация клавиш будет отображаться рядом с пунктом меню, но нужную команду с ее помощью вызвать не удастся.*

## Мнемонические клавиши (AIR)

**Adobe AIR 1.0 и более поздних версий**

Мнемонические клавиши являются частью клавиатурного интерфейса операционных систем. Linux, Mac OS X и Windows позволяют открывать меню и выбирать команды с помощью клавиатуры, но существуют небольшие различия.

В Mac OS X пользователь вводит первую букву или две в названии команды и нажимает клавишу Return. Свойство `mnemonicIndex` игнорируется.

В Windows имеет значение только одна буква. По умолчанию значимой буквой является только первая буква в названии пункта меню, но если назначить элементу мнемоническую клавишу, то любая буква может стать значимым символом. Если два пункта меню имеют одинаковый значимый символ (неважно, назначена ли мнемоническая клавиша), то работа пользователя с меню несколько изменится. Вместо выбора меню или команды нажатием одной клавиши придется нажать эту клавишу столько раз, сколько потребуется для выделения нужного пункта, а затем нажать «Ввод» для подтверждения выбора. Чтобы избежать непредсказуемого поведения, следует назначать каждому пункту меню уникальную мнемоническую клавишу.

В Linux мнемонические клавиши по умолчанию не заданы. Необходимо указать значение для свойства `mnemonicIndex` элемента меню, чтобы активировать мнемоническую клавишу.

Задайте мнемонический символ путем назначения индекса строки названия пункта меню. Индекс первого символа — 0. Таким образом, чтобы использовать в качестве мнемонического символа для пункта меню букву «r», необходимо задать свойству `mnemonicIndex` значение 2.

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.mnemonicIndex = 2;
```

## Состояние пункта меню

**Adobe AIR 1.0 и более поздних версий**

Пункты меню имеют два свойства состояния: `checked` и `enabled`:

**checked** Установите значение `true` для отображения «галочки» рядом с пунктом меню.

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.checked = true;
```

**enabled** Переключайте значение с `true` на `false`, чтобы включать и выключать команду. Отключенные элементы отображаются затененными и не могут отправлять события `select`.

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.enabled = false;
```

## Прикрепление объекта к пункту меню

**Adobe AIR 1.0 и более поздних версий**

Свойство `data` класса NativeMenuItem позволяет ссылаться на любой объект в каждом пункте. Например, в меню «Открыть последний файл» можно назначить каждому из пунктов меню объект File соответствующего документа.

```
var file:File = File.applicationStorageDirectory.resolvePath("GreatGatsby.pdf")
var menuItem:NativeMenuItem = docMenu.addItem(new NativeMenuItem(file.name));
menuItem.data = file;
```

# Создание исходных меню (AIR)

**Adobe AIR 1.0 и более поздних версий**

В этом разделе рассказывается, как создавать исходные меню различных типов, поддерживаемые средой AIR.

## Создание объекта корневого меню

**Adobe AIR 1.0 и более поздних версий**

Чтобы создать объект NativeMenu, который будет служить корневым меню, воспользуйтесь конструктором NativeMenu:

```
var root:NativeMenu = new NativeMenu();
```

В случае меню программ и окон корневое меню — это строка меню, в которой могут быть только пункты, содержащие подменю. Контекстные и всплывающие меню не имеют строки меню, поэтому в их случае корневое меню может содержать не только подменю, но также команды и линии-разделители.

После создания меню можно добавлять в него пункты. Пункты отображаются в меню в том порядке, в котором они были добавлены, кроме случаев, когда пункты выводятся по индексам с помощью метода `addItemAt()` объекта меню.

Меню может относиться к программе, окну, значку, быть контекстным или всплывающим. Все эти типы описываются ниже:

**Настройка меню программы или меню окна**

Важно, чтобы создаваемый код был приспособлен как к меню приложения (поддерживается в Mac OS), так и к меню окна (поддерживается в других операционных системах)

```
var root:NativeMenu = new NativeMenu();
if (NativeApplication.supportsMenu)
{
    NativeApplication.nativeApplication.menu = root;
}
else if (NativeWindow.supportsMenu)
{
    nativeWindow.menu = root;
}
```

***Примечание.*** *В Mac OS для каждого приложения определяется меню со стандартными элементами. Если назначить объект NativeMenu свойству* `menu` *объекта NativeApplication, стандартное меню будет заменено. Можно не заменять меню, а использовать стандартное.*

В Adobe Flex представлен класс FlexNativeMenu, позволяющий легко создавать меню, которые работают на разных платформах. Если вы применяете Flex Framework, используйте классы FlexNativeMenu вместо класса NativeMenu.

**Настройка контекстного меню интерактивного объекта**

```
interactiveObject.contextMenu = root;
```

**Настройка меню значка Dock или меню значка на панели задач**

Важно, чтобы создаваемый код был приспособлен как к меню приложения (поддерживается в Mac OS), так и к меню окна (поддерживается в других операционных системах)

```
if (NativeApplication.supportsSystemTrayIcon)
{
    SystemTrayIcon(NativeApplication.nativeApplication.icon).menu = root;
}
else if (NativeApplication.supportsDockIcon)
{
    DockIcon(NativeApplication.nativeApplication.icon).menu = root;
}
```

***Примечание.*** *В Mac OS X определено стандартное меню для значка Dock. Если назначить новый объект NativeMenu свойству menu объекта DockIcon, его пункты будут отображаться над стандартными. Удалить или изменить пункты стандартного меню нельзя.*

**Отображение всплывающего меню**

```
root.display(stage, x, y);
```

### Дополнительные разделы справки

Разработка приложений AIR для разных платформ

## Создание подменю

**Adobe AIR 1.0 и более поздних версий**

Для создания подменю добавьте объект NativeMenuItem в родительское меню, затем назначьте объект NativeMenu, определяющий подменю свойства `submenu` этого пункта. В AIR пункты подменю и связанные с ними объекты меню можно создавать двумя способами.

Пункт меню и связанный с ним объект меню можно создать одновременно с помощью метода `addSubmenu()`:

```
var editMenuItem:NativeMenuItem = root.addSubmenu(new NativeMenu(), "Edit");
```

Создать элемент меню и назначить объект меню его свойству `submenu` по отдельности:

```
var editMenuItem:NativeMenuItem = root.addItem("Edit", false);
editMenuItem.submenu = new NativeMenu();
```

## Создание команды меню

**Adobe AIR 1.0 и более поздних версий**

Для создания команды меню добавьте в меню объект NativeMenuItem и добавьте прослушиватель событий, ссылающийся на функцию, которая реализует команду меню:

```
var copy:NativeMenuItem = new NativeMenuItem("Copy", false);
copy.addEventListener(Event.SELECT, onCopyCommand);
editMenu.addItem(copy);
```

Событие `select` может прослушиваться на самом пункте-команде (как показано в примере). Также `select` может прослушиваться в родительском объекте меню.

***Примечание.*** *Пункты меню, представляющие подменю и линии-разделители, не отправляют события `select`, поэтому не могут быть использованы в качестве команд.*

## Создание линий-разделителей в меню

**Adobe AIR 1.0 и более поздних версий**

Для создания линии-разделителя создайте объект NativeMenuItem и задайте параметру `isSeparator` значение `true` в конструкторе. Затем добавьте элемент разделителя в нужное место в меню:

```
var separatorA:NativeMenuItem = new NativeMenuItem("A", true);
editMenu.addItem(separatorA);
```

Если разделителю присвоена какая-либо метка, она не отображается.

# О контекстных меню в HTML (AIR)

**Adobe AIR 1.0 и более поздних версий**

В HTML-содержимом, отображаемом при помощи объекта HTMLLoader, для отображения контекстного меню может использоваться событие `contextmenu`. По умолчанию контекстное меню отображается автоматически, когда пользователь вызывает событие контекстного меню на выделенном тексте (щелчком правой кнопкой мыши). Чтобы меню по умолчанию не открывалось, следует прослушивать событие `contextmenu` и вызывать метод `preventDefault()` объекта события:

```
function showContextMenu(event){
    event.preventDefault();
}
```

Затем можно выводить заказное контекстное меню с помощью DHTML-технологий или путем отображения исходного меню AIR. В примере ниже показано, как выводить исходное контекстное меню вызовом метода `display()` в ответ на событие HTML `contextmenu`:

```
<html>
<head>
<script src="AIRAliases.js" language="JavaScript" type="text/javascript"></script>
<script language="javascript" type="text/javascript">

function showContextMenu(event){
    event.preventDefault();
    contextMenu.display(window.nativeWindow.stage, event.clientX, event.clientY);
}

function createContextMenu(){
    var menu = new air.NativeMenu();
    var command = menu.addItem(new air.NativeMenuItem("Custom command"));
    command.addEventListener(air.Event.SELECT, onCommand);
    return menu;
}

function onCommand(){
    air.trace("Context command invoked.");
}

var contextMenu = createContextMenu();
</script>
</head>
<body>
<p oncontextmenu="showContextMenu(event)" style="-khtml-user-select:auto;">Custom context
menu.</p>
</body>
</html>
```

# Отображение всплывающих исходных меню (AIR)

**Adobe AIR 1.0 и более поздних версий**

Любой объект NativeMenu может отображаться в произвольное время и в произвольном месте над окном. Для этого нужно вызвать метод `display()`. Этот метод должен ссылаться на рабочую область, поэтому только содержимое в изолированной программной среде приложения может выводиться в качестве всплывающего меню.

Показанный ниже метод выводит меню, определенное объектом NativeMenu под названием `popupMenu`, в ответ на щелчок мыши:

```
private function onMouseClick(event:MouseEvent):void {
    popupMenu.display(event.target.stage, event.stageX, event.stageY);
}
```

***Примечание.*** *Меню не обязательно должно отображаться как непосредственный ответ на событие. Любой метод может вызывать функцию `display()`.*

# Обработка событий меню

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Меню отправляет события, когда пользователь выделяет меню или его пункт.

## Обзор событий для классов меню

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Обрабатывать события меню можно с помощью добавления к меню или отдельным его пунктам прослушивателей событий.

| Объект | Отправляемые события |
|---|---|
| NativeMenu (AIR) | Event.PREPARING (Adobe AIR 2.6 и более поздние версии)<br>Event.DISPLAYING<br>Event.SELECT (передается от дочерних пунктов и подменю) |
| NativeMenuItem (AIR) | Event.PREPARING (Adobe AIR 2.6 и более поздние версии)<br>Event.SELECT<br>Event.DISPLAYING (передается от родительского меню) |
| ContextMenu | ContextMenuEvent.MENU_SELECT |
| ContextMenuItem | ContextMenuEvent.MENU_ITEM_SELECT<br>Event.SELECT (AIR) |

## Выбор событий меню

**Adobe AIR 1.0 и более поздних версий**

Для обработки щелчка мыши на пункте меню добавьте к объекту NativeMenuItem прослушиватель для события `select`:

```
var menuCommandX:NativeMenuItem = new NativeMenuItem("Command X");
menuCommandX.addEventListener(Event.SELECT, doCommandX)
```

Так как события `select` передаются вверх по иерархии, можно также прослушивать события select в родительском меню. При прослушивании на уровне меню для определения того, какая команда была выбрана, можно использовать свойство `target` объекта события. В примере ниже показано, как отслеживать выбранную команду с помощью метки:

```
 var colorMenuItem:NativeMenuItem = new NativeMenuItem("Choose a color");
var colorMenu:NativeMenu = new NativeMenu();
colorMenuItem.submenu = colorMenu;

var red:NativeMenuItem = new NativeMenuItem("Red");
var green:NativeMenuItem = new NativeMenuItem("Green");
var blue:NativeMenuItem = new NativeMenuItem("Blue");
colorMenu.addItem(red);
colorMenu.addItem(green);
colorMenu.addItem(blue);

if(NativeApplication.supportsMenu){
    NativeApplication.nativeApplication.menu.addItem(colorMenuItem);
    NativeApplication.nativeApplication.menu.addEventListener(Event.SELECT, colorChoice);
} else if (NativeWindow.supportsMenu){
    var windowMenu:NativeMenu = new NativeMenu();
    this.stage.nativeWindow.menu = windowMenu;
    windowMenu.addItem(colorMenuItem);
    windowMenu.addEventListener(Event.SELECT, colorChoice);
}

function colorChoice(event:Event):void {
    var menuItem:NativeMenuItem = event.target as NativeMenuItem;
    trace(menuItem.label + " has been selected");
}
```

Если используется класс ContextMenuItem, можно прослушивать либо событие `select`, либо событие `menuItemSelect`. Событие `menuItemSelect` дает дополнительную информацию об объекте, которому принадлежит контекстное меню, но не передается вверх по иерархии.

## Отображение событий меню

**Adobe AIR 1.0 и более поздних версий**

Для обработки открытия меню можно добавить прослушиватель события `displaying`, отправляемого перед отображением меню. Событие displaying может использоваться для обновления меню, например при добавлении или удалении элементов или при обновлении установленных флажков для отдельных пунктов. Вы также можете прослушивать событие `menuSelect` объекта ContextMenu.

В AIR 2.6 и более поздних версиях можно использовать событие `preparing` для обновления меню в ответ на отображение меню или выбор пункта меню с помощью клавиатурного сокращения.

# Пример собственного меню: меню окна и приложения (AIR)

**Adobe AIR 1.0 и более поздних версий**

В примере ниже показано, как создавать меню, показанное в разделе ««Структура исходного меню (AIR)» на странице 682».

Меню приспособлено для работы как в ОС Windows, где поддерживаются только меню окна, так и в Mac OS X, где поддерживаются только меню приложения. Для определения системы конструктор класса MenuExample проверяет статические свойства `supportsMenu` классов NativeWindow и NativeApplication. Если `NativeWindow.supportsMenu` равно `true`, конструктор создает объект NativeMenu для окна, после чего создает и добавляет подменю «Файл» и «Правка». Если `NativeApplication.supportsMenu` равно `true`, конструктор создает и добавляет меню «Файл» и «Правка» к существующему меню операционной системы Mac OS X.

В примере также показана обработка событий. Событие `select` обрабатывается на уровне пункта меню или на уровне меню. Каждое меню в цепочке, начиная с меню, содержащего выделенный пункт, до корневого, отвечает на событие `select`. Событие `displaying` используется с меню «Открыть последний файл». Непосредственно перед открытием меню список элементов обновляется на основе последнего массива Documents (в этом примере он не меняется). Хотя в примере это не показано, прослушивать события `displaying` можно на отдельных пунктах.

```
package {
    import flash.display.NativeMenu;
    import flash.display.NativeMenuItem;
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.filesystem.File;
    import flash.desktop.NativeApplication;

    public class MenuExample extends Sprite
    {
        private var recentDocuments:Array =
            new Array(new File("app-storage:/GreatGatsby.pdf"),
                      new File("app-storage:/WarAndPeace.pdf"),
                      new File("app-storage:/Iliad.pdf"));

        public function MenuExample()
        {
            var fileMenu:NativeMenuItem;
            var editMenu:NativeMenuItem;

            if (NativeWindow.supportsMenu){
                stage.nativeWindow.menu = new NativeMenu();
                stage.nativeWindow.menu.addEventListener(Event.SELECT, selectCommandMenu);
                fileMenu = stage.nativeWindow.menu.addItem(new NativeMenuItem("File"));
                fileMenu.submenu = createFileMenu();
                editMenu = stage.nativeWindow.menu.addItem(new NativeMenuItem("Edit"));
                editMenu.submenu = createEditMenu();
            }

            if (NativeApplication.supportsMenu){
                NativeApplication.nativeApplication.menu.addEventListener(Event.SELECT,
selectCommandMenu);
                fileMenu = NativeApplication.nativeApplication.menu.addItem(new
NativeMenuItem("File"));
                fileMenu.submenu = createFileMenu();
                editMenu = NativeApplication.nativeApplication.menu.addItem(new
NativeMenuItem("Edit"));
                editMenu.submenu = createEditMenu();
            }
        }
```

```
public function createFileMenu():NativeMenu {
    var fileMenu:NativeMenu = new NativeMenu();
    fileMenu.addEventListener(Event.SELECT, selectCommandMenu);

    var newCommand:NativeMenuItem = fileMenu.addItem(new NativeMenuItem("New"));
    newCommand.addEventListener(Event.SELECT, selectCommand);
    var saveCommand:NativeMenuItem = fileMenu.addItem(new NativeMenuItem("Save"));
    saveCommand.addEventListener(Event.SELECT, selectCommand);
    var openRecentMenu:NativeMenuItem =
            fileMenu.addItem(new NativeMenuItem("Open Recent"));
    openRecentMenu.submenu = new NativeMenu();
    openRecentMenu.submenu.addEventListener(Event.DISPLAYING,
                                    updateRecentDocumentMenu);
    openRecentMenu.submenu.addEventListener(Event.SELECT, selectCommandMenu);

    return fileMenu;
}

public function createEditMenu():NativeMenu {
    var editMenu:NativeMenu = new NativeMenu();
    editMenu.addEventListener(Event.SELECT, selectCommandMenu);

    var copyCommand:NativeMenuItem = editMenu.addItem(new NativeMenuItem("Copy"));
    copyCommand.addEventListener(Event.SELECT, selectCommand);
    copyCommand.keyEquivalent = "c";
    var pasteCommand:NativeMenuItem =
            editMenu.addItem(new NativeMenuItem("Paste"));
    pasteCommand.addEventListener(Event.SELECT, selectCommand);
    pasteCommand.keyEquivalent = "v";
    editMenu.addItem(new NativeMenuItem("", true));
    var preferencesCommand:NativeMenuItem =
            editMenu.addItem(new NativeMenuItem("Preferences"));
    preferencesCommand.addEventListener(Event.SELECT, selectCommand);

    return editMenu;
}

private function updateRecentDocumentMenu(event:Event):void {
    trace("Updating recent document menu.");
    var docMenu:NativeMenu = NativeMenu(event.target);

    for each (var item:NativeMenuItem in docMenu.items) {
        docMenu.removeItem(item);
    }

    for each (var file:File in recentDocuments) {
        var menuItem:NativeMenuItem =
                docMenu.addItem(new NativeMenuItem(file.name));
        menuItem.data = file;
        menuItem.addEventListener(Event.SELECT, selectRecentDocument);
    }
}

private function selectRecentDocument(event:Event):void {
    trace("Selected recent document: " + event.target.data.name);
}
```

```
        private function selectCommand(event:Event):void {
            trace("Selected command: " + event.target.label);
        }

        private function selectCommandMenu(event:Event):void {
            if (event.currentTarget.parent != null) {
                var menuItem:NativeMenuItem =
                        findItemForMenu(NativeMenu(event.currentTarget));
                if (menuItem != null) {
                    trace("Select event for \"" +
                            event.target.label +
                            "\" command handled by menu: " +
                            menuItem.label);
                }
            } else {
                trace("Select event for \"" +
                        event.target.label +
                        "\" command handled by root menu.");
            }
        }

        private function findItemForMenu(menu:NativeMenu):NativeMenuItem {
            for each (var item:NativeMenuItem in menu.parent.items) {
                if (item != null) {
                    if (item.submenu == menu) {
                        return item;
                    }
                }
            }
            return null;
        }
    }
}
```

# Глава 37. Значки панели задач AIR

**Adobe AIR 1.0 и более поздних версий**

Многие операционные системы имеют панель задач, например док Mac OS X, на которой может отображаться значок приложения. Adobe® AIR® предоставляет интерфейс для взаимодействия со значком приложения на панели задач через свойство `NativeApplication.nativeApplication.icon`.

- Использование значков области уведомлений и доков (Flex)
- Использование значков области уведомлений и доков (Flash)

### Дополнительные разделы справки

flash.desktop.NativeApplication

flash.desktop.DockIcon

flash.desktop.SystemTrayIcon

## О значках панели задач

**Adobe AIR 1.0 и более поздних версий**

AIR создает объект `NativeApplication.nativeApplication.icon` автоматически. Объект имеет тип DockIcon или SystemTrayIcon в зависимости от операционной системы. Определить, какой из этих подклассов InteractiveIcon поддерживается AIR в данной операционной системе, можно при помощи свойств `NativeApplication.supportsDockIcon` и `NativeApplication.supportsSystemTrayIcon`. Базовый класс InteractiveIcon содержит свойства `width`, `height` и `bitmaps`, при помощи которых можно изменить изображение значка. Однако при попытке получить доступ к свойствам DockIcon или SystemTrayIcon не в той операционной системе возникает ошибка времени выполнения.

Для установки или изменения изображения значка создайте массив, содержащий одно или несколько изображений, и присвойте его свойству `NativeApplication.nativeApplication.icon.bitmaps`. Размер значков панели задач может отличаться в зависимости от операционной системы. Во избежание снижения качества изображения из-за масштабирования можно добавить в массив `bitmaps` несколько размеров изображения. При наличии нескольких изображений AIR выбирает размер, ближайший к текущему размеру отображения значка панели задач, применяя масштабирование лишь при необходимости. Следующий пример кода задает изображение для значка панели задач, выбирая из двух изображений:

```
NativeApplication.nativeApplication.icon.bitmaps =
            [bmp16x16.bitmapData, bmp128x128.bitmapData];
```

Для изменения изображения значка присвойте массив, содержащий новое изображение или изображения, свойству `bitmaps`. Можно анимировать значок, изменив изображение в ответ на событие `enterFrame` или `timer`.

Чтобы удалить значок из области уведомления в Windows и Linux или восстановить вид значка по умолчанию в Mac OS X, задайте для `bitmaps` пустой массив.

```
NativeApplication.nativeApplication.icon.bitmaps = [];
```

# Значки дока

**Adobe AIR 1.0 и более поздних версий**

AIR поддерживает значки дока, если `NativeApplication.supportsDockIcon` имеет значение `true`. Свойство `NativeApplication.nativeApplication.icon` представляет значок приложения в доке (а не значок дока окна).

***Примечание.*** *AIR не поддерживает изменение значков окна в доке в операционной системе Mac OS X. Кроме того, изменения значка приложения в доке применяются только на время выполнения приложения — при завершении работы приложения значок принимает свой обычный вид.*

## Меню значков дока

**Adobe AIR 1.0 и более поздних версий**

В стандартное меню дока можно добавить команды, создав объект NativeMenu, содержащий эти команды, и присвоив его свойству `NativeApplication.nativeApplication.icon.menu`. Добавленные пункты меню отображаются поверх стандартных пунктов меню значков дока.

## Оповещения дока

**Adobe AIR 1.0 и более поздних версий**

Подпрыгивание значка дока достигается при помощи метода `NativeApplication.nativeApplication.icon.bounce()`. Если для параметра `bounce() priority` задано значение informational, значок прыгает один раз. Если для него задано значение critical, значок прыгает до тех пор, пока пользователь не активирует приложение. Константы для параметра `priority` определены в классе NotificationType.

***Примечание.*** *Значок не прыгает, если приложение уже активно.*

## События значков дока

**Adobe AIR 1.0 и более поздних версий**

При нажатии на значок дока объект NativeApplication отправляет событие `invoke`. Если приложение еще не запущено, система запускает его. В противном случае событие `invoke` отправляется экземпляру выполняющегося приложения.

# Значки области уведомлений

**Adobe AIR 1.0 и более поздних версий**

AIR поддерживает значки области уведомлений, если для свойства `NativeApplication.supportsSystemTrayIcon` задано значение `true`, что справедливо в настоящее время только для Windows и большинства дистрибутивов Linux. В Windows и Linux значки области уведомлений отображаются в области уведомлений панели задач. По умолчанию не отображается ни один значок. Для отображения значка присвойте массив, содержащий объекты BitmapData, свойству `bitmaps` значка. Для изменения изображения значка присвойте массив, содержащий новые изображения, свойству `bitmaps`. Для удаления значка задайте для `bitmaps` значение `null`.

## Меню значков области уведомлений

**Adobe AIR 1.0 и более поздних версий**

Можно добавить меню к значку области уведомлений, создав объект NativeMenu и присвоив его свойству `NativeApplication.nativeApplication.icon.menu` (операционная система не предоставляет меню по умолчанию). Доступ к меню значка области уведомлений осуществляется щелчком правой кнопки мыши.

## Всплывающие подсказки значков области уведомлений

**Adobe AIR 1.0 и более поздних версий**

Можно добавить к значку всплывающую подсказку, задав свойство tooltip:

```
NativeApplication.nativeApplication.icon.tooltip = "Application name";
```

## События значков области уведомлений

**Adobe AIR 1.0 и более поздних версий**

Объект SystemTrayIcon, на который указывает свойство NativeApplication.nativeApplication.icon, отправляет ScreenMouseEvent для событий `click`, `mouseDown`, `mouseUp`, `rightClick`, `rightMouseDown` и `rightMouseUp`. Эти события, а также меню значков, позволяют пользователям взаимодействовать с приложением при отсутствии его видимых окон.

## Пример: создание приложения без окон

**Adobe AIR 1.0 и более поздних версий**

Следующий пример создает приложение AIR, которое имеет значки в области уведомлений, но не имеет видимых окон. (Для свойства `visible` приложения не должно быть установлено значение `true` в дескрипторе приложения; в противном случае окно будет видимо при запуске приложения.)

```
package
{
    import flash.display.Loader;
    import flash.display.NativeMenu;
    import flash.display.NativeMenuItem;
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.desktop.DockIcon;
    import flash.desktop.SystemTrayIcon;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.desktop.NativeApplication;

    public class SysTrayApp extends Sprite
    {
        public function SysTrayApp():void{
            NativeApplication.nativeApplication.autoExit = false;
            var icon:Loader = new Loader();
            var iconMenu:NativeMenu = new NativeMenu();
            var exitCommand:NativeMenuItem = iconMenu.addItem(new NativeMenuItem("Exit"));
                exitCommand.addEventListener(Event.SELECT, function(event:Event):void {
                    NativeApplication.nativeApplication.icon.bitmaps = [];
                    NativeApplication.nativeApplication.exit();
                });

            if (NativeApplication.supportsSystemTrayIcon) {
                NativeApplication.nativeApplication.autoExit = false;
                icon.contentLoaderInfo.addEventListener(Event.COMPLETE, iconLoadComplete);
                icon.load(new URLRequest("icons/AIRApp_16.png"));

                var systray:SystemTrayIcon =
                    NativeApplication.nativeApplication.icon as SystemTrayIcon;
                systray.tooltip = "AIR application";
                systray.menu = iconMenu;
            }

            if (NativeApplication.supportsDockIcon){
                icon.contentLoaderInfo.addEventListener(Event.COMPLETE,iconLoadComplete);
                icon.load(new URLRequest("icons/AIRApp_128.png"));
                var dock:DockIcon = NativeApplication.nativeApplication.icon as DockIcon;
                dock.menu = iconMenu;
            }
        }

        private function iconLoadComplete(event:Event):void
        {
            NativeApplication.nativeApplication.icon.bitmaps =
                [event.target.content.bitmapData];
        }
    }
}
```

***Примечание.*** *При использовании компонента Flex WindowedApplication необходимо задать для атрибута* `visible` *тега WindowedApplication значение* `false`*. Этот атрибут заменяет настройку в дескрипторе приложения.*

***Примечание.*** *Этот пример предполагает использование файлов изображений с именами* `AIRApp_16.png` *и* `AIRApp_128.png` *в подкаталоге* `icons` *приложения. (Файлы с образцами значков, которые можно копировать в папку проекта, включены в пакет AIR SDK.)*

# Кнопки и значки окон на панели задач

**Adobe AIR 1.0 и более поздних версий**

Значки окон, как правило, отображаются на панели задач в области отображения ярлыков открытых окон, или доке, чтобы пользователи могли легко получить доступ к окнам фонового режима или свернутым окнам. Док операционной системы Mac OS X отображает значок приложения, а также значки свернутых окон. На панелях задач в Microsoft Windows и Linux отображаются кнопка со значком программы и заголовки для каждого окна обычного типа в приложении.

## Подсветка кнопок окон на панели задач

**Adobe AIR 1.0 и более поздних версий**

Если окно открыто в фоновом режиме, можно уведомить пользователя о том, что произошло ожидаемое событие, связанное с окном. В Mac OS X оповещение пользователя может осуществляться за счет прыгающего значка приложения в доке (как описано в разделе «Оповещения дока» на странице 696). В Windows и Linux можно подсветить кнопку окна на панели задач, вызвав метод `notifyUser()` экземпляра NativeWindow. Параметр `type`, передаваемый этому методу, определяет срочность оповещения:

- `NotificationType.CRITICAL`: значок окна мигает до тех пор, пока пользователь не активирует окно.
- `NotificationType.INFORMATIONAL`: значок окна подсвечивается, изменяя цвет.

  ***Примечание.*** *В Linux поддерживается только информационный тип уведомлений. Передача значения любого типа в функцию* `notifyUser()` *создаст точно такой же эффект.*

  Следующее выражение подсвечивает кнопку окна на панели задач:

```
stage.nativeWindow.notifyUser(NotificationType.CRITICAL);
```

  Вызов метода `NativeWindow.notifyUser()` в операционной системе, не поддерживающей оповещение на уровне окон, не действует. При помощи свойства `NativeWindow.supportsNotification` можно определить, поддерживается ли оповещение на уровне окон.

## Создание окон без кнопок или значков на панели задач

**Adobe AIR 1.0 и более поздних версий**

В операционной системе Windows окна типов *utility* или *lightweight* не отображаются на панели задач. Невидимые окна также не отображаются на панели задач.

Поскольку начальное окно всегда относится к типу *normal*, для создания приложения, окна которого не должны отображаться на панели задач, необходимо либо закрыть исходное окно, либо сделать его невидимым. Чтобы закрыть все окна приложения, не завершая работу самого приложения, задайте для свойства `autoExit` объекта NativeApplication значение `false` перед закрытием последнего окна. Чтобы сделать начальное окно постоянно невидимым, вставьте `<visible>false</visible>` в элемент `<initalWindow>` файла дескриптора приложения (не задавайте для свойства `visible` значение `true` и не вызывайте метод `activate()` окна).

Для новых окон, открытых в приложении, установите свойство `type` объекта NativeWindowInitOption, передаваемого конструктору окна, на значение `NativeWindowType.UTILITY` или `NativeWindowType.LIGHTWEIGHT`.

В Mac OS X свернутые окна отображаются на панели задач дока. Чтобы значок свернутого окна не отображался, рекомендуется скрыть окно, а не сворачивать его. Следующий пример прослушивает событие изменения `nativeWindowDisplayState` и отменяет его при сворачивании окна. Вместо этого обработчик задает для свойства `visible` значение `false`:

```
 private function preventMinimize(event:NativeWindowDisplayStateEvent):void{
    if(event.afterDisplayState == NativeWindowDisplayState.MINIMIZED){
        event.preventDefault();
        event.target.visible = false;
    }
}
```

При сворачивании окна в док Mac OS X при установке свойства `visible` на значение `false` значок дока не удаляется. Пользователь может щелкнуть значок, чтобы снова отобразить окно.

# Глава 38. Работа с файловой системой

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Flash® Player обеспечивает основные возможности чтения и записи файлов, используя класс FileReference. В целях безопасности при работе проигрывателя Flash Player пользователь должен всегда давать разрешение на чтение или запись файла.

Adobe® AIR® обеспечивает более полный доступ к файловой системе главного компьютера, чем это позволяет Flash Player. С помощью API файловой системы AIR можно создавать и открывать каталоги и файлы, управлять ими, записывать данные и т. д.

### Дополнительные разделы справки

flash.net.FileReference

flash.net.FileReferenceList

flash.filesystem.File

flash.filesystem.FileStream

## Использование класса FileReference

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Объект FileReference представляет файл данных на клиентском компьютере или на сервере. Методы класса FileReference позволяют приложению загружать и сохранять файлы данных на жестком диске, а также передавать данные файла на удаленные серверы и обратно.

Класс FileReference имеет два разных подхода к загрузке, передаче и сохранению файлов данных. С момента введения класс FileReference включает методы `browse()`, `upload()` и `download()`. Используйте метод `browse()`, чтобы предоставить пользователю возможность выбрать файл. Используйте метод `upload()` для переноса данных файла на удаленный сервер. Используйте метод `download()` для получения данных сервера и их сохранения в локальном файле. Начиная с Flash Player 10 и Adobe AIR 1.5 класс FileReference также включает методы `load()` и `save()`. Методы `load()` и `save()` обеспечивают возможность непосредственного вызова и сохранения локальных файлов. Использование этих методов идентично методам в классах URLLoader и Loader с соответствующими именами.

***Примечание.*** *Класс File, который расширяет класс FileReference, и класс FileStream обеспечивают дополнительные функции работы с файлами и локальной файловой системой. Классы File и FileStream поддерживаются только AIR, но не поддерживаются Flash Player.*

## Класс FileReference

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Каждый объект FileReference представляет один файл данных на локальном компьютере. Свойства класса FileReference содержат сведения о размере файла, его типе, имени, расширении, авторе, дате создания и дате изменения.

***Примечание.*** *Свойство `creator` поддерживается только в Mac OS. На всех остальных платформах оно имеет значение `null`.*

***Примечание.*** *Свойство `extension` поддерживается только в Adobe AIR.*

Экземпляр класса FileReference можно создать двумя способами.

- Используйте оператор `new`, как показано в следующем коде:

```
import flash.net.FileReference;
var fileRef:FileReference = new FileReference();
```

- Вызовите метод `FileReferenceList.browse()`, открывающий диалоговое окно, с помощью которого пользователь может выбрать один или несколько файлов для отправки. Затем метод создает массив объектов FileReference, если пользователь успешно выбирает один или несколько файлов.

Создав объект FileReference, можно выполнить следующие операции.

- Вызовите метод `FileReference.browse()`, который открывает диалоговое окно и подсказывает пользователю выбрать один файл из локальной файловой системы. Обычно это делается перед последующим вызовом метода `FileReference.upload()` или `FileReference.load()`. Вызовите метод `FileReference.upload()` для отправки файла на удаленный сервер. Вызовите метод `FileReference.load()` для открытия локального файла.
- Вызовите метод `FileReference.download()`. Метод `download()` открывает диалоговое окно, в котором пользователь может выбрать местоположение для сохранения нового файла. После этого выполняется загрузка данных с сервера и их сохранение в новом файле.
- Вызовите метод `FileReference.load()`. Данный метод начинает загрузку данных из файла, который до этого был задан методом `browse()`. Метод `load()` не может быть вызван до завершения работы метода `browse()` (до выбора файла пользователем).
- Вызовите метод `FileReference.save()`. Этот метод открывает диалоговое окно и приглашает пользователя выбрать одно местоположение для файла в локальной файловой системе. Затем он производит запись данных в выбранное местоположение.

***Примечание.*** *Одновременно может выполняться только один вызов `browse()`, `download()` или `save()`, так как нельзя открыть сразу несколько диалоговых окон.*

Свойства объекта FileReference, такие как `name`, `size` или `modificationDate` определяются только после того, как произойдет одно из следующих событий.

- Вызван метод `FileReference.browse()` или `FileReferenceList.browse()`, и пользователь выбрал файл в диалоговом окне.
- Вызван метод `FileReference.download()`, и пользователь выбрал местоположение для нового файла в диалоговом окне.

***Примечание.*** *До завершения загрузки заполняется только свойство `FileReference.name`. Все остальные свойства становятся доступны после того, как файл загружен полностью.*

Во время выполнения методов `FileReference.browse()`, `FileReferenceList.browse()`, `FileReference.download()`, `FileReference.load()` или `FileReference.save()` большинство проигрывателей продолжают воспроизводить содержимое SWF-файла, а также отправлять события и выполнять код.

Для выполнения операций отправки и загрузки SWF-файл может вызывать файлы только из собственного домена, а также из доменов, указанных в файле политики. Необходимо поместить файл политики на соответствующий сервер, если данный сервер находится за пределами домена SWF-файла, инициирующего отправку или загрузку.

См. описание класса FileReference.

## Загрузка данных из файлов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `FileReference.load()` позволяет загружать в память данные из локального файла.

***Примечание.*** *Сначала код должен вызвать метод `FileReference.browse()`, чтобы пользователь мог выбрать файл для загрузки. Это ограничение не относится к содержимому, работающему в Adobe AIR в изолированной программной среде системы защиты приложения.*

Метод `FileReference.load()` возвращает данные сразу после вызова, однако загружаемые данные становятся доступными немного позже. Объект FileReference отправляет события для вызова методов прослушивателей на каждом этапе выполнения загрузки.

Во время загрузки объект FileReference отправляет следующие события.

- Событие `open` (`Event.OPEN`): отправляется в момент начала операции загрузки.
- Событие `progress` (`ProgressEvent.PROGRESS`): периодически отправляется по мере считывания байтов данных из файла.
- Событие `complete` (`Event.COMPLETE`): отправляется после успешного завершения операции загрузки файла.
- Событие `ioError` (`IOErrorEvent.IO_ERROR`): отправляется, если загрузку не удается выполнить из-за ошибки ввода/вывода при открытии файла или чтении данных.

После отправки события об успешном завершении операции объектом FileReference, загруженные данные становятся доступными в качестве ByteArray в свойстве `data` объекта FileReference.

В следующем примере показано, как попросить пользователя выбрать файл и после этого загрузить данные указанного файла в память.

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample1 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample1()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
            fileRef.browse([textTypeFilter]);
        }
        public function onFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.addEventListener(Event.COMPLETE, onComplete);
            fileRef.load();
        }

        public function onProgress(evt:ProgressEvent):void
        {
            trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }

        public function onComplete(evt:Event):void
        {
            trace("File was successfully loaded.");
            trace(fileRef.data);
        }

        public function onCancel(evt:Event):void
        {
            trace("The browse request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

В примере кода сначала создается объект FileReference с именем `fileRef`, который затем вызывает метод `browse()`. Метод `browse()` открывает диалоговое окно, в котором пользователь может выбрать файл. После выбора файла код вызывает метод `onFileSelected()`. Этот метод добавляет прослушиватели для событий `progress` и `complete`, а затем вызывает метод `load()` объекта FileReference. Другие методы обработчиков в примере просто выводят сообщения для отслеживания прогресса операции загрузки. По завершении загрузки приложение отображает содержимое загруженного файла с использованием метода `trace()`.

В Adobe AIR класс FileStream обеспечивает дополнительные функции для чтения данных из локального файла. См. раздел ««Чтение и запись файлов» на странице 741».

## Сохранение данных в локальных файлах

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `FileReference.save()` позволяет сохранять данные в локальном файле. Сначала он открывает диалоговое окно, чтобы пользователь ввел имя для нового файла и местоположение, в котором его нужно сохранить. После выбора имени и местоположения файла эти данные записываются в новый файл. При успешном сохранении файла свойства объекта FileReference заполняются свойствами локального файла.

***Примечание.*** *Код может вызывать метод `FileReference.save()` только в ответ на событие пользователя, такое как щелчок мыши или нажатие клавиши. В противном случае выводится сообщение об ошибке. Это ограничение не относится к содержимому, работающему в Adobe AIR в изолированной программной среде системы защиты приложения.*

Метод `FileReference.save()` возвращает данные сразу после вызова. Затем объект FileReference отправляет события для вызова методов прослушивателей на каждом этапе выполнения сохранения.

Во время сохранения объект FileReference отправляет следующие события.

- Событие `select` (`Event.SELECT`): отправляется, когда пользователь указывает местоположение и имя для сохранения нового файла.
- Событие `cancel` (`Event.CANCEL`): отправляется, когда пользователь нажимает кнопку «Отмена» в диалоговом окне.
- Событие `open` (`Event.OPEN`): отправляется в момент начала операции загрузки.
- Событие `progress` (`ProgressEvent.PROGRESS`): периодически отправляется по мере считывания байтов данных из файла.
- Событие `complete` (`Event.COMPLETE`): отправляется после успешного завершения операции загрузки файла.
- Событие `ioError` (`IOErrorEvent.IO_ERROR`): отправляется, если загрузку не удается выполнить из-за ошибки ввода/вывода при открытии файла или чтении данных.

Тип объекта, передаваемый в параметре `data` метода `FileReference.save()`, определяет способ записи данных в файл:

- Если это значение String, оно сохраняется как текстовый файл в кодировке UTF-8.
- Если это объект XML, он записывается в XML-файл с полным сохранением форматирования.
- Если это объект ByteArray, его содержимое записывается непосредственно в файл без каких-либо преобразований.
- Если объект относится к другому типу, то метод `FileReference.save()` вызывает для него метод `toString()` и сохраняет полученное значение String в текстовый файл с кодировкой UTF-8. Если не удается вызвать метод `toString()` объекта, выдается ошибка.

Если параметр `data` имеет значение `null`, выдается ошибка.

Следующий код расширяет предыдущий пример кода с использованием метода `FileReference.load()`. После чтения данных из файла этот код просит пользователя ввести имя файла и сохраняет данные в новый файл.

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample2 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample2()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
            fileRef.browse([textTypeFilter]);
        }
        public function onFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.addEventListener(Event.COMPLETE, onComplete);
            fileRef.load();
        }

        public function onProgress(evt:ProgressEvent):void
        {
            trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }
        public function onCancel(evt:Event):void
        {
            trace("The browse request was canceled by the user.");
        }
        public function onComplete(evt:Event):void
        {
            trace("File was successfully loaded.");
            fileRef.removeEventListener(Event.SELECT, onFileSelected);
            fileRef.removeEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.removeEventListener(Event.COMPLETE, onComplete);
            fileRef.removeEventListener(Event.CANCEL, onCancel);
            saveFile();
        }
        public function saveFile():void
        {
            fileRef.addEventListener(Event.SELECT, onSaveFileSelected);
```

```
            fileRef.save(fileRef.data,"NewFileName.txt");
        }

        public function onSaveFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onSaveProgress);
            fileRef.addEventListener(Event.COMPLETE, onSaveComplete);
            fileRef.addEventListener(Event.CANCEL, onSaveCancel);
        }

        public function onSaveProgress(evt:ProgressEvent):void
        {
            trace("Saved " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }

        public function onSaveComplete(evt:Event):void
        {
            trace("File saved.");
            fileRef.removeEventListener(Event.SELECT, onSaveFileSelected);
            fileRef.removeEventListener(ProgressEvent.PROGRESS, onSaveProgress);
            fileRef.removeEventListener(Event.COMPLETE, onSaveComplete);
            fileRef.removeEventListener(Event.CANCEL, onSaveCancel);
        }

        public function onSaveCancel(evt:Event):void
        {
            trace("The save request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

После завершения загрузки всех данных файла, код вызывает метод `onComplete()`. Метод `onComplete()` удаляет прослушиватели событий загрузки и вызывает метод `saveFile()`. Метод `saveFile()` вызывает метод `FileReference.save()`. Метод `FileReference.save()` открывает новое диалоговое окно, где пользователь может ввести имя и местоположение для сохранения нового файла. Оставшиеся методы прослушивателей событий отслеживают прогресс сохранения файла вплоть до его завершения.

В Adobe AIR класс FileStream обеспечивает дополнительные функции для записи данных в локальный файл. См. раздел ««Чтение и запись файлов» на странице 741».

## Отправка файлов на сервер

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы отправить файлы на сервер, сначала вызовите метод `browse()`, чтобы пользователь мог выбрать один или несколько файлов. Затем при вызове метода `FileReference.upload()` выбранный файл передается на сервер. Если пользователь выбрал несколько файлов с помощью метода `FileReferenceList.browse()`, проигрыватель Flash Player создает массив с именем `FileReferenceList.fileList`, содержащий выбранные файлы. После этого можно использовать метод `FileReference.upload()`, чтобы загрузить каждый файл по отдельности.

***Примечание.*** *Метод `FileReference.browse()` позволяет загружать только отдельные файлы. Чтобы предоставить пользователю возможность загружать несколько файлов, используйте метод `FileReferenceList.browse()`.*

По умолчанию системное диалоговое окно выбора файлов позволяет выбирать любой тип файлов с локального компьютера. Разработчики могут указать один или несколько пользовательских фильтров типов файлов с помощью класса FileFilter и передать массив экземпляров фильтра файлов методу `browse()`.

```
var imageTypes:FileFilter = new FileFilter("Images (*.jpg, *.jpeg, *.gif, *.png)", "*.jpg;
*.jpeg; *.gif; *.png");
var textTypes:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt; *.rtf");
var allTypes:Array = new Array(imageTypes, textTypes);
var fileRef:FileReference = new FileReference();
fileRef.browse(allTypes);
```

Когда пользователь выбрал файлы и нажал кнопку «Открыть» в системном окне выбора файлов, отправляется событие `Event.SELECT`. Если для выбора отправляемого файла использовался метод `FileReference.browse()`, следующий код отправляет файл на веб-сервер.

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
try
{
    var success:Boolean = fileRef.browse();
}
catch (error:Error)
{
    trace("Unable to browse for files.");
}
function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest("http://www.[yourdomain].com/fileUploadScript.cfm")
    try
    {
        fileRef.upload(request);
    }
    catch (error:Error)
    {
        trace("Unable to upload file.");
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

*Чтобы отправить данные на сервер с помощью метода `FileReference.upload()`, нужно использовать свойства `URLRequest.method` и `URLRequest.data` для отправки переменных с помощью метода `POST` или `GET`.*

При попытке отправить файл с помощью метода `FileReference.upload()` отправляется любое из следующих событий.

- `open` event (`Event.OPEN`): отправляется в момент начала операции отправки.
- Событие `progress` (`ProgressEvent.PROGRESS`): периодически отправляется по мере отправки байтов данных из файла.
- Событие `complete` (`Event.COMPLETE`): отправляется после успешного завершения операции загрузки файла.
- Событие `httpStatus` (`HTTPStatusEvent.HTTP_STATUS`): отправляется при сбое отправки в результате ошибки протокола HTTP.
- Событие `httpResponseStatus` (`HTTPStatusEvent.HTTP_RESPONSE_STATUS`): отправляется, если вызов метода `upload()` или `uploadUnencoded()` пытается обратиться к данным через HTTP и Adobe AIR может обнаружить и вернуть код состояния для запроса.
- Событие `securityError` (`SecurityErrorEvent.SECURITY_ERROR`): отправляется при сбое отправки вследствие нарушения системы безопасности.
- `uploadCompleteData` event (`DataEvent.UPLOAD_COMPLETE_DATA`): отправляется после получения с сервера данных о успешном завершении отправки.
- Событие `ioError` (`IOErrorEvent.IO_ERROR`): отправляется при сбое отправки по любой из следующих причин:
  - Возникает ошибка ввода-вывода, когда проигрыватель Flash Player считывает, записывает или передает файл.
  - SWF-файл пытается загрузить файл на сервер, требующий проверки подлинности (имя пользователя и пароль). В процессе отправки Flash Player не дает пользователям возможности ввести пароли.
  - Параметр `url` содержит недействительный протокол. Метод `FileReference.upload()` должен использовать HTTP или HTTPS.

*Проигрыватель Flash Player не обеспечивает полную поддержку серверов, требующих проверки подлинности. Только SWF-файлы, выполняемые в обозревателе с использованием подключаемого модуля или элемента управления Microsoft ActiveX®, могут выводить диалоговое окно, чтобы пользователь ввел имя пользователя и пароль для проверки подлинности, и только после этого показывать диалоговое окно загрузки. При загрузке с помощью подключаемого модуля или элемента управления ActiveX, а также при загрузке или отправке данных посредством автономного или внешнего проигрывателя происходит сбой передачи файлов.*

Для создания сценария сервера с помощью ColdFusion, чтобы принять отправку файла из проигрывателя Flash Player, можно использовать код, подобный следующему:

```
<cffile action="upload" filefield="Filedata" destination="#ExpandPath('./')#"
nameconflict="OVERWRITE" />
```

Этот код ColdFusion отправляет файл, переданный проигрывателем Flash Player, и сохраняет его в том же каталоге, что и шаблон ColdFusion, перезаписывая файлы с одинаковыми именами. Предыдущий код показывает только минимальный фрагмент кода, необходимый для принятия отправки файла. Этот сценарий не следует использовать в рабочей среде. В идеале следует добавить проверку данных, чтобы пользователи могли отправлять только допустимые типы файлов, такие как изображения, а потенциально опасные сценарии, выполняемые на стороне сервера — нет.

Следующий код демонстрирует отправки файлов с помощью PHP, включая проверку данных. Сценарий ограничивает число отправляемых файлов в каталоге до 10, следит, чтобы размер файлов не превышал 200 КБ, и разрешает отправлять и сохранять в файловой системе только JPEG-, GIF- или PNG-файлы.

```
<?php
$MAXIMUM_FILESIZE = 1024 * 200; // 200KB
$MAXIMUM_FILE_COUNT = 10; // keep maximum 10 files on server
echo exif_imagetype($_FILES['Filedata']);
if ($_FILES['Filedata']['size'] <= $MAXIMUM_FILESIZE)
{
    move_uploaded_file($_FILES['Filedata']['tmp_name'],
"./temporary/".$_FILES['Filedata']['name']);
    $type = exif_imagetype("./temporary/".$_FILES['Filedata']['name']);
    if ($type == 1 || $type == 2 || $type == 3)
    {
        rename("./temporary/".$_FILES['Filedata']['name'],
"./images/".$_FILES['Filedata']['name']);
    }
    else
    {
        unlink("./temporary/".$_FILES['Filedata']['name']);
    }
}
$directory = opendir('./images/');
$files = array();
while ($file = readdir($directory))
{
    array_push($files, array('./images/'.$file, filectime('./images/'.$file)));
}
usort($files, sorter);
if (count($files) > $MAXIMUM_FILE_COUNT)
{
    $files_to_delete = array_splice($files, 0, count($files) - $MAXIMUM_FILE_COUNT);
    for ($i = 0; $i < count($files_to_delete); $i++)
    {
        unlink($files_to_delete[$i][0]);
    }
}
print_r($files);
closedir($directory);

function sorter($a, $b)
{
    if ($a[1] == $b[1])
    {
        return 0;
    }
    else
    {
        return ($a[1] < $b[1]) ? -1 : 1;
    }
}
?>
```

В сценарий отправки можно передать дополнительные переменные с помощью метода запроса `POST` или `GET`. Чтобы отправить дополнительные переменные `POST` сценарию отправки, можно использовать следующий код:

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();
function selectHandler(event:Event):void
{
    var params:URLVariables = new URLVariables();
    params.date = new Date();
    params.ssid = "94103-1394-2345";
    var request:URLRequest = new
URLRequest("http://www.yourdomain.com/FileReferenceUpload/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    request.data = params;
    fileRef.upload(request, "Custom1");
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

В предыдущем примере создается объект URLVariables, который передается сценарию, выполняемому на удаленном сервере. В предыдущих версиях ActionScript переменные можно передавать сценарию отправки, выполняемому на стороне сервера, путем передачи значений в строке запроса. ActionScript 3.0 позволяет передавать переменные удаленному сценарию с помощью объекта URLRequest, который позволяет передавать данные с помощью метода `POST` или `GET`. Это, в свою очередь, упрощает и ускоряет передачу более объемных пакетов данных. Чтобы указать метод запроса, с помощью которого передаются переменные (`GET` или `POST`), можно задать свойству `URLRequest.method` значение `URLRequestMethod.GET` или `URLRequestMethod.POST`.

Кроме того, ActionScript 3.0 позволяет переопределять стандартное имя поля отправки данных `Filedata` благодаря использованию второго параметра в методе `upload()`, как показано в предыдущем примере, где значение `Filedata` заменено на `Custom1`.

По умолчанию проигрыватель Flash Player не пытается отправлять тестовый пакет данных. Чтобы переопределить это поведение, передайте значение `true` в качестве третьего параметра метода `upload()`. Тестовая отправка выполняется для того, чтобы проверить, будет ли успешной передача фактических файлов и проверка подлинности сервера (если она требуется).

***Примечание.*** *В настоящее время тестовая отправка поддерживается проигрывателем Flash Player только в ОС Windows.*

Сценарий сервера, обрабатывающий отправку файла, должен ожидать HTTP-запрос `POST` со следующими элементами:

- `Content-Type` (тип содержимого) со значением `multipart/form-data.`
- `Content-Disposition` (расположение содержимого), у которого атрибут `name` имеет значение `Filedata`, а атрибут `filename` указывает имя исходного файла. Можно указать пользовательский атрибут `name`, передав значение для параметра `uploadDataFieldName` в методе `FileReference.upload()`.
- Двоичное содержимое файла.

Ниже приводится пример HTTP-запроса `POST`:

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filedata"; filename="sushi.jpg"
Content-Type: application/octet-stream

Test File
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
(actual file data,,,)
```

Следующий пример HTTP-запроса `POST` отправляет три переменные `POST`: `api_sig`, `api_key` и `auth_token`, а также использует пользовательское имя поля для отправки данных `"photo"`.

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_sig"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_key"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="auth_token"

XXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="photo"; filename="sushi.jpg"
Content-Type: application/octet-stream

(actual file data,,,)
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7--
```

## Загрузка файлов с сервера

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы предоставить пользователям возможность загружать файлы с сервера, можно использовать метод `FileReference.download()` с двумя параметрами: `request` и `defaultFileName`. Первый параметр представляет собой объект URLRequest, содержащий URL-адрес файла для загрузки. Второй параметр не является обязательным. Он позволяет указать имя файла по умолчанию, которое отображается в диалоговом окне загрузки файла. Если опустить второй параметр, `defaultFileName`, будет использоваться имя файла из указанного URL-адреса.

Следующий код загружает файл с именем index.xml из того же каталога, в котором находится SWF-файл.

```
var request:URLRequest = new URLRequest("index.xml");
var fileRef:FileReference = new FileReference();
fileRef.download(request);
```

Чтобы задать имя по умолчанию currentnews.xml вместо index.xml, укажите параметр `defaultFileName`, как показано в следующем фрагменте кода.

```
var request:URLRequest = new URLRequest("index.xml");
var fileToDownload:FileReference = new FileReference();
fileToDownload.download(request, "currentnews.xml");
```

Переименование файла полезно, если имя файла на сервере непонятное или сгенерировано сервером. Параметр `defaultFileName` также полезно указывать, когда файл загружается не напрямую, а с использованием сценария, выполняемого на стороне сервера. Например, необходимо задать параметр `defaultFileName`, если на сервере имеется сценарий, загружающий определенные файлы исходя из переданных ему переменных URL. В противном случае именем по умолчанию для загруженного файла будет имя сценария, выполняемого на стороне сервера.

Данные можно отправлять на сервер с помощью метода `download()` путем добавления параметров к URL, которые должны анализироваться сценарием сервера. Следующий код ActionScript 3.0 загружает документ на базе параметров, передаваемых сценарию ColdFusion.

```
package
{
    import flash.display.Sprite;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.net.URLRequestMethod;
    import flash.net.URLVariables;

    public class DownloadFileExample extends Sprite
    {
        private var fileToDownload:FileReference;
        public function DownloadFileExample()
        {
            var request:URLRequest = new URLRequest();
            request.url = "http://www.[yourdomain].com/downloadfile.cfm";
            request.method = URLRequestMethod.GET;
            request.data = new URLVariables("id=2");
            fileToDownload = new FileReference();
            try
            {
                fileToDownload.download(request, "file2.txt");
            }
            catch (error:Error)
            {
                trace("Unable to download file.");
            }
        }
    }
}
```

Следующий код демонстрирует сценарий ColdFusion (download.cfm), который загружает один из двух файлов с сервера в зависимости от значения переменной URL.

```
<cfparam name="URL.id" default="1" />
<cfswitch expression="#URL.id#">
    <cfcase value="2">
        <cfcontent type="text/plain" file="#ExpandPath('two.txt')#" deletefile="No" />
    </cfcase>
    <cfdefaultcase>
        <cfcontent type="text/plain" file="#ExpandPath('one.txt')#" deletefile="No" />
    </cfdefaultcase>
</cfswitch>
```

## класс FileReferenceList

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс FileReferenceList дает пользователю возможность выбрать один или несколько файлов для отправки сценарию, выполняемому на стороне сервера. Отправка файла обрабатывается методом `FileReference.upload()`, который необходимо вызывать для каждого файла, выбранного пользователем.

Следующий код создает два объекта FileFilter (`imageFilter` и `textFilter`) и передает их в массиве методу `FileReferenceList.browse()`. В результате этого в системном диалоговом окне открытия файла отображаются два возможных фильтра для типов файлов.

```
var imageFilter:FileFilter = new FileFilter("Image Files (*.jpg, *.jpeg, *.gif, *.png)",
"*.jpg; *.jpeg; *.gif; *.png");
var textFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt; *.rtf");
var fileRefList:FileReferenceList = new FileReferenceList();
try
{
    var success:Boolean = fileRefList.browse(new Array(imageFilter, textFilter));
}
catch (error:Error)
{
    trace("Unable to browse for files.");
}
```

Использование класса FileReferenceList обеспечивает возможность выбирать и отправлять один или несколько файлов аналогично тому, как метод `FileReference.browse()` дает возможность выбирать файлы, только FileReferenceList позволяет выбирать сразу несколько файлов. Для отправки нескольких файлов требуется вызывать для каждого из них метод `FileReference.upload()`, как показано в следующем коде.

```
var fileRefList:FileReferenceList = new FileReferenceList();
fileRefList.addEventListener(Event.SELECT, selectHandler);
fileRefList.browse();

function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest("http://www.[yourdomain].com/fileUploadScript.cfm");
    var file:FileReference;
    var files:FileReferenceList = FileReferenceList(event.target);
    var selectedFileArray:Array = files.fileList;
    for (var i:uint = 0; i < selectedFileArray.length; i++)
    {
        file = FileReference(selectedFileArray[i]);
        file.addEventListener(Event.COMPLETE, completeHandler);
        try
        {
            file.upload(request);
        }
        catch (error:Error)
        {
            trace("Unable to upload files.");
        }
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

Так как событие `Event.COMPLETE` добавляется к каждому объекту FileReference в массиве, проигрыватель Flash Player вызывает метод `completeHandler()` после отправки отдельного файла.

# Использование API-интерфейса файловой системы AIR

**Adobe AIR 1.0 и более поздних версий**

API файловой системы Adobe AIR включает в себя следующие классы:

- File;
- FileMode;
- FileStream.

API-интерфейс файловой системы позволяет выполнять следующие (а также иные) действия:

- копировать, создавать, удалять и перемещать файлы и каталоги;
- получать информацию о файлах и каталогах;
- читать и записывать файлы.

## Основные сведения о файлах AIR

**Adobe AIR 1.0 и более поздних версий**

Краткое описание работы с файловой системой в AIR и примеры кода можно найти в следующих статьях по быстрому началу работы в центре Adobe Developer Connection:

- Создание редактора текстовых файлов (Flash)
- Создание текстового редактора (Flex)
- Создание приложения для поиска по каталогу (Flex)
- Чтение и запись в XML-файл установок (Flex)
- Сжатие файлов и данных (Flex)

В Adobe AIR есть классы, используемые для создания, доступа и управления файлами и папками. Они содержатся в пакете flash.filesystem. Ниже указаны способы их применения:

| Классы File | Описание |
|---|---|
| File | Объект File представляет собой путь к файлу или каталогу. Объект File используется для создания указателя на файл или папку для взаимодействия с ними. |
| FileMode | Класс FileMode определяет строковые константы, которые применяются в параметре `fileMode` методов `open()` и `openAsync()` класса FileStream. Параметр `fileMode` этих методов определяет доступные действия объекта FileStream при открытии файла, включая запись, чтение, добавление и обновление данных. |
| FileStream | Объект FileStream открывает файлы для чтения и записи. После создания объекта File, указывающего на новый или существующий файл, этот указатель передается объекту FileStream, чтобы данные в файле можно было открывать и считывать или записывать их. |

Некоторые методы класса File имеют как синхронную, так и асинхронную версии:

- `File.copyTo()` и `File.copyToAsync()`
- `File.deleteDirectory()` и `File.deleteDirectoryAsync()`
- `File.deleteFile()` и `File.deleteFileAsync()`
- `File.getDirectoryListing()` и `File.getDirectoryListingAsync()`
- `File.moveTo()` и `File.moveToAsync()`
- `File.moveToTrash()` и `File.moveToTrashAsync()`

Кроме того, операции FileStream могут выполняться синхронно или асинхронно в зависимости от того, каким методом объект FileStream открывает файл: `open()` или `openAsync()`.

Асинхронные версии позволяют запускать фоновые процессы и отправлять события после их завершения (или в случае ошибки). Во время работы этих асинхронных фоновых процессов может исполняться другой код. Для асинхронных версий операций необходимо настроить функции прослушивателей событий с помощью метода `addEventListener()` объекта File или FileStream, вызывающего функцию.

Синхронные версии упрощают написание кода, так как он не зависит от прослушивателей событий. Тем не менее, так как вместе с синхронным методом не может исполняться другой код, важные процессы, например визуализация экранных объектов и анимация, могут откладываться.

# Работа с объектами File в AIR

**Adobe AIR 1.0 и более поздних версий**

Объект File является указателем на файл или каталог в файловой системе.

Класс File является расширением класса FileReference. Класс FileReference, встречающийся в проигрывателе Adobe® Flash® и среде AIR™, является указателем на файл. В класс File добавлены свойства и методы, из соображений безопасности не представленные в проигрывателе Flash (при проигрывании SWF-файла в браузере).

## О классе File

**Adobe AIR 1.0 и более поздних версий**

Класс File имеет целый ряд применений:

- Получение путей к определенным каталогам, включая каталог пользователей, каталог документов пользователей, каталог запуска приложения и каталог самого приложения.
- Копирование файлов и каталогов
- Перемещение файлов и каталогов
- Удаление файлов и каталогов (или перемещение в корзину)
- Составление списка файлов и каталогов внутри каталога
- Создание временных файлов и папок

Объект File, указывающий путь к файлу, можно использовать для чтения и записи данных с помощью класса FileStream.

Объект File может указывать путь к еще не существующему файлу или каталогу. В этом случае этот объект можно использовать для создания файла или каталога.

## Пути к объектам File

**Adobe AIR 1.0 и более поздних версий**

У каждого объекта File есть два свойства, определяющие его путь:

| Свойство | Описание |
|---|---|
| `nativePath` | Указывает путь к файлу с учетом конкретной платформы. Например, в Windows это может быть «C:\Sample directory\test.txt», а в Mac OS — «/Sample directory/test.txt». В Windows в свойстве `nativePath` в качестве разделителя используется обратная косая черта (\), а в Mac OS и Linux — прямая косая черта (/). |
| `url` | Это может быть схема URL-адресов для файлов, указывающая на файл. Например, в Windows это может быть «file:///c:/Sample%20directory/test.txt», а в Mac OS — «file:///Sample%20directory/test.txt». Помимо схемы `file`, среда выполнения включает и другие специальные URL-схемы, подробнее описанные в разделе «Поддерживаемые AIR схемы URL-адресов» на странице 727 |

Класс File включает в себя статические свойства для указания на стандартные каталоги в ОС Mac OS, Windows и Linux. К таким свойствам относятся:

- `File.applicationStorageDirectory` — каталог хранилища, который является уникальным для каждого установленного приложения AIR. Этот каталог подходит для хранения динамических активов приложений и настроек пользователя. Большие объемы данных рекомендуется хранить в другом месте.

  На устройствах Android и iOS каталог хранилища приложения удаляется при удалении приложения с устройства или когда пользователь выполняет очистку данных приложения, однако на других платформах такого не происходит.

- `File.applicationDirectory` — каталог, в который устанавливается приложение (вместе с установленными активами). В некоторых операционных системах приложение хранится в одном пакетном файле, а не в физическом каталоге. В этом случае содержимое может стать недоступным, если используется собственный путь. Каталог приложения доступен только для чтения.
- `File.desktopDirectory` — каталог рабочего стола пользователя. Если платформа не определяет каталог рабочего стола, используется другой каталог в файловой системе.
- `File.documentsDirectory` — каталог с документами пользователя. Если платформа не определяет каталог документов, используется другой каталог в файловой системе.
- `File.userDirectory` — каталог пользователя. Если платформа не определяет каталог пользователя, используется другой каталог в файловой системе.

***Примечание.*** *Если платформа не определяет стандартные каталоги рабочего стола, каталог документов пользователя и каталог пользователя,* `File.documentsDirectory`*,* `File.desktopDirectory` *и* `File.userDirectory` *могут указывать на один и тот же каталог.*

В различных операционных системах эти свойства имеют разные значения. Например, в ОС Mac и Windows собственные пути к каталогу рабочего стола пользователя различаются. Однако свойство `File.desktopDirectory` указывает соответствующий путь к каталогу на каждой платформе. При создании приложений для работы на различных платформах используйте эти свойства как основу для указания на другие каталоги и файлы, используемые приложением. Затем используйте метод `resolvePath()` для коррекции пути. Например, данный код указывает на файл preferences.xml в каталоге хранилища приложения:

```
var prefsFile:File = File.applicationStorageDirectory;
prefsFile = prefsFile.resolvePath("preferences.xml");
```

Хотя класс File позволяет указывать определенный путь к файлу, он может привести к приложениям, которые не могут работать на разных платформах. Например, путь C:\Documents and Settings\joe\ работает только в ОС Windows. По этим причинам лучше использовать статические свойства класса File, такие как `File.documentsDirectory`.

## Типовое расположение каталогов

| Платформа | Тип каталога | Стандартное расположение в файловой системе |
|---|---|---|
| Android | Приложение | `/data/data/` |
| | Хранилище приложения | `/data/data/air.applicationID/filename/Local Store` |
| | Кэш | /data/data/applicationID/cache |
| | Настольный компьютер | `/mnt/sdcard` |
| | Документы | `/mnt/sdcard` |
| | Временный | `/data/data/applicationID/cache/FlashTmp.randomString` |
| | Пользователь | `/mnt/sdcard` |
| iOS | Приложение | `/var/mobile/Applications/uid/filename.app` |
| | Хранилище приложения | `/var/mobile/Applications/uid/Library/Application Support/applicationID/Local Store` |
| | Кэш | /var/mobile/Applications/uid/Library/Caches |
| | Настольный компьютер | недоступен |
| | Документы | `/var/mobile/Applications/uid/Documents` |
| | Временный | `/private/var/mobile/Applications/uid/tmp/FlashTmpNNN` |
| | Пользователь | недоступен |
| Linux | Приложение | `/opt/filename/share` |
| | Хранилище приложения | `/home/userName/.appdata/applicationID/Local Store` |
| | Настольный компьютер | `/home/userName/Desktop` |
| | Документы | `/home/userName/Documents` |
| | Временный | `/tmp/FlashTmp.randomString` |
| | Пользователь | `/home/userName` |

| Платформа | Тип каталога | Стандартное расположение в файловой системе |
|---|---|---|
| Mac | Приложение | `/Applications/filename.app/Contents/Resources` |
| | Хранилище приложения | `/Users/`***`userName`***`/Library/Preferences/`***`applicationid`***`/Local Store` (AIR 3.2 и более ранних версий)<br>***`path`***`/Library/Application Support/`***`applicationid`***`/Local Store` (AIR 3.3 и более поздних версий), где path — это `/Users/`***`userName`***`/Library/Containers/`***`bundle-id`***`/Data` (изолированная среда) или `/Users/userName` (при выполнении за пределами изолированной среды) |
| | Кэш | /Users/userName/Library/Caches |
| | Настольный компьютер | `/Users/`***`userName`***`/Desktop` |
| | Документы | `/Users/`***`userName`***`/Documents` |
| | Временный | `/private/var/folders/JY/randomString/TemporaryItems/FlashTmp` |
| | Пользователь | `/Users/`***`userName`*** |
| Windows | Приложение | `C:\Program Files\filename` |
| | Хранилище приложения | `C:\Documents and settings\userName\ApplicationData\applicationID\Local Store` |
| | Кэш | C:\Documents and settings\userName\Local Settings\Temp |
| | Настольный компьютер | `C:\Documents and settings\userName\Desktop` |
| | Документы | `C:\Documents and settings\userName\My Documents` |
| | Временный | `C:\Documents and settings\userName\Local Settings\Temp\randomString.tmp` |
| | Пользователь | `C:\Documents and settings\userName` |

Фактические собственные пути к этим каталогам варьируются в зависимости от операционной системы и конфигурации компьютера. Пути, показанные в данной таблице, являются типовыми примерами. При ссылке на эти каталоги всегда следует использовать соответствующие статические свойства класса File. В приложении AIR значения свойств `applicationID` и `filename`, приведенные в таблице, взяты из дескриптора приложения. Если в дескрипторе приложения указано свойство publisher ID, идентификатор издателя будет добавлен к идентификатору приложения в этих путях. Значением свойства `userName` является имя учетной записи пользователя, выполняющего установку.

## Объект File, указывающий на каталог

**Adobe AIR 1.0 и более поздних версий**

Существует ряд способов настроить объект File так, чтобы он указывал на каталог.

### Указание на начальный каталог пользователя

**Adobe AIR 1.0 и более поздних версий**

Объект File может указывать на начальный каталог пользователя. В приведенном ниже коде объект File указывает на подкаталог AIR Test начального каталога:

```
var file:File = File.userDirectory.resolvePath("AIR Test");
```

### Указание на каталог документов пользователя

**Adobe AIR 1.0 и более поздних версий**

Объект File может указывать на каталог документов пользователя. В приведенном ниже коде объект File указывает на подкаталог AIR Test каталога документов:

```
var file:File = File.documentsDirectory.resolvePath("AIR Test");
```

### Указание на каталог рабочего стола пользователя

**Adobe AIR 1.0 и более поздних версий**

Объект File может указывать на каталог рабочего стола. В приведенном ниже коде объект File указывает на подкаталог AIR Test рабочего стола:

```
var file:File = File.desktopDirectory.resolvePath("AIR Test");
```

### Указание на каталог хранилища приложения

**Adobe AIR 1.0 и более поздних версий**

Объект File может указывать на каталог хранилища приложения. В каждом приложении AIR есть путь, определяющий каталог хранилища данных этого приложения. Этот каталог уникален для каждого приложения и каждого пользователя. Его можно использовать для хранения информации, связанной с конкретным пользователем или приложением (например, файлы данных или настроек). Например, в приведенном ниже коде объект File указывает на файл настроек prefs.xml, содержащийся в каталоге хранилища приложения:

```
var file:File = File.applicationStorageDirectory;
file = file.resolvePath("prefs.xml");
```

Местоположение каталога хранилища программы обычно складывается из имени пользователя и идентификатора приложения. Представленная ниже информация о местоположениях в файловой системе будет полезна при выполнении отладки приложения. Для нахождения файлов в этом каталоге всегда используйте свойство `File.applicationStorage` или схему URI `app-storage:`

- В ОС Mac OS — зависит от версии AIR:

  **AIR 3.2 и более ранних версий**: `/Users/user name/Library/Preferences/`*`applicationID`*`/Local Store/`

  **AIR 3.3 и более поздних версий**: *`path`*`/Library/Application Support/`*`applicationID`*`/Local Store`, где *`path`* — это `/Users/`*`username`*`/Library/Containers/`*`bundle-id`*`/Data` (изолированная среда) или `/Users/`*`username`* (при выполнении за пределами изолированной среды)

  Например (AIR 3.2):

  ```
  /Users/babbage/Library/Preferences/com.example.TestApp/Local Store
  ```

- В Windows — в каталоге Documents and Settings, в папке:

  *C:\Documents and Settings\имя пользователя*`\Application Data\`*applicationID*`\Local Store\`

  Например:

  ```
  C:\Documents and Settings\babbage\Application Data\com.example.TestApp\Local Store
  ```

- В Linux — в каталоге:

/home/*имя пользователя*/.appdata/*идентификатор приложения*/Local Store/

Например:

/home/babbage/.appdata/com.example.TestApp/Local Store

- В Android:

  /data/data/androidPackageID/applicationID/Local Store

  Например:

  /data/data/air.com.example.TestApp/com.example.TestApp/Local Store

***Примечание.*** *Если у приложения имеется идентификатор издателя, то он также используется для указания пути к каталогу хранилища этого приложения.*

URL-адрес (и свойство `url`) объекта File, созданного с помощью `File.applicationStorageDirectory`, использует схему URL-адресов `app-storage` (см. раздел «Поддерживаемые AIR схемы URL-адресов» на странице 727), как показано ниже:

```
var dir:File = File.applicationStorageDirectory;
dir = dir.resolvePath("preferences");
trace(dir.url); // app-storage:/preferences
```

### Указание на каталог приложения

**Adobe AIR 1.0 и более поздних версий**

Объект File может указывать на каталог, в который установлено приложение (т.н. каталог приложения). На этот каталог можно ссылаться с помощью свойства `File.applicationDirectory`. В нем можно найти файл дескриптора и другие ресурсы, установленные вместе с приложением. Например, в приведенном ниже коде объект File указывает на каталог *images* внутри каталога приложения:

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("images");
```

URL-адрес (и свойство `url`) объекта File, созданного с помощью `File.applicationDirectory`, использует схему URL-адресов `app` (см. раздел «Поддерживаемые AIR схемы URL-адресов» на странице 727), как показано ниже:

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("images");
trace(dir.url); // app:/images
```

***Примечание.*** *В Android свойство `nativePath` не может быть использовано для получения доступа к файлам в пакете приложения. Свойство `nativePath` представляет собой пустую строку. При обращении к файлам в каталоге приложения всегда используйте URL вместо собственного пути.*

### Указание на каталог кэша

**Adobe AIR 3.6 и более поздних версий**

Объекту File можно указать на временный каталог операционной системы (каталог кэша) при помощи свойства `File.cacheDirectory`. Такой каталог содержит временные файлы, не являющиеся необходимыми для работы приложений и не вызывающие проблем или потери данных пользователя при удалении.

На большинстве операционных систем каталог кэша является временным каталогом. В системе iOS каталог кэша соответствует каталогу Caches библиотеки приложений. Файлы в этом каталоге не подлежат резервному копированию на серверное хранилище и могут быть удалены операционной системой, если заканчивается свободное место на устройстве. Дополнительные сведения см. в разделе «Управление резервным копированием и кэшированием файлов» на странице 728.

### Указание на корневой каталог файловой системы

**Adobe AIR 1.0 и более поздних версий**

Метод `File.getRootDirectories()` составляет список всех корневых томов, таких как «C:», и подключенных томов на компьютере Windows. В ОС Mac и Linux этот метод всегда возвращает уникальный корневой каталог компьютера (каталог «/»). Метод `StorageVolumeInfo.getStorageVolumes()` предоставляет дополнительные сведения о подключенных томах запоминающих устройств (см. раздел ««Работа с томами запоминающих устройств» на странице 739»).

***Примечание.*** *В Android отсутствует доступ для чтения корневого каталога файловой системы. Возвращается объект File, указывающий на каталог с собственным путем «/», однако свойства этого объекта не имеют точных значений. Например, свойство* `spaceAvailable` *всегда равно 0.*

### Указание на явно заданный каталог

**Adobe AIR 1.0 и более поздних версий**

Объект File может указывать на явно заданный каталог. Для этого нужно задать свойство `nativePath` объекта File, как показано в примере ниже (в Windows):

```
var file:File = new File();
file.nativePath = "C:\\AIR Test";
```

**Важно:** такое указание явного пути может привести к тому, что код не будет работать на разных платформах. Например, предыдущий пример работоспособен только в ОС Windows. Для определения местонахождения каталога, работающего с разными платформами, можно использовать статические свойства объекта File, например `File.applicationStorageDirectory`. Для навигации по относительному пути используйте метод `resolvePath()` (см. следующий раздел).

### Навигация с использованием относительных путей

**Adobe AIR 1.0 и более поздних версий**

Метод `resolvePath()` может использоваться для получения пути относительно другого пути. Например, объект File указывает на подкаталог AIR Test начального каталога пользователя.

```
var file:File = File.userDirectory;
file = file.resolvePath("AIR Test");
```

Свойство `url` объекта File может указывать на каталог на основе URL-строки, как показано ниже:

```
var urlStr:String = "file:///C:/AIR Test/";
var file:File = new File()
file.url = urlStr;
```

Дополнительные сведения см. в разделе ««Изменение путей к файлу» на странице 727».

### Разрешение пользователю обзора для выбора каталога

**Adobe AIR 1.0 и более поздних версий**

Класс File включает метод `browseForDirectory()`, выводящий системное диалоговое окно, в котором пользователь может выбрать каталог и назначить его объекту. Метод `browseForDirectory()` является асинхронным. Объект File отправляет событие `select`, если пользователь выбирает каталог и нажимает кнопку «Открыть», или событие `cancel`, если пользователь нажимает кнопку «Отмена».

Например, в приведенном ниже коде пользователь может выбрать каталог и на основе этого выбора создать путь вывода:

```
var file:File = new File();
file.addEventListener(Event.SELECT, dirSelected);
file.browseForDirectory("Select a directory");
function dirSelected(e:Event):void {
    trace(file.nativePath);
}
```

***Примечание.*** *В Android метод `browseForDirectory()` не поддерживается. Вызов этого метода не дает никаких результатов. Моментально отправляется событие cancel. Чтобы разрешить пользователям выбирать каталог, используйте вместо этого пользовательское диалоговое окно, определенное приложением.*

### Указание на каталог, из которого запускается приложение

**Adobe AIR 1.0 и более поздних версий**

Расположение каталога, из которого запускается приложение, можно получить, проверив свойство `currentDirectory` объекта InvokeEvent, отправляемого при запуске приложения. Дополнительные сведения см. в разделе ««Захват аргументов командной строки» на странице 944».

## Объект File, указывающий на файл

**Adobe AIR 1.0 и более поздних версий**

Существует несколько способов задания файла, на который указывает объект File.

### Указание на явно заданный путь к файлу

**Adobe AIR 1.0 и более поздних версий**

**Важно:** указание явного пути может привести к тому, что код не будет работать на разных платформах. Например, использование пути C:/foo.txt применимо только в ОС Windows. Для определения местонахождения каталога, работающего с разными платформами, можно использовать статические свойства объекта File, например `File.applicationStorageDirectory`. Для навигации по относительному пути используйте метод `resolvePath()` (см. раздел ««Изменение путей к файлу» на странице 727»).

Свойство `url` объекта File может указывать на файл или каталог на основе URL-строки, как показано ниже:

```
var urlStr:String = "file:///C:/AIR Test/test.txt";
var file:File = new File()
file.url = urlStr;
```

Также можно передать функции конструктора `File()` URL-адрес, как показано ниже:

```
var urlStr:String = "file:///C:/AIR Test/test.txt";
var file:File = new File(urlStr);
```

Свойство `url` всегда возвращает URL в формате URI (например, пробелы замещаются знаками `"%20`):

```
file.url = "file:///c:/AIR Test";
trace(file.url); // file:///c:/AIR%20Test
```

Также свойство `nativePath` объекта File можно использовать для явного задания пути. Например, если приведенный ниже код запустить на компьютере под управлением Windows, то объект File будет указывать на файл text.txt в подкаталоге AIR Test на диске «C:»:

```
var file:File = new File();
file.nativePath = "C:/AIR Test/test.txt";
```

Также можно передать этот путь функции конструктора `File()`, как показано ниже:

```
var file:File = new File("C:/AIR Test/test.txt");
```

В качестве разделителя пути к файлу в свойстве `nativePath` следует использовать переднюю косую черту (/). В Windows можно также использовать обратную косую черту (\), но при этом приложение может не работать на других платформах.

Дополнительные сведения см. в разделе ««Изменение путей к файлу» на странице 727».

### Нумерация файлов в каталоге

**Adobe AIR 1.0 и более поздних версий**

С помощью метода `getDirectoryListing()` объекта File можно получить массив объектов File, указывающих на файлы и подкаталоги в корневом каталоге. Дополнительные сведения см. в разделе ««Нумерация каталогов» на странице 735».

### Разрешение пользователю обзора для выбора файла

**Adobe AIR 1.0 и более поздних версий**

Класс File включает следующие методы, выводящие системное диалоговое окно, в котором пользователь может выбрать файл и назначить его объекту:

- `browseForOpen()`
- `browseForSave()`
- `browseForOpenMultiple()`

Все они асинхронны. Методы `browseForOpen()` и `browseForSave()` отправляют событие select, когда пользователь выбирает файл (или, в случае метода browseForSave(), целевой путь). В случае методов `browseForOpen()` и `browseForSave()`, выбранный целевой объект File указывает на выбранные файлы. Метод `browseForOpenMultiple()` отправляет событие `selectMultiple`, когда пользователь выбирает файлы. Событие `selectMultiple` относится к типу FileListEvent, имеющему свойство `files`, которое является массивом объектов File (указывающих на выбранные файлы).

Например, в приведенном ниже коде пользователь выбирает файл в диалоговом окне «Открыть»:

```
 var fileToOpen:File = File.documentsDirectory;
selectTextFile(fileToOpen);

function selectTextFile(root:File):void
{
    var txtFilter:FileFilter = new FileFilter("Text", "*.as;*.css;*.html;*.txt;*.xml");
    root.browseForOpen("Open", [txtFilter]);
    root.addEventListener(Event.SELECT, fileSelected);
}

function fileSelected(event:Event):void
{
    trace(fileToOpen.nativePath);
}
```

Если при вызове метода browse в приложении уже открыто другое диалоговое окно, среда выполнения генерирует исключение Error.

***Примечание.*** *В Android с помощью методов `browseForOpen()` и `browseForOpenMultiple()` можно выбирать только файлы изображений, видео и аудио. В диалоговом окне browseForSave() также отображаются только файлы мультимедиа, несмотря на то что пользователь имеет возможность ввести произвольное имя файла. Для открытия и сохранения других файлов, кроме файлов мультимедиа, рассмотрите возможность использования пользовательских диалоговых окон вместо этих методов.*

## Изменение путей к файлу

**Adobe AIR 1.0 и более поздних версий**

Путь к существующему объекту File можно изменить, вызвав метод `resolvePath()` или изменив свойство `nativePath` или `url` объекта, как показано ниже (для Windows):

```
var file1:File = File.documentsDirectory;
file1 = file1.resolvePath("AIR Test");
trace(file1.nativePath); // C:\Documents and Settings\userName\My Documents\AIR Test
var file2:File = File.documentsDirectory;
file2 = file2.resolvePath("..");
trace(file2.nativePath); // C:\Documents and Settings\userName
var file3:File = File.documentsDirectory;
file3.nativePath += "/subdirectory";
trace(file3.nativePath); // C:\Documents and Settings\userName\My Documents\subdirectory
var file4:File = new File();
file4.url = "file:///c:/AIR Test/test.txt";
trace(file4.nativePath); // C:\AIR Test\test.txt
```

При использовании свойства `nativePath` в качестве разделителя пути к файлу используйте переднюю косую черту (/). В Windows можно также использовать обратную косую черту (\), но при этом приложение может не работать на других платформах.

## Поддерживаемые AIR схемы URL-адресов

**Adobe AIR 1.0 и более поздних версий**

В AIR для определения свойства `url` объекта File могут использоваться следующие схемы URL-адресов.

| Схема URL-адресов | Описание |
|---|---|
| file | Используется для задания путей относительно корневого тома файловой системы. Например:<br>`file:///c:/AIR Test/test.txt`<br>Стандарт URL требует, чтобы URL-адрес файла записывался в виде `file://<host>/<path>`. В отдельных случаях `<host>` может быть пустой строкой, которая интерпретируется как «компьютер, с которого получается интерпретируемый URL-адрес». По этой причине URL-адреса файлов часто обозначаются тремя косыми чертами (///). |
| app | Используется для задания путей относительно корневого каталога установленного приложения (каталога, содержащего файл application.xml установленного приложения). Например, следующий путь указывает на подкаталог с изображениями в каталоге установленного приложения:<br>`app:/images` |
| app-storage | Используется для задания путей относительно каталога хранилища приложения. AIR определяет для каждого установленного приложения уникальный каталог хранилища, куда сохраняются все данные, относящиеся к этому приложению. Например, следующий путь указывает на файл prefs.xml в подкаталоге параметров каталога хранилища приложения:<br>`app-storage:/settings/prefs.xml` |

## Управление резервным копированием и кэшированием файлов

**Adobe AIR 3.6 и более поздние версии, только в ОС iOS и OS X**

Некоторые операционные системы, в частности, iOS и Mac OS X, имеют возможность автоматического резервного копирования файлов приложений на удаленное хранилище. Кроме того, в iOS существуют определенные правила, касающиеся резервного копирования и мест хранения файлов с различным назначением.

Ниже приведен краткий обзор этих правил — рекомендаций Apple в отношении резервного копирования и хранения файлов. Дополнительная информация приведена в следующих разделах.

- Если файл не подлежит резервному копированию и (только в iOS) может быть удален операционной системой при сокращении свободного места на устройстве, его следует сохранить в каталог кэша (`File.cacheDirectory`). Этот каталог является предпочтительным местом хранения файлов на iOS, его следует использовать для большинства файлов, которые могут быть созданы или загружены заново.
- Если файл не подлежит резервному копированию, но также не подлежит удалению операционной системой, его следует сохранить в один их каталогов библиотеки приложений, например в каталог хранилища приложений (`File.applicationStorageDirectory`) или в каталог документов (`File.documentsDirectory`). Свойство `preventBackup` такого объекта File следует задать как `true`. Таковы рекомендации Apple в отношении содержимого, которое можно восстановить или загрузить заново, но которое необходимо для правильной работы приложения в автономном режиме.

### Назначение файлов для резервного копирования

В целях экономии пропускной способности и места в резервном хранилище рекомендации Apple для приложений систем iOS и Mac гласят, что для резервного копирования следует назначать только те файлы, которые содержат пользовательские данные или данные, которые невозможно восстановить или загрузить заново.

По умолчанию резервному копированию подвергаются все файлы в каталогах библиотеки приложений. В системе Mac OS X это каталог хранилища приложений (application storage). В системе iOS это каталог хранилища приложений (application storage), каталог приложений (application), каталог рабочего стола (desktop), каталог документов (documents) и каталог пользователя (user) — эти каталоги соответствуют каталогам библиотеки приложений в iOS. Соответственно, все файлы в этих каталогах по умолчанию подвергаются резервному копированию на серверное хранилище.

При сохранении в один из этих каталогов файла, который может быть создан приложением заново, следует отметить особым флагом, что этот файл не подлежит резервному копированию операционной системой. Для этого необходимо задать у этого объекта File свойство `preventBackup` со значением `true`.

Обратите внимание, что в iOS при размещении файла в любом из каталогов библиотеки приложений, даже если свойство `preventBackup` файла имеет значение `true`, этот файл помечается как постоянный и не подлежит удалению операционной системой.

**Управление кэшированием и удалением файлов**

Рекомендации Apple для приложений iOS гласят, что операционная система должна иметь возможность в случае, если устройство испытывает недостаток места хранения, удалить как можно больший объем содержимого, подлежащего восстановлению.

В iOS файлы в каталогах библиотеки приложений (например, в каталоге хранилища приложений или в каталоге документов) помечаются как постоянные и не подлежат удалению операционной системой.

Файлы, которые могут быть созданы приложением заново и которые можно без опаски удалить в случае недостатка свободного места, следует хранить в каталоге кэша приложений. Доступ к каталогу кэша осуществляется через статическое свойство `File.cacheDirectory`.

В системе iOS этот каталог кэша соответствует каталогу кэша приложений (<Application Home>/Library/Caches). В других операционных системах этому свойству так же присваивается путь к соответствующему каталогу. К примеру, в Mac OS X это тот же каталог Caches в библиотеке приложений. В ОС Android каталог кэша соответствует каталогу кэша приложений. В ОС Windows каталог кэша соответствует временному каталогу операционной системы. И на Android, и на Windows именно в этом местоположении создаются временные файлы и каталоги при вызове методов `createTempDirectory()` и `createTempFile()` класса File.

## Нахождение одного пути к файлу относительно другого

**Adobe AIR 1.0 и более поздних версий**

Метод `getRelativePath()` может использоваться для нахождения пути к одному файлу относительно другого:

```
 var file1:File = File.documentsDirectory.resolvePath("AIR Test");
var file2:File = File.documentsDirectory
file2 = file2.resolvePath("AIR Test/bob/test.txt");

trace(file1.getRelativePath(file2)); // bob/test.txt
```

Второй параметр метода `getRelativePath()` — `useDotDot` — позволяет возвращать синтаксис `..` для определения родительских каталогов:

```
var file1:File = File.documentsDirectory;
file1 = file1.resolvePath("AIR Test");
var file2:File = File.documentsDirectory;
file2 = file2.resolvePath("AIR Test/bob/test.txt");
var file3:File = File.documentsDirectory;
file3 = file3.resolvePath("AIR Test/susan/test.txt");

trace(file2.getRelativePath(file1, true)); // ../..
trace(file3.getRelativePath(file2, true)); // ../../bob/test.txt
```

## Получение канонических версий имен файлов

**Adobe AIR 1.0 и более поздних версий**

Имена файлов и пути к ним в Windows и Mac OS не учитывают регистр. В примере ниже два объекта File указывают на один и тот же файл:

```
File.documentsDirectory.resolvePath("test.txt");
File.documentsDirectory.resolvePath("TeSt.TxT");
```

Однако в именах документов и каталогов используются заглавные буквы. Так, в примерах ниже мы исходим из того, что в каталоге документов существует некая папка с названием AIR Test:

```
var file:File = File.documentsDirectory.resolvePath("AIR test");
trace(file.nativePath); // ... AIR test
file.canonicalize();
trace(file.nativePath); // ... AIR Test
```

Метод `canonicalize()` преобразует объект `nativePath` так, чтобы в имени файла или каталога использовался правильный регистр. В файловых системах, чувствительных к регистру (таких как Linux), если существует несколько файлов, имена которых различаются только регистрами, метод `canonicalize()` настраивает путь для первого найденного файла (в порядке, определяемом файловой системой).

Метод `canonicalize()` также можно использовать для преобразования коротких имен файлов (имен «8.3») в длинные имена в Windows, как показано ниже:

```
var path:File = new File();
path.nativePath = "C:\\AIR~1";
path.canonicalize();
trace(path.nativePath); // C:\AIR Test
```

## Работа с пакетами и символьными ссылками

**Adobe AIR 1.0 и более поздних версий**

Разные операционные системы поддерживают пакетные файлы и файлы символьных ссылок:

**Пакеты**. В Mac OS каталог может быть назначен пакетом и в этом случае будет отображаться в файловом менеджере Mac OS Finder как единый файл, а не как каталог.

**Символьные ссылки**. Символьные ссылки поддерживаются в Mac OS, Linux и Windows Vista. С помощью символьных ссылок один файл может ссылаться на другой файл или каталог на диске. Не путайте символьные ссылки с псевдонимами. Псевдонимы всегда считаются файлами (не каталогами), а чтение или запись псевдонимов и ярлыков никаким образом не изменяет файлы, на которые они указывают. Символьная ссылка, в свою очередь, повторяет поведение файла или каталога, на который она указывает. Она может считаться файлом или каталогом, а при чтении и записи символьной ссылки изменения будут производиться над самим файлом, на который она указывает, а не над ссылкой. Кроме того, в Windows для свойства `isSymbolicLink` объекта File, ссылающегося на узловой пункт (используется в файловой системе NTFS), устанавливается значение `true`.

Класс File включает свойства `isPackage` и `isSymbolicLink`, позволяющие проверить, ссылается ли объект File на пакет или символьную ссылку.

В приведенном ниже коде каталог рабочего стола просматривается и создается список подкаталогов, *не являющихся* пакетами:

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (desktopNodes[i].isDirectory && !desktopNodes[i].isPackage)
    {
        trace(desktopNodes[i].name);
    }
}
```

В приведенном ниже коде просматривается каталог рабочего стола и создается список файлов и каталогов, *не являющихся* символьными ссылками:

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (!desktopNodes[i].isSymbolicLink)
    {
        trace(desktopNodes[i].name);
    }
}
```

Метод `canonicalize()` изменяет путь к символьной ссылке, так чтобы он указывал на файл или каталог, на который она ссылается. В приведенном ниже коде просматривается каталог рабочего стола пользователя и выводятся пути, на которые ссылаются файлы, являющиеся символьными ссылками:

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (desktopNodes[i].isSymbolicLink)
    {
        var linkNode:File = desktopNodes[i] as File;
        linkNode.canonicalize();
        trace(linkNode.nativePath);
    }
}
```

## Определение свободного места на томе

**Adobe AIR 1.0 и более поздних версий**

Свойство `spaceAvailable` объекта File указывает на количество свободного места в расположении объекта File в байтах. Например, в приведенном ниже коде проверяется свободное место в каталоге хранилища приложения.

```
trace(File.applicationStorageDirectory.spaceAvailable);
```

Если объект File ссылается на каталог, свойство `spaceAvailable` указывает объем свободного места, доступного для использования файлом. Если объект File ссылается на файл, свойство `spaceAvailable` обозначает объем, который потенциально может занять файл. Если расположения файла не существует, свойство `spaceAvailable` принимает значение 0. Если объект File ссылается на символьную ссылку, свойство `spaceAvailable` означает свободный объем в том месте, на которое указывает символьная ссылка.

Как правило, объем, который потенциально может занять файл, равен свободному объему всего тома, на котором содержится файл или каталог. Тем не менее при подсчете свободного места могут учитываться квоты и ограничения на каждый каталог.

При добавлении к тому файла или каталога, как правило, требуется чуть больше места, чем сам размер файла или каталога. Например, операционной системе может потребоваться больше места для хранения индексов. Либо секторы дисков могут задействовать дополнительное место. Кроме того, объем свободного места изменяется динамически. Поэтому не следует ожидать, что вы сможете использовать все заявленное свободное место. Сведения о записи в файловую систему см. в разделе ««Чтение и запись файлов» на странице 741».

Метод `StorageVolumeInfo.getStorageVolumes()` предоставляет дополнительные сведения о подключенных томах запоминающих устройств (см. раздел ««Работа с томами запоминающих устройств» на странице 739»).

## Открытие файлов с помощью выбранной по умолчанию системной программы

**Adobe AIR 2 и более поздних версий**

В AIR 2 можно открыть файл с помощью программы, зарегистрированной в операционной системе для его открытия. Например, приложение AIR может открыть DOC-файл с помощью программы, которая зарегистрирована для его открытия. Используйте метод `openWithDefaultApplication()` объекта File для открытия файла. Например, следующий код открывает файл test.doc на рабочем столе пользователя с помощью программы, выбранной по умолчанию для файлов DOC:

```
var file:File = File.deskopDirectory;
file = file.resolvePath("test.doc");
file.openWithDefaultApplication();
```

***Примечание.*** *В ОС Linux тип файла MIME (а не расширение) определяет приложение по умолчанию для файла.*

Следующий код дает пользователю возможность перейти к mp3-файлу и открыть его в приложении, которое выбрано по умолчанию для воспроизведения mp3-файлов:

```
var file:File = File.documentsDirectory;
var mp3Filter:FileFilter = new FileFilter("MP3 Files", "*.mp3");
file.browseForOpen("Open", [mp3Filter]);
file.addEventListener(Event.SELECT, fileSelected);

function fileSelected(e:Event):void
{
    file.openWithDefaultApplication();
}
```

Метод `openWithDefaultApplication()` нельзя использовать по отношению к файлам, расположенным в каталоге приложения.

AIR не позволяет применять метод `openWithDefaultApplication()` для открытия ряда файлов. В Windows AIR запрещает открывать файлы определенных типов (например, EXE или BAT). В ОС Mac OS и Linux среда AIR предотвращает открытие файлов в определенном приложении. (Включая приложения «Терминал» и AppletLauncher в ОС Mac OS и csh, bash или ruby в ОС Linux.) Попытка открыть один из этих файлов с помощью метода `openWithDefaultApplication()` приводит к появлению исключения. Полный список запрещаемых типов файлов см. в записи справочника по языку для метода `File.openWithDefaultApplication()`.

***Примечание.*** *Это ограничение не существует для приложения AIR, установленного с помощью собственного установщика (расширенное приложение рабочего стола).*

## Получение информации о файловой системе

**Adobe AIR 1.0 и более поздних версий**

Класс File включает следующие статические свойства, содержащие полезную информацию о файловой системе:

| Свойство | Описание |
|---|---|
| `File.lineEnding` | Последовательность символов закрытия строки, используемая домашней операционной системой. В Mac OS и Linux это символ новой строки. В Windows это символ перевода каретки, за которым следует символ новой строки. |
| `File.separator` | Разделительный символ, используемый в домашней операционной системе. В Mac OS и Linux это косая черта (/). В ОС Windows это обратная косая черта (\). |
| `File.systemCharset` | Кодировка файлов по умолчанию, используемая домашней операционной системой. Это относится к набору символов операционной системы, соответствующему ее языку. |

Класс `Capabilities` также включает полезную информацию о системе, которая может пригодиться при работе с файлами.

| Свойство | Описание |
|---|---|
| Capabilities.hasIME | Показывает, установлен ли редактор методов ввода (IME) в системе, где работает проигрыватель: да (`true`) или нет (`false`). |
| Capabilities.language | Указывает код языка системы, в которой работает проигрыватель. |
| Capabilities.os | Указывает текущую операционную систему. |

***Примечание.*** *Будьте внимательны, используя* `Capabilities.os` *для определения характеристик системы. Если существует более определенное свойство для определения характеристик системы, воспользуйтесь им. В противном случае вы рискуете написать код, который работает корректно не на всех платформах. Например, рассмотрим следующий код:*

```
var separator:String;
if (Capablities.os.indexOf("Mac") > -1)
{
    separator = "/";
}
else
{
    separator = "\\";
}
```

У этого кода возникают проблемы при работе в ОС Linux. Лучше просто использовать свойство `File.separator`.

## Работа с каталогами

**Adobe AIR 1.0 и более поздних версий**

Среда выполнения предоставляет возможности работы с каталогами в локальной файловой системе.

Дополнительные сведения о создании объектов File, указывающих на каталоги, см. в разделе «Объект File, указывающий на каталог» на странице 721.

### Создание каталогов

**Adobe AIR 1.0 и более поздних версий**

Метод `File.createDirectory()` позволяет создать каталог. Например, в приведенном ниже коде создается каталог под названием «AIR Test», являющийся подкаталогом начального каталога пользователя:

```
var dir:File = File.userDirectory.resolvePath("AIR Test");
dir.createDirectory();
```

Если каталог существует, метод `createDirectory()` не выполняет никаких действий.

В некоторых моделях объект FileStream создает каталоги при открытии файлов. Отсутствующие каталоги создаются при создании экземпляра FileStream со значением параметра `fileMode` конструктора `FileStream()`, равным `FileMode.APPEND` или `FileMode.WRITE`. Дополнительные сведения см. в разделе ««Последовательность действий при чтении и записи файлов» на странице 741».

### Создание временного каталога

**Adobe AIR 1.0 и более поздних версий**

Класс File включает метод `createTempDirectory()`, который создает во временном каталоге System еще один каталог, как показано ниже:

```
var temp:File = File.createTempDirectory();
```

Метод `createTempDirectory()` автоматически создает уникальный временный каталог, что избавляет вас от необходимости каждый раз задавать уникальное расположение.

Временный каталог может использоваться для хранения файлов, используемых в ходе одного сеанса приложения. Обратите внимание, что существует метод `createTempFile()`, позволяющий создавать новые уникальные временные файлы во временном каталоге System.

Перед закрытием приложения рекомендуется удалять временные каталоги, так как они *не* удаляются автоматически на всех устройствах.

## Нумерация каталогов

**Adobe AIR 1.0 и более поздних версий**

С помощью метода `getDirectoryListing()` или `getDirectoryListingAsync()` объекта File можно получить массив объектов File, указывающих на файлы и подкаталоги каталога.

Например, в приведенном ниже коде перечислено содержимое каталога документов пользователя (подкаталоги при этом не исследуются):

```
var directory:File = File.documentsDirectory;
var contents:Array = directory.getDirectoryListing();
for (var i:uint = 0; i < contents.length; i++)
{
    trace(contents[i].name, contents[i].size);
}
```

При использовании асинхронного варианта метода объект события `directoryListing` получает свойство `files`, представляющее собой массив объектов File, относящихся к каталогам:

```
var directory:File = File.documentsDirectory;
directory.getDirectoryListingAsync();
directory.addEventListener(FileListEvent.DIRECTORY_LISTING, dirListHandler);

function dirListHandler(event:FileListEvent):void
{
    var contents:Array = event.files;
    for (var i:uint = 0; i < contents.length; i++)
    {
        trace(contents[i].name, contents[i].size);
    }
}
```

## Копирование и перемещение каталогов

**Adobe AIR 1.0 и более поздних версий**

Каталоги можно копировать и перемещать с помощью тех же методов, которые используются для копирования и перемещения файлов. Например, в приведенном ниже коде каталоги копируются синхронно:

```
var sourceDir:File = File.documentsDirectory.resolvePath("AIR Test");
var resultDir:File = File.documentsDirectory.resolvePath("AIR Test Copy");
sourceDir.copyTo(resultDir);
```

Если задать для параметра `overwrite` метода `copyTo()` значение true, все файлы и папки в существующем целевом каталоге будут удалены и замещены файлами и папками исходного каталога (даже если в исходном каталоге нет целевого файла).

Каталог, указанный в качестве параметра `newLocation` метода `copyTo()` задает путь к итоговому каталогу. Он *не задаетродительский* каталог, в котором будет содержаться итоговый.

Дополнительные сведения см. в разделе «Копирование и перемещение файлов» на странице 737.

### Удаление содержимого каталога

**Adobe AIR 1.0 и более поздних версий**

Класс File содержит методы `deleteDirectory()` и `deleteDirectoryAsync()`. Эти методы служат для удаления каталогов. Первый работает синхронно, второй — асинхронно (см. раздел ««Основные сведения о файлах AIR» на странице 717»). Оба метода содержат параметр `deleteDirectoryContents` (который принимает булево значение). Когда этот параметр равен `true` (его значение по умолчанию — `false`), то при вызове метода удаляются и непустые каталоги, в остальных случаях удаляются только пустые каталоги.

Например, в приведенном ниже коде подкаталог AIR Test синхронно удаляется из каталога документов пользователя:

```
var directory:File = File.documentsDirectory.resolvePath("AIR Test");
directory.deleteDirectory(true);
```

В приведенном ниже коде подкаталог AIR Test асинхронно удаляется из каталога документов пользователя:

```
var directory:File = File.documentsDirectory.resolvePath("AIR Test");
directory.addEventListener(Event.COMPLETE, completeHandler)
directory.deleteDirectoryAsync(true);

function completeHandler(event:Event):void {
    trace("Deleted.")
}
```

Также присутствуют методы `moveToTrash()` и `moveToTrashAsync()`, с помощью которых каталог можно поместить в корзину. Дополнительные сведения см. в разделе ««Перемещение файла в корзину» на странице 739».

## Работа с файлами

**Adobe AIR 1.0 и более поздних версий**

С помощью API-интерфейса файла AIR вы сможете добавить в свое приложение основные интерактивные возможности. Например, будет возможно читать и записывать файлы, копировать и удалять их, и многое другое. Так как у ваших приложений есть доступ к локальной файловой системе, ознакомьтесь с разделом «Безопасность AIR» на странице 1160, если вы еще этого не сделали.

***Примечание.*** *Тип файла можно связывать с приложением AIR (чтобы при двойном щелчке по файлу он открывался в этом приложении). Дополнительные сведения см. в разделе «Управление сопоставлением файлов» на странице 952.*

### Получение информации о файле

**Adobe AIR 1.0 и более поздних версий**

Класс File включает следующие свойства, содержащие информацию о файле или каталоге, на который указывает объект File:

| Свойство File | Описание |
|---|---|
| creationDate | Дата создания файла на локальном диске. |
| creator | Сейчас не используется. Пользуйтесь свойством `extension`. (Это свойство выводит тип creator для файлов Macintosh и используется только в версиях Mac OS, предшествующих Mac OS X.) |
| downloaded | (AIR 2 и более поздних версий) Указывает, был ли файл или каталог, указанный в ссылке, загружен из Интернета. свойство имеет значение только для операционных систем, в которых файлы могут помечаться во время загрузки:<br>• ОС Windows XP с пакетом обновления SP2 или более поздним и ОС Windows Vista<br>• ОС Mac OS 10.5 и более поздних версий |
| exists | Определяет, существует ли файл или каталог, на который ведет ссылка. |
| extension | Расширение файла, являющееся частью его имени. Расширение следует за точкой («.») в имени файла (но не включает ее). Если в имени файла нет точки, то значение extension равно `null`. |
| icon | Объект Icon, содержащий значки, заданные для этого файла. |
| isDirectory | Указывает, ссылается ли объект File на каталог. |
| modificationDate | Дата последнего изменения файла или каталога на локальном диске. |
| name | Имя файла или каталога (включая расширение файла, если имеется) на локальном диске. |
| nativePath | Полный путь в представлении домашней операционной системы. См. раздел ««Пути к объектам File» на странице 718». |
| parent | Папка, содержащая папку или файл, представленный объектом File. Это свойство равно `null`, если объект File ссылается на файл или каталог на корневом уровне файловой системы. |
| size | Размер файла на локальном диске в байтах. |
| type | Сейчас не используется. Пользуйтесь свойством `extension`. (На компьютерах Macintosh это свойство представляет четырехсимвольный тип файла, используемый только в версиях Mac OS, предшествующих Mac OS X.) |
| url | URL-адрес файла или каталога. См. раздел ««Пути к объектам File» на странице 718». |

Дополнительные сведения об этих свойствах см. в описании класса File в справочнике ActionScript® 3.0 для платформы ® Flash® Platform.

## Копирование и перемещение файлов

**Adobe AIR 1.0 и более поздних версий**

Класс File содержит два метода для копирования файлов и каталогов: `copyTo()` и `copyToAsync()`. Класс File содержит два метода для перемещения файлов и каталогов: `moveTo()` и `moveToAsync()`. Методы `copyTo()` и `moveTo()` являются синхронными, а методы `copyToAsync()` и `moveToAsync()` — асинхронными (см. раздел ««Основные сведения о файлах AIR» на странице 717»).

Для копирования файла необходимо создать два объекта File. Один из них указывает на файл, который следует скопировать или переместить, и вызывает соответствующий метод, а второй указывает целевой путь (путь назначения).

В примере ниже файл test.txt копируется из подкаталога AIR Test каталога документов пользователя в файл copy.txt в том же каталоге:

```
var original:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var newFile:File = File.resolvePath("AIR Test/copy.txt");
original.copyTo(newFile, true);
```

В этом примере значение параметра `overwrite` метода `copyTo()` (второго параметра) равно `true`. При задании для `overwrite` значения `true` существующий целевой файл перезаписывается. Этот параметр не обязателен. При значении `false` (значение по умолчанию) операция отправляет событие IOErrorEvent, если целевой файл существует (при этом файл не копируется).

Варианты методов копирования и перемещения, имеющие в названии элемент «Async», работают асинхронно. Метод `addEventListener()` позволяет отслеживать выполнение задачи или условия ошибки, как показано ниже:

```
var original = File.documentsDirectory;
original = original.resolvePath("AIR Test/test.txt");

var destination:File = File.documentsDirectory;
destination =  destination.resolvePath("AIR Test 2/copy.txt");

original.addEventListener(Event.COMPLETE, fileMoveCompleteHandler);
original.addEventListener(IOErrorEvent.IO_ERROR, fileMoveIOErrorEventHandler);
original.moveToAsync(destination);

function fileMoveCompleteHandler(event:Event):void {
    trace(event.target); // [object File]
}
function fileMoveIOErrorEventHandler(event:IOErrorEvent):void {
    trace("I/O Error.");
}
```

Класс File также включает методы `File.moveToTrash()` и `File.moveToTrashAsync()`, служащие для перемещения файла или каталога в корзину.

## Удаление файла

**Adobe AIR 1.0 и более поздних версий**

Класс File содержит методы `deleteFile()` и `deleteFileAsync()`. Эти методы служат для удаления файлов. Первый работает синхронно, второй — асинхронно (см. раздел ««Основные сведения о файлах AIR» на странице 717»).

Например, в приведенном ниже коде файл test.txt синхронно удаляется из каталога документов пользователя:

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.deleteFile();
```

В этом коде файл test.txt асинхронно удаляется из каталога документов пользователя:

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.addEventListener(Event.COMPLETE, completeHandler)
file.deleteFileAsync();

function completeHandler(event:Event):void {
    trace("Deleted.")
}
```

Также существуют методы `moveToTrash()` и `moveToTrashAsync`, с помощью которых файл или каталог можно переместить в корзину. Дополнительные сведения см. в разделе ««Перемещение файла в корзину» на странице 739».

### Перемещение файла в корзину

**Adobe AIR 1.0 и более поздних версий**

Класс File содержит методы `moveToTrash()` и `moveToTrashAsync()`. Эти методы отправляют файл или каталог в корзину. Первый из них работает синхронно, а второй — асинхронно (см. раздел ««Основные сведения о файлах AIR» на странице 717»).

Например, в следующем коде файл test.txt синхронно перемещается из каталога документов пользователя в корзину:

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.moveToTrash();
```

***Примечание.*** *В операционных системах, которые не поддерживают концепцию восстанавливаемой папки корзины, файлы сразу же удаляются.*

### Создание временного файла

**Adobe AIR 1.0 и более поздних версий**

Класс File включает метод `createTempFile()`, который создает во временном системном каталоге еще один файл, как показано ниже:

```
var temp:File = File.createTempFile();
```

Метод `createTempFile()` автоматически создает уникальный временный файл, что избавляет вас от необходимости каждый раз задавать уникальное расположение.

Временный файл может использоваться для временного хранения данных, используемых в ходе одного сеанса приложения. Обратите внимание, что существует метод `createTempDirectory()`, позволяющий создавать новые уникальные временные каталоги во временном каталоге System.

Перед закрытием приложения рекомендуется удалять временные файлы, так как они *не* удаляются автоматически на всех устройствах.

## Работа с томами запоминающих устройств

**Adobe AIR 2 и более поздних версий**

В AIR 2 можно обнаружить момент подключения и отключения томов запоминающего устройства. Класс StorageVolumeInfo определяет единичный объект `storageVolumeInfo`. Объект `StorageVolumeInfo.storageVolumeInfo` отправляет событие `storageVolumeMount` при подключении тома запоминающего устройства. А когда том отключается, он отправляет событие `storageVolumeUnmount`. Класс StorageVolumeChangeEvent определяет эти события.

***Примечание.*** *В современных версиях ОС Linux объект StorageVolumeInfo отправляет только события `storageVolumeMount` и `storageVolumeUnmount` для физических устройств и сетевых дисков, установленных в определенных местоположениях.*

Свойство `storageVolume` класса StorageVolumeChangeEvent является объектом StorageVolume. Класс StorageVolume определяет основные свойства тома запоминающего устройства:

- `drive` — буква тома жесткого диска в ОС Windows (`null` в остальных операционных системах)
- `fileSystemType` — тип файловой системы в томе запоминающего устройства (например, «FAT», «NTFS», «HFS» или «UFS»);
- `isRemoveable` — является ли том съемным (`true`) или нет (`false`);
- `isWritable` — является ли том доступным для записи (`true`) или нет (`false`);
- `name` — имя тома;
- `rootDirectory` — объект File, соответствующий корневому каталогу тома;

Класс StorageVolumeChangeEvent также включает свойство `rootDirectory`. Свойство `rootDirectory` представляет собой объект File, ссылающийся на корневой каталог тома запоминающего устройства, которое было подключено или отключено.

Свойство `storageVolume` объекта StorageVolumeChangeEvent не определено (`null`) для отключенного тома. Однако вы можете открыть свойство `rootDirectory` этого события.

Следующий код выводит имя и путь к файлу в томе запоминающего устройства, когда оно подключается:

```
StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME
_MOUNT, onVolumeMount);
function onVolumeMount(event:StorageVolumeChangeEvent):void
{
    trace(event.storageVolume.name, event.rootDirectory.nativePath);
}
```

Следующий код выводит путь к файлу в томе запоминающего устройства, когда оно отключается:

```
StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME
_UNMOUNT, onVolumeUnmount);
function onVolumeUnmount(event:StorageVolumeChangeEvent):void
{
    trace(event.rootDirectory.nativePath);
}
```

Объект `StorageVolumeInfo.storageVolumeInfo` включает метод `getStorageVolumes()`. Этот метод возвращает вектор объектов StorageVolume, относящийся к подключенным в данный момент томам запоминающего устройства. В следующем коде показано, как составляется список корневых каталогов всех подключенных томов запоминающего устройства:

```
var volumes:Vector.<StorageVolume> = new Vector.<StorageVolume>;
volumes = StorageVolumeInfo.storageVolumeInfo.getStorageVolumes();
for (var i:int = 0; i < volumes.length; i++)
{
    trace(volumes[i].name, volumes[i].rootDirectory.nativePath);
}
```

***Примечание.*** *В современных дистрибутивах Linux метод `getStorageVolumes()` возвращает объекты, соответствующие физическим устройствам или сетевым дискам, подключенным в определенных местоположениях.*

Метод `File.getRootDirectories()` выводит список корневых каталогов (см. раздел «Указание на корневой каталог файловой системы» на странице 724). Однако объекты StorageVolume (перечисленные методом `StorageVolumeInfo.getStorageVolumes()`) предоставляют дополнительную информацию о томах запоминающего устройства.

Свойство `spaceAvailable` свойства `rootDirectory` для объекта StorageVolume можно применять, чтобы получить сведения о свободном месте на томе запоминающего устройства. См. раздел «Определение свободного места на томе» на странице 732.

### Дополнительные разделы справки

StorageVolume

StorageVolumeInfo

## Чтение и запись файлов

**Adobe AIR 1.0 и более поздних версий**

С помощью класса FileStream приложения AIR могут читать и записывать данные в файловую систему.

### Последовательность действий при чтении и записи файлов

**Adobe AIR 1.0 и более поздних версий**

Последовательность действий при записи и чтении файлов показана ниже.

**Инициализация объекта File, задающего путь.**

Объект File представляет собой путь к файлу, с которым планируется работать (или к файлу, который будет создан позже).

```
var file:File = File.documentsDirectory;
file = file.resolvePath("AIR Test/testFile.txt");
```

В примере используется свойство `File.documentsDirectory` и метод `resolvePath()` объекта File для инициализации объекта File. Тем не менее есть множество других способов указания объекта File на файл. Дополнительные сведения см. в разделе ««Объект File, указывающий на файл» на странице 725».

**Инициализируйте объект FileStream.**

**Вызовите метод open() или openAsync() объекта FileStream.**

Вызываемый метод зависит от того, требуется ли открывать файл для синхронных или асинхронных операций. Объект File используется в качестве параметра `file` метода open. Для параметра `fileMode` необходимо задать константу из класса FileMode, указывающую, как будет использоваться файл.

Например, в приведенном ниже коде инициализируется объект FileStream, используемый для создания файла и перезаписи существующих данных:

```
var fileStream:FileStream = new FileStream();
fileStream.open(file, FileMode.WRITE);
```

Дополнительные сведения см. в разделах ««Инициализация объекта FileStream, открытие и закрытие файлов» на странице 743» и ««Методы открытия объекта FileStream» на странице 742».

**Если файл открыт асинхронно (методом openAsync()), добавьте и настройте прослушиватели событий для объекта FileStream.**

Эти методы прослушивателя событий реагируют на события, отправляемые объектом FileStream в различных ситуациях, включая считывание данных из файла, возникновение ошибок ввода/вывода, завершение записи данных в полном объеме.

Дополнительные сведения см. в разделе «Асинхронное программирование и события, генерируемые асинхронно открытым объектом FileStream» на странице 747.

**При необходимости добавьте код для чтения и записи данных.**

Класс FileStream предлагает целый ряд методов чтения и записи. (Название каждого из них начинается со слова «read» или «write».) Метод, выбранный для чтения или записи данных, зависит от формата данных в целевом файле.

Например, если данные имеют кодировку UTF, можно использовать методы `readUTFBytes()` и `writeUTFBytes()`. Если данные обрабатываются как байтовые массивы, можно использовать методы `readByte()`, `readBytes()`, `writeByte()` и `writeBytes()`. Дополнительные сведения см. в разделе «Форматы данных и выбор методов чтения и записи» на странице 748.

Если файл открыт асинхронно, то перед вызовом метода чтения убедитесь, что доступно достаточно данных. Дополнительные сведения см. в разделе ««Буфер чтения и свойство bytesAvailable объекта FileStream» на странице 746».

Если перед записью в файл требуется проверить объем свободного места на диске, это можно сделать с помощью свойства spaceAvailable объекта File. Дополнительные сведения см. в разделе ««Определение свободного места на томе» на странице 732».

**После завершения работы с файлом вызовите метод close() объекта FileStream.**

Вызов метода close() делает файл доступным для других приложений.

Дополнительные сведения см. в разделе «Инициализация объекта FileStream, открытие и закрытие файлов» на странице 743.

Образец приложения, в котором для чтения и записи используется класс FileStream, рассматривается в следующих статьях в Центре разработки Adobe AIR:

- Создание текстового редактора
- Создание текстового редактора
- Чтение и запись в XML-файл установок

## Работа с объектами FileStream

**Adobe AIR 1.0 и более поздних версий**

Класс FileStream определяет методы для открытия, чтения и записи файлов.

### Методы открытия объекта FileStream

**Adobe AIR 1.0 и более поздних версий**

Методы `open()` и `openAsync()` объекта FileStream содержат параметр `fileMode`, определяющий некоторые свойства файлового потока:

- возможность чтения из файла

- возможность записи в файл
- положение добавления данных при записи — в конец файла
- действия в ситуации, когда файл (и его родительские каталоги) не существует

Ниже представлены различные режимы файлов, которые можно задать с помощью параметра `fileMode` методов `open()` и `openAsync()`:

| Режим файла | Описание |
|---|---|
| FileMode.READ | Указывает, что открытый файл предназначен только для чтения. |
| FileMode.WRITE | Указывает, что открытый файл предназначен для записи. Если файл не существует, он создается при открытии объекта FileStream. Если файл существует, все имеющиеся данные удаляются. |
| FileMode.APPEND | Указывает, что открытый файл предназначен для добавления данных. Если файл не существует, он будет создан. Если файл существует, данные не переписываются, а новые данные записываются в конец файла. |
| FileMode.UPDATE | Указывает, что открытый файл предназначен для чтения и записи. Если файл не существует, он будет создан. Этот режим подходит для случайного доступа к файлу для чтения и записи. Чтение может осуществляться из любого положения в файле. При записи перезаписываются только те байты, поверх которых записываются новые данные, остальные байты остаются неизменными. |

### Инициализация объекта FileStream, открытие и закрытие файлов

**Adobe AIR 1.0 и более поздних версий**

При открытии объекта FileStream он становится доступен для чтения и записи данных в файл. Объект FileStream открывается путем передачи метода `open()` или `openAsync()` объекта FileStream:

```
 var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.READ);
```

Параметр `fileMode` (второй параметр методов `open()` и `openAsync()`) указывает, в каком режиме нужно открыть файл: для чтения, записи, добавления или обновления данных. Дополнительные сведения см. в разделе «Методы открытия объекта FileStream» на странице 742.

Если при открытии файлов для асинхронных операций используется метод `openAsync()`, настройте прослушиватели для обработки асинхронных событий:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completeHandler);
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.addEventListener(IOErrorEvent.IO_Error, errorHandler);
myFileStream.openAsync(myFile, FileMode.READ);

function completeHandler(event:Event):void {
    // ...
}

function progressHandler(event:ProgressEvent):void {
    // ...
}

function errorHandler(event:IOErrorEvent):void {
    // ...
}
```

Файл открывается для синхронной или асинхронной операции в зависимости от того, какой используется метод: `open()` или `openAsync()`. Дополнительные сведения см. в разделе ««Основные сведения о файлах AIR» на странице 717».

Если в режиме `FileMode.READ` или `FileMode.UPDATE` открывающему методу объекта FileStream задать параметр `fileMode`, данные считываются в буфер чтения, как только открывается объект FileStream. Дополнительные сведения см. в разделе ««Буфер чтения и свойство bytesAvailable объекта FileStream» на странице 746».

Метод `close()` объекта FileStream можно вызвать для закрытия связанного с приложением файла, в результате чего он будет доступен для использования другими приложениями.

### Свойство position объекта FileStream

**Adobe AIR 1.0 и более поздних версий**

Свойство `position` объекта FileStream определяет, откуда считываются и куда записываются данные при вызове следующего метода чтения или записи.

Перед операцией чтения или записи свойство `position` необходимо задать любому действительному месту в файле.

Например, в приведенном ниже коде строка `"hello"` (в кодировке UTF) находится в файле в положении 8:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.UPDATE);
myFileStream.position = 8;
myFileStream.writeUTFBytes("hello");
```

При первом открытии объекта FileStream свойство `position` имеет значение 0.

Перед операцией чтения значение `position` должно быть не менее 0 и не более числа байтов в файле (они определяют имеющиеся позиции в файле).

Значение свойства `position` изменяется только при следующих условиях:

- если свойство `position` задается явно.

- если вызывается метод чтения.
- если вызывается метод записи.

Если вызывается метод чтения или записи объекта FileStream, к свойству `position` немедленно прибавляется количество записанных или считанных байтов. В зависимости от используемого метода чтения или записи, значение свойства `position` увеличивается на заданное количество считываемых байтов или на количество свободных байтов. При последовательном вызове метода чтения или записи эта операция начинается с новой позиции.

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.UPDATE);
myFileStream.position = 4000;
trace(myFileStream.position); // 4000
myFileStream.writeBytes(myByteArray, 0, 200);
trace(myFileStream.position); // 4200
```

Тем не менее существует и исключение: если объект FileStream открыт в режиме добавления данных, свойство `position` не изменяется после вызова метода чтения или записи. (В режиме добавления данные всегда записываются в конец файла, вне зависимости от значения свойства `position`.)

Если файл открыт для асинхронных операций, операция записи не завершится, пока не будет выполнена следующая строка кода. Однако при вызове нескольких асинхронных методов подряд среда выполнения будет выполнять их по очереди:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.openAsync(myFile, FileMode.WRITE);
myFileStream.writeUTFBytes("hello");
myFileStream.writeUTFBytes("world");
myFileStream.addEventListener(Event.CLOSE, closeHandler);
myFileStream.close();
trace("started.");

closeHandler(event:Event):void
{
    trace("finished.");
}
```

Вот как выглядит пошаговое выполнение этого кода:

```
started.
finished.
```

Значение свойства `position`*может быть задано* сразу после вызова метода чтения или записи (или в любое время), и тогда следующая операция чтения или записи начнется из этого положения. Например, обратите внимание, что в приведенном ниже коде свойство `position` задается сразу после вызова операции `writeBytes()`, а значение свойства `position` остается равным 300 даже после завершения операции записи:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.openAsync(myFile, FileMode.UPDATE);
myFileStream.position = 4000;
trace(myFileStream.position); // 4000
myFileStream.writeBytes(myByteArray, 0, 200);
myFileStream.position = 300;
trace(myFileStream.position); // 300
```

### Буфер чтения и свойство bytesAvailable объекта FileStream

**Adobe AIR 1.0 и более поздних версий**

При открытии объекта FileStream с возможностью чтения (т. е. если параметр `fileMode` метода `open()` или `openAsync()` равен `READ` или `UPDATE`) среда выполнения сохраняет данные во внутреннем буфере. Объект FileStream начинает чтение данных в буфер сразу же после открытия файла (вызывается метод `open()` или `openAsync()` объекта FileStream).

Если файл открыт для синхронных операций (с помощью метода `open()`), указатель `position` можно установить на любое действительное положение (в границах файла) и начать чтение любого объема данных (в границах файла), как показано ниже (здесь мы исходим из того, что в файле не менее 100 байт):

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.READ);
myFileStream.position = 10;
myFileStream.readBytes(myByteArray, 0, 20);
myFileStream.position = 89;
myFileStream.readBytes(myByteArray, 0, 10);
```

Неважно, в каком режиме открыт файл: для синхронных или асинхронных операций, — методы чтения всегда обращаются к «доступным» байтам, представленным свойством `bytesAvalable`. При синхронном чтении все байты файла доступны постоянно. При асинхронном чтении байты становятся доступны, начиная с положения, указанного свойством `position`, в ряде асинхронно заполненных сегментов буфера, о чем сообщают события `progress`.

Если файл открыт для *синхронных* операций, свойство `bytesAvailable` всегда равно числу байтов от значения свойства `position` до конца файла (для чтения доступны все байты в файле).

Если файл открыт для *асинхронных* операций, то, прежде чем вызывать метод чтения, необходимо убедиться, что в буфере чтения достаточно данных. Если файл открыт асинхронно, то по мере выполнения операции чтения данные из файла (начиная со значения `position`, заданного в начале операции чтения) добавляются в буфер, а значение свойства `bytesAvailable` увеличивается по мере считывания байтов. Свойство `bytesAvailable` указывает количество доступных байтов, начиная с байта в положении, заданном свойством `position`, и до конца буфера. Объект FileStream периодически отправляет событие `progress`.

Если файл открыт асинхронно, то, по мере того как данные в буфере чтения становятся доступны, объект FileStream периодически отправляет событие `progress`. Например, в приведенном ниже коде данные считываются в объект ByteArray, `bytes`, по мере того как они считываются в буфер:

```
var bytes:ByteArray = new ByteArray();
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.openAsync(myFile, FileMode.READ);

function progressHandler(event:ProgressEvent):void
{
    myFileStream.readBytes(bytes, myFileStream.position, myFileStream.bytesAvailable);
}
```

Если файл открыт асинхронно, считываться могут только данные в буфере чтения. Более того, по мере чтения данных, они удаляются из буфера чтения. Перед началом операции чтения необходимо удостовериться, что в буфере есть данные. Например, в приведенном ниже коде считываются 8000 байт, начиная с положения 4000:

```
 var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
myFileStream.position = 4000;

var str:String = "";

function progressHandler(event:Event):void
{
    if (myFileStream.bytesAvailable > 8000 )
    {
        str += myFileStream.readMultiByte(8000, "iso-8859-1");
    }
}
```

Во время записи объект FileStream не считывает данные в буфер чтения. После завершения операции записи (когда все данные из буфера записи записаны в файл) объект FileStream открывает новый буфер чтения (если связанный объект FileStream был открыт для чтения) и начинает считывать в него данные, начиная с положения, заданного свойством `position`. Значение свойства `position` может быть последним записанным байтом или другим положением, если пользователь указывает иное значение для объекта `position` после операции записи.

### Асинхронное программирование и события, генерируемые асинхронно открытым объектом FileStream

**Adobe AIR 1.0 и более поздних версий**

Если файл открыт асинхронно (с помощью метода `openAsync()`), то чтение и запись также производятся асинхронно. Во время считывания данных в буфер чтения и записи выходных данных может выполняться другой код ActionScript.

Это означает, что необходимо зарегистрировать прослушиватели для событий, генерируемых асинхронно открытым объектом FileStream.

При регистрации на событие `progress`, когда новые данные будут доступны для чтения, отправляется уведомление, как показано ниже:

```
 var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";

function progressHandler(event:ProgressEvent):void
{
    str += myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

Все данные можно считать, зарегистрировав прослушиватели событий `complete`, как показано ниже:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";
function completeHandler(event:Event):void
{
    str = myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

Примерно таким же образом, каким входные данные помещаются в буфер для асинхронного чтения, данные, записываемые в асинхронном потоке, помещаются в буфер и асинхронно записываются в файл. По мере записи данных в файл объект FileStream периодически отправляет объект `OutputProgressEvent`. Объект `OutputProgressEvent` включает свойство `bytesPending`, значением которого является количество оставшихся байтов для записи. Можно зарегистрироваться на прослушивание события `outputProgress`, и тогда, когда буфер будет записан в файл, будет отправлено уведомление (возможно, как раз вовремя, чтобы отобразить диалоговое окно хода операции). Однако это необязательно. В частности, метод `close()` может вызываться без учета незаписанных байтов. Объект FileStream продолжает запись данных, а событие `close` доставляется после записи в файл последнего байта и закрытия указанного файла.

### Форматы данных и выбор методов чтения и записи

**Adobe AIR 1.0 и более поздних версий**

Каждый файл представляет собой набор байтов на диске. В ActionScript данные любого файла могут быть представлены в форме байтового массива (ByteArray). Например, ниже показано, как данные считываются из файла в объект ByteArray с названием `bytes`:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completeHandler);
myFileStream.openAsync(myFile, FileMode.READ);
var bytes:ByteArray = new ByteArray();

function completeHandler(event:Event):void
{
    myFileStream.readBytes(bytes, 0, myFileStream.bytesAvailable);
}
```

Таким же образом данные записываются из объекта ByteArray с названием `bytes` в файл:

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.WRITE);
myFileStream.writeBytes(bytes, 0, bytes.length);
```

Однако далеко не всегда есть смысл хранить данные в объекте ByteArray. И далеко не всегда данные содержатся в файле указанного формата.

Например, данные в файле могут быть в текстовом формате, и тогда их можно представить в виде строки (объект String).

По этой причине класс FileStream содержит методы чтения и записи данных не только для объектов ByteArray. Например, метод `readMultiByte()` позволяет считывать данные из файла и сохранять их в виде строки, как показано ниже:

```
 var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";

function completeHandler(event:Event):void
{
    str = myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

Второй параметр метода `readMultiByte()` задает формат, который в языке ActionScript используется для интерпретации данных (например, «iso-8859-1»). Adobe AIR поддерживает общепринятые кодировки наборов символов (см. «Поддерживаемые наборы символов»).

Класс FileStream также содержит метод `readUTFBytes()`, считывающий данные из буфера чтения в строку с помощью набора символов UTF-8. Так как символы в кодировке UTF-8 могут иметь разную длину, не используйте метод `readUTFBytes()` для реакции на событие `progress`, потому что данные в конце буфера чтения могут представлять неполный символ. (Это также относится к использованию метода `readMultiByte()` с кодировками символов различной длины.) По этой причине весь набор данных необходимо считать, когда объект FileStream отправляет событие `complete`.

Существуют и другие схожие методы записи: `writeMultiByte()` и `writeUTFBytes()`, — используемые при работе с объектами String и текстовыми файлами.

Методы `readUTF()` и `writeUTF()` (не путать с `readUTFBytes()` и `writeUTFBytes()`) также используются для чтения и записи текстовых данных в файл, но в этом случае предполагается, что текстовым данным предшествуют данные, задающие длину этих текстовых данных — в стандартных текстовых файлах это встречается редко.

Некоторые текстовые файлы в кодировке UTF начинаются с символа «UTF-BOM» (отметка порядка байтов), определяющего порядок следования байтов и формат кодировки (UTF-16 или UTF-32).

Пример чтения и записи в текстовый файл приводится в разделе «Пример: чтение XML-файла в XML-объект» на странице 751.

Методы `readObject()` и `writeObject()` подходят для хранения и извлечения данных из сложных объектов ActionScript. Данные кодируются в формат сообщений AMF (ActionScript Message Format). Для работы с данными в этом формате Adobe AIR, Flash Player, Flash Media Server и Flex Data Services включают в себя соответствующие API.

Существуют и другие методы чтения и записи (например, `readDouble()` и `writeDouble()`). Тем не менее при их использовании проверяйте, чтобы формат файла совпадал с форматом определенных этими методами данных.

Форматы файлов бывают и более сложными, чем обычный текстовый. Например, в MP3-файле содержатся сжатые данные, которые могут интерпретироваться только алгоритмами развертывания и раскодирования MP3-файлов. MP3-файлы также могут содержать теги ID3 с метаданными об этом файле (например, название композиции и имя исполнителя). Существует множество версий формата ID3; самая простая из них (ID3 версии 1) рассматривается в разделе «Пример: чтение и запись данных со случайным доступом» на странице 752.

Структура других файловых форматов (изображений, баз данных, документов приложений и т. д.) отличается, поэтому для того, чтобы работать с ними в ActionScript, необходимо понимать, как они организованы.

## Использование методов load() и save()

**Flash Player 10 и более поздних версий, Adobe AIR 1.5 и более поздних версий**

В проигрывателе Adobe Flash Player 10 к классу FileReference добавлены методы `load()` и `save()`. Эти методы также присутствуют в AIR 1.5, и класс File наследует их из класса FileReference. Эти методы были разработаны для обеспечения пользователей безопасными средствами загрузки и сохранения данных их файлов в проигрыватель Adobe Flash. Но приложения AIR могут также использовать эти методы как простой способ асинхронной загрузки и сохранения файлов.

Например, в следующем коде строка сохраняется в текстовый файл:

```
var file:File = File.applicationStorageDirectory.resolvePath("test.txt");
var str:String = "Hello.";
file.addEventListener(Event.COMPLETE, fileSaved);
file.save(str);
function fileSaved(event:Event):void
{
    trace("Done.");
}
```

Параметр `data` метода `save()` может принимать значения String, XML или ByteArray3. Если значением аргумента является String или XML, метод сохраняет файл как текстовый файл с кодировкой UTF-8.

При выполнении этого демонстрационного кода приложение открывает диалоговое окно, в котором пользователь выбирает местоположение для сохраненных файлов.

В следующем коде загружается строка из текстового файла с кодировкой UTF-8:

```
var file:File = File.applicationStorageDirectory.resolvePath("test.txt");
file.addEventListener(Event.COMPLETE, loaded);
file.load();
var str:String;
function loaded(event:Event):void
{
    var bytes:ByteArray = file.data;
    str = bytes.readUTFBytes(bytes.length);
    trace(str);
}
```

Класс FileStream предоставляет больше функциональных возможностей, чем методы `load()` и `save()`:

- Используя класс FileStream, можно считывать и записывать данные как синхронно, так и асинхронно.
- Использование класса FileStream позволяет выполнить инкрементальную запись в файл.
- Использование класса FileStream позволяет открывать файл для произвольного доступа (как для чтения, так и для записи любого раздела файла).
- Класс FileStream позволяет указывать имеющийся тип доступа к файлу, устанавливая параметр `fileMode` в методе `open()` или `openAsync()`.
- Класс FileStream позволяет сохранять данные в файлы, не предлагая пользователю диалоговые окна открытия или сохранения файлов.
- При чтении данных с помощью класса FileStream можно напрямую использовать типы, отличные от байтовых массивов.

## Пример: чтение XML-файла в XML-объект

**Adobe AIR 1.0 и более поздних версий**

Ниже приводятся примеры чтения и записи в текстовый файл, содержащий XML-данные.

Для чтения инициализируйте объекты File и FileStream, вызовите метод `readUTFBytes()` объекта FileStream и преобразуйте строку в XML-объект:

```
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
var fileStream:FileStream = new FileStream();
fileStream.open(file, FileMode.READ);
var prefsXML:XML = XML(fileStream.readUTFBytes(fileStream.bytesAvailable));
fileStream.close();
```

Точно так же, для записи данных в файл нужно всего лишь создать объекты File и FileStream и вызвать методы записи объекта FileStream. XML-данные в виде строки нужно передать методу записи, как показано ниже:

```
var prefsXML:XML = <prefs><autoSave>true</autoSave></prefs>;
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
fileStream = new FileStream();
fileStream.open(file, FileMode.WRITE);

var outputString:String = '<?xml version="1.0" encoding="utf-8"?>\n';
outputString += prefsXML.toXMLString();

fileStream.writeUTFBytes(outputString);
fileStream.close();
```

В этих примерах используются методы `readUTFBytes()` и `writeUTFBytes()`, так как предполагается, что файлы закодированы в формат UTF-8. В противном случае требуется использовать другой метод (см. раздел «Форматы данных и выбор методов чтения и записи» на странице 748).

В предыдущем примере используются объекты FileStream, открытые для синхронных операций. Файлы также могут быть открыты для асинхронных операций (которые зависят от функций прослушивателей событий и их реакции на события). Например, ниже приведен пример асинхронного чтения из XML-файла:

```
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
var fileStream:FileStream = new FileStream();
fileStream.addEventListener(Event.COMPLETE, processXMLData);
fileStream.openAsync(file, FileMode.READ);
var prefsXML:XML;

function processXMLData(event:Event):void
{
    prefsXML = XML(fileStream.readUTFBytes(fileStream.bytesAvailable));
    fileStream.close();
}
```

Метод `processXMLData()` вызывается после того, как файл полностью считан в буфер чтения, т. е. когда объект FileStream отправляет событие `complete`. Он вызывает метод `readUTFBytes()` для получения версии считываемых данных в виде строки и создает на основании этой строки XML-объект `prefsXML`.

Пример приложения, демонстрирующий эти возможности, см. в разделе «Чтение и запись в XML-файл установок».

## Пример: чтение и запись данных со случайным доступом

**Adobe AIR 1.0 и более поздних версий**

MP3-файлы могут содержать теги ID3 — сегменты в начале или в конце файла, хранящие метаинформацию о записи. Существует несколько версий формата тегов ID3. В этом примере показано, как читать и записывать данные в MP3-файл с тегами ID3 самого простого формата (ID3 версии 1.0), используя функцию случайного доступа к данным файла. Это означает, что чтение и запись производятся в произвольные положения в файле.

В MP3-файле с тегами ID3 версии 1 данные ID3 содержатся в последних 128 байтах файла.

При случайном доступе к файлу для чтения и записи важно указать `FileMode.UPDATE` в качестве параметра `fileMode` метода `open()` или `openAsync()`:

```
var file:File = File.documentsDirectory.resolvePath("My Music/Sample ID3 v1.mp3");
var fileStr:FileStream = new FileStream();
fileStr.open(file, FileMode.UPDATE);
```

Это позволит производить как чтение, так и запись в файл.

Когда файл открыт, с помощью свойства `position` можно указать положение за 128 байт до конца файла:

```
fileStr.position = file.size - 128;
```

В этом коде свойство `position` указывает на это положение в файле, так как формат ID3 версии 1.0 подразумевает, что теги ID3 хранятся в последних 128 байтах файла. Кроме того, в спецификации выдвигается еще ряд требований. Они приведены ниже:

- Первые 3 байта тегов содержат строку `"TAG"`.
- Следующие 30 символов содержат название MP3-дорожки в формате строки.
- Следующие 30 символов содержат имя исполнителя в строковом формате.
- Следующие 30 символов содержат название альбома в строковом формате.
- Следующие 4 символа содержат год выпуска в строковом формате.
- Следующие 30 символов содержат комментарии в строковом формате.
- Следующий байт содержит код, определяющий жанр композиции.
- Все текстовые данные кодируются в формат ISO 8859-1.

Метод `id3TagRead()` проверяет данные после чтения (после события `complete`):

```
function id3TagRead():void
{
    if (fileStr.readMultiByte(3, "iso-8859-1").match(/tag/i))
    {
        var id3Title:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Artist:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Album:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Year:String = fileStr.readMultiByte(4, "iso-8859-1");
        var id3Comment:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3GenreCode:String =  fileStr.readByte().toString(10);
    }
}
```

Точно также можно осуществлять случайный доступ для записи в файл. Например, можно разобрать переменную `id3Title`, чтобы проверить, правильный ли используется регистр (с помощью методов класса String), а затем записать в файл измененную строку с названием `newTitle`, как показано ниже:

```
fileStr.position = file.length - 125;    // 128 - 3
fileStr.writeMultiByte(newTitle, "iso-8859-1");
```

В соответствии со стандартом ID3 версии 1, длина строки `newTitle` должна быть 30 символов, и она должна заканчиваться кодом символа 0 (`String.fromCharCode(0)`).

# Глава 39. Хранение локальных данных

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс SharedObject используется для хранения небольшого объема данных на клиентском компьютере. В Adobe AIR используется также класс EncryptedLocalStore для сохранения на локальном компьютере небольшого объема личных данных пользователя из приложения AIR.

Также в этой файловой системе можно читать и записывать файлы и (в Adobe AIR) иметь доступ к файлам локальной базы данных. Дополнительные сведения см. в разделах ««Работа с файловой системой» на странице 701» и ««Работа с локальными базами данных SQL в AIR» на странице 768».

Существует ряд факторов системы защиты, связанных с общими объектами. Дополнительные сведения см. в разделе ««Общие объекты» на странице 1158» главы ««Безопасность» на странице 1121».

## Общие объекты

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Общий объект, который иногда называется «Flash cookie», представляет собой файл с данными, который могут создавать на компьютере посещаемые сайты. Чаще всего общие объекты используются для улучшения просмотра Интернета. Например, они позволяют настроить внешний вид часто посещаемого веб-сайта.

### Об общих объектах

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Общие объекты функционируют подобно файлам cookie браузера. Класс SharedObject используется для хранения данных на локальном жестком диске пользователя и вызова этих данных во время того же или последующего сеанса. Приложения имеют доступ только к собственным данным SharedObject и только при их запуске на одном и том же домене. Данные не отправляются на сервер и недоступны для приложений, запускаемых на других доменах, однако доступны для приложений того же домена.

#### Сравнение общих объектов с файлами cookie

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

У файлов cookie и общих объектов существует много общего. Так как большинство веб-программистов знакомы с принципом работы файлов cookie, полезным будет сравнить файлы cookie с локальными объектами SharedObjects.

Как правило, файлы cookie, соответствующие стандарту RFC 2109, имеют следующие свойства:

- срок действия, истекающий по умолчанию в конце сеанса;
- возможность деактивирования клиентом на специфическом для сайта основании;
- максимальное общее число файлов cookie, равное 300, но не более 20 файлов для одного сайта;
- размер файла ограничен 4 Кб;

- иногда расцениваются как угроза безопасности, в результате чего возможно их отключение на клиентском компьютере;
- папка сохранения указывается браузером клиента;
- передача от клиента на сервер происходит посредством HTTP.

  В отличие от файлов cookie, общие файлы имеют следующие свойства:

- отсутствие срока действия по умолчанию;
- размер файла по умолчанию ограничен 100 Кб;
- возможность сохранения простых типов данных (например, String, Array и Date).
- папка сохранения указывается приложением (в начальном каталоге пользователя);
- передача от клиента на сервер отсутствует.

### О классе SharedObject

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При помощи класса SharedObject можно создавать и удалять общие объекты, а также определять текущий размер используемого объекта SharedObject.

## Создание общих объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для создания объекта SharedObject используется метод `SharedObject.getLocal()` со следующим синтаксисом:

```
SharedObject.getLocal("objectName" [, pathname]): SharedObject
```

В следующем примере описывается процесс создания общего объекта с именем mySO:

```
public var mySO:SharedObject;
mySO = SharedObject.getLocal("preferences");
```

Таким образом на компьютере клиента создается файл с именем preferences.sol.

Понятие *local* указывает на местоположение общего объекта. В этом случае Adobe® Flash® Player сохраняет файл SharedObject в локальном начальном каталоге клиента.

При создании общего объекта Flash Player создает новый каталог для приложения и домена в изолированной среде. Также создается файл *.sol, хранящий данные SharedObject. Каталогом по умолчанию является подкаталог начального каталога пользователя. В следующей таблице представлены возможное расположение каталога по умолчанию:

| Операционная система | Расположение |
|---|---|
| Windows 95/98/ME/2000/XP | `c:/Documents and Settings/`*`username`*`/Application Data/Macromedia/Flash Player/#SharedObjects` |

| Операционная система | Расположение |
|---|---|
| Windows Vista/Windows 7 | `c:/Users/`*`username`*`/AppData/Roaming/Macromedia/Flash Player/#SharedObjects` |
| Macintosh OS X | `/Users/`*`username`*`/Library/Preferences/Macromedia/Flash Player/#SharedObjects/`*`web_domain/path_to_application/application_name/object_name`*`.sol` |
| Linux/Unix | `/home/username/.macromedia/Flash_Player/#SharedObjects/`*`web_domain/path_to_application/application_name/object_name`*`.sol` |

Под именем каталога #SharedObjects подразумевается каталог с произвольным именем. Далее следует каталог, имя которого идентично имени хоста, далее — путь к приложению и, наконец, файл *.sol.

Например, при запросе приложения с именем MyApp.swf, находящегося на локальном хосте в подкаталоге с именем /sos, Flash Player сохраняет файл *.sol по следующему пути (для Windows XP):

```
c:/Documents and Settings/fred/Application Data/Macromedia/Flash
Player/#SharedObjects/KROKWXRK/#localhost/sos/MyApp.swf/data.sol
```

***Примечание.*** *Если имя в методе `SharedObject.getLocal()`, Flash Player присваивает файлу имя undefined.sol.*

По умолчанию Flash может локально сохранять постоянные объекты SharedObject размером до 100 Кб в каждом домене. Это значение может изменяться пользователем. При попытке сохранения приложением данных в общем объекте, увеличивающим его до размера более 100 Кб, Flash Player отображает диалоговое окно «Локальное сохранение», в котором пользователь может разрешить или запретить использование нескольких локальных хранилищ доменом, запрашивающим доступ.

## Определение пути

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Существует возможность использования необязательного параметра *pathname* для указания местоположения файла SharedObject. Этот файл должен являться подкаталогом папки SharedObject данного домена. Например, при запросе приложения на локальном хосте и вводе следующей строки:

```
mySO = SharedObject.getLocal("myObjectFile","/");
```

Flash Player записывает файл SharedObject в каталог /#localhost (или в каталог /localhost, если приложение является офлайн-приложением). Использование данной возможности наиболее оправдано при необходимости доступа нескольких приложений к одному и тому же общему объекту. В этом случае клиент может запустить два приложения Flex, в каждом из которых указан путь к общему объекту, находящемуся в корне домена; после этого клиент получит доступ из обоих приложений к одному и тому же общему объекту. Для совместного использования данных приложениями, не обеспечивающими неизменности данных, можно использовать объект LocalConnection.

При вводе несуществующего каталога Flash Player не создает файл SharedObject.

## Добавление данных к общему объекту

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В файл *.sol SharedObject можно добавлять данные, используя свойство `data` объекта SharedObject. Для добавления новых данных к общему объекту используйте следующий синтаксис:

```
sharedObject_name.data.variable = value;
```

В следующем примере выполняется добавление свойств `userName`, `itemNumbers` и `adminPrivileges`, а также их значений, к SharedObject:

```
public var currentUserName:String = "Reiner";
public var itemsArray:Array = new Array(101,346,483);
public var currentUserIsAdmin:Boolean = true;
mySO.data.userName = currentUserName;
mySO.data.itemNumbers = itemsArray;
mySO.data.adminPrivileges = currentUserIsAdmin;
```

После присвоения значений свойству `data` Flash Player должен выполнить запись этих значений в файл SharedObject. Для инициирования записи значений в файл SharedObject используйте метод `SharedObject.flush()`, например:

```
mySO.flush();
```

Если пользователь не вызывает метод `SharedObject.flush()`, Flash Player записывает значения в файл при выходе из приложения. Однако такой способ не позволяет пользователю увеличивать доступное пространство, в котором Flash Player хранит данные, если они при записи превышают настройки по умолчанию. Поэтому рекомендуется вызывать метод `SharedObject.flush()`.

При использовании метода `flush()` для записи общих объектов на жесткий диск пользователя следует соблюдать осторожность и проверять, не запретил ли пользователь локальное хранение с помощью менеджера параметров Flash Player Settings Manager (www.macromedia.com/support/documentation/ru/flashplayer/help/settings_manager07.html), как показано в следующем примере.

```
var so:SharedObject = SharedObject.getLocal("test");
trace("Current SharedObject size is " + so.size + " bytes.");
so.flush();
```

### Сохранение объектов в общих объектах

Простые объекты, например, Arrays или Strings, могут сохраняться в свойстве SharedObject `data`.

В следующем примере представлен класс ActionScript, определяющий методы, которые управляют взаимодействием с общим объектом. Эти методы позволяют пользователю добавлять и удалять объекты из общих объектов. В данном классе хранится класс ArrayCollection, содержащий простые объекты.

```
package {
    import mx.collections.ArrayCollection;
    import flash.net.SharedObject;

    public class LSOHandler {

        private var mySO:SharedObject;
        private var ac:ArrayCollection;
        private var lsoType:String;

        // The parameter is "feeds" or "sites".
        public function LSOHandler(s:String) {
            init(s);
        }

        private function init(s:String):void {
            ac = new ArrayCollection();
            lsoType = s;
            mySO = SharedObject.getLocal(lsoType);
            if (getObjects()) {
                ac = getObjects();
            }
        }

        public function getObjects():ArrayCollection {
            return mySO.data[lsoType];
        }

        public function addObject(o:Object):void {
            ac.addItem(o);
            updateSharedObjects();
        }

        private function updateSharedObjects():void {
            mySO.data[lsoType] = ac;
            mySO.flush();
        }
    }

}
```

Следующее приложение Flex создает экземпляр класса ActionScript для всех типов общих объектов, которым требуется экземпляр. После этого приложение вызывает методы данного класса при добавлении или удалении URL-адресов блогов или сайтов.

```
<?xml version="1.0"?>
<!-- lsos/BlogAggregator.mxml -->
<mx:Application
    xmlns:local="*"
    xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="initApp()"
    backgroundColor="#ffffff"
>
    <mx:Script>
        <![CDATA[
        import mx.collections.ArrayCollection;
        import mx.utils.ObjectUtil;
        import flash.net.SharedObject;

        [Bindable]
        public var welcomeMessage:String;

        [Bindable]
        public var localFeeds:ArrayCollection = new ArrayCollection();

        [Bindable]
        public var localSites:ArrayCollection = new ArrayCollection();

        public var lsofeeds:LSOHandler;
        public var lsosites:LSOHandler;

        private function initApp():void {
            lsofeeds = new LSOHandler("feeds");
            lsosites = new LSOHandler("sites");

            if (lsofeeds.getObjects()) {
                localFeeds = lsofeeds.getObjects();
            }
            if (lsosites.getObjects()) {
                localSites = lsosites.getObjects();
            }
        }

        // Adds a new feed to the feeds DataGrid.
        private function addFeed():void {
            // Construct an object you want to store in the
            // LSO. This object can contain any number of fields.
            var o:Object = {name:ti1.text, url:ti2.text, date:new Date()};
            lsofeeds.addObject(o);

            // Because the DataGrid's dataProvider property is
            // bound to the ArrayCollection, Flex updates the
            // DataGrid when you call this method.
            localFeeds = lsofeeds.getObjects();

            // Clear the text fields.
            ti1.text = '';
            ti2.text = '';
        }

        // Removes feeds from the feeds DataGrid.
        private function removeFeed():void {
```

```
            // Use a method of ArrayCollection to remove a feed.
            // Because the DataGrid's dataProvider property is
            // bound to the ArrayCollection, Flex updates the
            // DataGrid when you call this method. You do not need
            // to update it manually.
            if (myFeedsGrid.selectedIndex > -1) {

localFeeds.removeItemAt(myFeedsGrid.selectedIndex);
             }
        }

        private function addSite():void {
            var o:Object = {name:ti3.text, date:new Date()};
            lsosites.addObject(o);
            localSites = lsosites.getObjects();
            ti3.text = '';
        }

        private function removeSite():void {
            if (mySitesGrid.selectedIndex > -1) {

localSites.removeItemAt(mySitesGrid.selectedIndex);
            }
        }

        ]]>
    </mx:Script>

    <mx:Label text="Blog aggregator" fontSize="28"/>

    <mx:Panel title="Blogs">
        <mx:Form id="blogForm">
            <mx:HBox>
                <mx:FormItem label="Name:">
                    <mx:TextInput id="ti1" width="100"/>
                </mx:FormItem>
                <mx:FormItem label="Location:">
                    <mx:TextInput id="ti2" width="300"/>
                </mx:FormItem>
                <mx:Button id="b1" label="Add Feed" click="addFeed()"/>
            </mx:HBox>

            <mx:FormItem label="Existing Feeds:">
                <mx:DataGrid
                    id="myFeedsGrid"
                    dataProvider="{localFeeds}"
                    width="400"
                />
            </mx:FormItem>
            <mx:Button id="b2" label="Remove Feed" click="removeFeed()"/>
        </mx:Form>
    </mx:Panel>

    <mx:Panel title="Sites">
```

```
        <mx:Form id="siteForm">
            <mx:HBox>
                <mx:FormItem label="Site:">
                    <mx:TextInput id="ti3" width="400"/>
                </mx:FormItem>
                <mx:Button id="b3" label="Add Site" click="addSite()"/>
            </mx:HBox>

            <mx:FormItem label="Existing Sites:">
                <mx:DataGrid
                    id="mySitesGrid"
                    dataProvider="{localSites}"
                    width="400"
                />
            </mx:FormItem>
            <mx:Button id="b4" label="Remove Site" click="removeSite()"/>
        </mx:Form>
    </mx:Panel>

</mx:Application>
```

**Сохранение объектов со строгим контролем типов в общих объектах**

Существует возможность сохранения экземпляров ActionScript со строгим контролем типов в общих объектах. Данная возможность осуществляется посредством вызова метода `flash.net.registerClassAlias()` для регистрации класса. При создании экземпляра класса, сохранении его в элементе данных общего объекта и последующем выводе данных пользователь получит экземпляр со строгим контролем типов. По умолчанию свойство SharedObject `objectEncoding` поддерживает кодировку AMF3 и извлекает сохраненный экземпляр из объекта SharedObject. Сохраненный экземпляр сохраняет тот же тип, который был указан при вызове метода `registerClassAlias()`.

## (Только для iOS) Предотвращение резервоного копирования локальных общих объектов в облачное хранилище

**Adobe AIR 3.7 и более поздние версии, только в ОС iOS**

Свойство `SharedObject.preventBackup` позволяет указывать, может ли производиться резервное копирование локальных общих объектов в облачное хранилище iOS. Таковы рекомендации Apple в отношении содержимого, которое можно восстановить или загрузить заново, но которое необходимо для правильной работы приложения в автономном режиме.

## Создание нескольких общих объектов

Имеется возможность создания нескольких общих объектов для одного приложения Flex. Для этого каждому объекту требуется присвоить уникальное имя экземпляра, например:

```
public var mySO:SharedObject = SharedObject.getLocal("preferences");
public var mySO2:SharedObject = SharedObject.getLocal("history");
```

Таким образом в локальном каталоге приложения Flex создаются файлы preferences.sol и history.sol.

## Создание защищенного экземпляра SharedObject

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Когда создается локальный или удаленный экземпляр SharedObject с помощью метода `getLocal()` или `getRemote()`, существует дополнительный параметр с именем `secure`, который определяет, разрешен ли доступ к общему объекту только для SWF-файлов, доставленных через HTTPS-соединение. Если данный параметр имеет значение `true` и SWF-файл доставляется через HTTPS, проигрыватель Flash Player создает новый защищенный общий объект или получает ссылку на существующий защищенный общий объект. Данный защищенный общий объект может считываться или записываться только теми SWF-файлами, доставленными по протоколу HTTPS, которые вызывают метод `SharedObject.getLocal()` с параметром secure, имеющим значение `true`. Если данный параметр имеет значение `false` и SWF-файл передается через HTTPS, проигрыватель Flash Player создает новый общий объект или получает ссылку на существующий общий объект.

Этот общий объект может считываться или записываться SWF-файлами, доставленными не через HTTPS-подключение. Если SWF-файл доставляется не через подключение HTTPS и вы пытаетесь установить для данного параметра значение `true`, то создание нового общего объекта (или доступ к ранее созданному защищенному общему объекту) завершится сбоем, будет выдано сообщение об ошибке, а общий объект получит значение `null`. Если вы попытаетесь выполнить следующий код не через подключение по протоколу HTTPS, метод `SharedObject.getLocal()` выдаст ошибку.

```
try
{
    var so:SharedObject = SharedObject.getLocal("contactManager", null, true);
}
catch (error:Error)
{
    trace("Unable to create SharedObject.");
}
```

Независимо от значения данного параметра созданные общие объекты добавляются к общему объему дискового пространства, доступного для домена.

## Отображение содержимого общего объекта

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В общих объектах значения хранятся в свойстве `data`. Отобразить каждое значение в экземпляре SharedObject можно с помощью цикла `for..in`, как в следующем примере.

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.hello = "world";
so.data.foo = "bar";
so.data.timezone = new Date().timezoneOffset;
for (var i:String in so.data)
{
    trace(i + ":\t" + so.data[i]);
}
```

## Удаление общих объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для уничтожения SharedObject у клиента используйте метод `SharedObject.clear()`. Данный метод не удалит каталоги из пути, установленного по умолчанию для общих объектов приложения.

Пример иллюстрирует удаление файла SharedObject с клиентского компьютера:

```
public function destroySharedObject():void {
    mySO.clear();
}
```

## Пример SharedObject

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В следующем примере демонстрируется возможность хранения простых объектов, таких как объект Date, в объекте SharedObject SharedObject без ручной сериализации и десериализации этих объектов.

Следующий пример начинается с приветствия пользователя как посетителя, прошедшего первую регистрацию. При нажатии кнопки «Выход» приложение сохраняет текущую дату в общем объекте. При следующем запуске этого приложения или обновлении страницы на экран выводится приветствие с уведомлением о времени выхода из приложения.

Для того, чтобы увидеть приложение в действии, запустите приложение, нажмите кнопку «Выход», после чего обновите страницу. Приложение отобразит дату и время последнего нажатия кнопки «Выход». Сохраненную информацию можно удалить в любое время с помощью кнопки «Удалить локальный общий объект».

```
<?xml version="1.0"?>
<!-- lsos/WelcomeMessage.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize="initApp()">
  <mx:Script><![CDATA[
  public var mySO:SharedObject;
  [Bindable]
  public var welcomeMessage:String;

  public function initApp():void {
     mySO = SharedObject.getLocal("mydata");
     if (mySO.data.visitDate==null) {
        welcomeMessage = "Hello first-timer!"
     } else {
        welcomeMessage = "Welcome back. You last visited on " +
           getVisitDate();
     }
  }

  private function getVisitDate():Date {
     return mySO.data.visitDate;
  }

  private function storeDate():void {
     mySO.data.visitDate = new Date();
     mySO.flush();
  }

  private function deleteLSO():void {
     // Deletes the SharedObject from the client machine.
     // Next time they log in, they will be a 'first-timer'.
     mySO.clear();
  }

  ]]></mx:Script>
  <mx:Label id="label1" text="{welcomeMessage}"/>
  <mx:Button label="Log Out" click="storeDate()"/>
  <mx:Button label="Delete LSO" click="deleteLSO()"/>
</mx:Application>
```

# Зашифрованная локальная система хранения данных

Класс EncryptedLocalStore (ELS) представляет собой механизм локального хранения зашифрованных данных, который может использоваться в качестве кэша небольшого размера для хранения закрытых данных приложения. К данным ELS нельзя получить доступ из других приложений. Смысл класса ELS в том, чтобы позволить приложениям хранить легко восстанавливаемые элементы, в частности, имена и пароли учетных записей и другую конфиденциальную информацию. Данные ELS нельзя считать постоянными, как описано ниже в разделах «Ограничения локального зашифрованного хранилища» и «Рекомендации по оптимальному использованию ELS».

***Примечание.*** *В дополнение к зашифрованному локальному хранилищу Adobe AIR также предлагает шифрование содержимого, хранимого в базах данных SQL. Дополнительные сведения см. в разделе ««Использование шифрования с базами данных SQL» на странице 814».*

Зашифрованное локальное хранилище может использоваться для хранения информации, требующей защиты, например учетных данных для входа, используемых веб-службами. Для защиты важной информации от других пользователей можно использовать ELS. Однако в этом случае данные не будут защищены от процессов, выполняемых от имени текущей учетной записи пользователя. Таким образом, этот способ не подходит для защиты важных данных приложения, например DRM или ключей шифрования.

На стационарных компьютерах для установления связи зашифрованного локального хранилища с каждым приложением и пользователем AIR использует интерфейс DPAPI в ОС Windows, KeyChain в ОС Mac OS и iOS, а также KeyRing или KWallet в ОС Linux. Для шифрования локального хранилища используется 128-битный алгоритм шифрования AES-CBC.

В ОС Android данные, сохраняемые классом EncryptedLocalStorage, не шифруются. Вместо этого защита данных обеспечивается на уровне пользователя системой безопасности ОС. ОС Android присваивает каждому приложению индивидуальный идентификатор пользователя. Приложения имеют доступ только к собственным файлам и файлам, созданным в общедоступных папках, таких как съемная карта памяти). Обратите внимание, что на «взломанных» устройствах Android приложения, выполняемые с правами суперпользователя, МОГУТ получать доступ к файлам других приложений. Таким образом, на взломанных устройствах зашифрованное локальное хранилище не обеспечивает такого высокого уровня защиты данных, как на невзломанных.

Информация в зашифрованном хранилище доступна только приложениям AIR в своей изолированной программной среде.

После обновления приложения AIR его обновленная версия не будет иметь доступ к существующим данным в зашифрованном хранилище, только если были выполнены следующие действия.

- Элементы были добавлены с параметром `stronglyBound` в значении `true`
- Существующая и обновленная версии опубликованы до AIR 1.5.3, и обновление имеет подпись миграции.

**Ограничения зашифрованного локального хранилища**

Данные в зашифрованном локальном хранилище защищены параметрами доступа пользователя. Другие лица не могут получить доступ к данным, находящимся в хранилище, если они не могут войти под именем данного пользователя. Однако данные недостаточно хорошо защищены от доступа других приложений, запускаемых прошедшим проверку подлинности пользователем.

Поскольку подлинность пользователя должна быть установлена, чтобы эти атаки сработали, личные данные пользователя, тем не менее, остаются защищенными (если сама учетная запись пользователя не оказалась раскрыта). Однако данные, которые ваше приложение должно держать в секрете от пользователей, например ключи, используемые для лицензирования или управления цифровыми правами, не находятся в безопасности. Таким образом, ELS — неподходящее место для хранения такой информации. Это подходит для хранения личных данных пользователя, например паролей.

Потеря данных в ELS может произойти по ряду причин. Например, пользователь может удалить приложение и зашифрованный файл. Или идентификатор издателя может измениться в результате обновления. Таким образом, ELS подходит для хранения личных данных пользователя, но не в качестве постоянного защищенного хранилища.

Параметр `stronglyBound` в значении `true` использовать не рекомендуется. Значение `true` данного параметра не обеспечивает дополнительного уровня защиты данных. Кроме того, при обновлении приложения доступ к данным становится невозможным, даже при сохранении исходного идентификатора издателя.

Скорость работы зашифрованного локального хранилища может снижаться, если объем хранимых данных превышает 10 МБ.

При установке приложения AIR программа-деинсталлятор не удаляет файлы из локального хранилища.

**Передовой опыт**

Для наиболее эффективного использования ELS соблюдайте следующие рекомендации.

- Используйте ELS для хранения важнейших данных пользователя, таких как пароли (при значении false параметра stronglyBound).
- Не используйте ELS для хранения секретов приложений, таких как ключи системы управления цифровыми правами или маркеры лицензий.
- Необходимо обеспечить приложению возможность восстановить данные, хранящиеся в ELS, в случае потери данных ELS. Например, этого можно добиться путем запроса на повторный ввод параметров доступа данного пользователя.
- Не используйте параметр `stronglyBound`.
- Если параметру `stronglyBound` присвоено значение `true`, то не перемещайте данные в хранилище во время обновления. Вместо этого после обновления создайте хранилище данных заново.
- Не сохраняйте в хранилище значительные объемы данных. Для объемных данных лучше всего использовать зашифрованную базу данных AIR SQL.

### Дополнительные разделы справки

flash.data.EncryptedLocalStore

## Помещение данных в зашифрованное локальное хранилище

Для помещения данных в локальное хранилище используйте статический метод `setItem()` класса EncryptedLocalStore. Данные хранятся в хэш-таблице с ключами-строками; данные сохраняются в виде байтовых массивов.

Например, в коде ниже строка сохраняется в зашифрованном локальном хранилище:

```
var str:String = "Bob";
var bytes:ByteArray = new ByteArray();
bytes.writeUTFBytes(str);
EncryptedLocalStore.setItem("firstName", bytes);
```

Третий параметр метода `setItem()` — `stronglyBound` — не является обязательным. Если этот параметр принимает значение `true`, то зашифрованное локальное хранилище привязывает хранящиеся в нем элементы к цифровой подписи сохраняющего приложения AIR и его битам:

```
var str:String = "Bob";
var bytes:ByteArray = new ByteArray();
bytes.writeUTFBytes(str);
EncryptedLocalStore.setItem("firstName", bytes, false);
```

Если значение свойства `stronglyBound` элемента равно `true`, последующие вызовы метода `getItem()` будут возможны, только если вызов осуществляется тем же приложением AIR, что и сохранение (если данные в файлах каталога приложения не менялись). Если вызывающее приложение AIR отличается от сохраняющего, то при вызове метода `getItem()` для элемента с сильной связью генерируется исключение Error. При обновлении приложения данные с сильной связью, ранее записанные в зашифрованное локальное хранилище, нельзя будет считать. Установка для `stronglyBound` значения `true` игнорируется на мобильных устройствах. Этот параметр всегда обрабатывается так, будто ему задано значение `false`.

Если параметр `stronglyBound` имеет значение `false` (по умолчанию), то для считывания данных должен оставаться без изменения только идентификатор издателя приложения. Биты приложения могут измениться (и они должны быть подписаны тем же издателем), но они могут не совпадать с битами в приложении, сохранившем данные. Обновленные приложения с тем же идентификатором издателя, что и у исходного приложения, по-прежнему будут иметь доступ к данным.

***Примечание.*** *На практике параметр `stronglyBound` в значении `true` не добавляет дополнительного уровня защиты данных. Таким образом, злоумышленник может внести изменения в приложение для получения доступа к данным, хранящимся в ELS. Кроме того, данные защищены от внешних угроз, не исходящих от пользователя, независимо от того, принимает ли параметр `stronglyBound` значение `true` или `false`. Поэтому устанавливать для параметра `stronglyBound` значение `true` не имеет смысла.*

## Доступ к данным, находящимся в зашифрованном локальном хранилище

**Adobe AIR 1.0 и более поздних версий**

Значение может быть извлечено из зашифрованного локального хранилища методом `EncryptedLocalStore.getItem()`, как показано ниже:

```
var storedValue:ByteArray = EncryptedLocalStore.getItem("firstName");
trace(storedValue.readUTFBytes(storedValue.length)); // "Bob"
```

## Удаление данных, находящихся в зашифрованном локальном хранилище

**Adobe AIR 1.0 и более поздних версий**

Значение может быть удалено из зашифрованного локального хранилища методом `EncryptedLocalStore.removeItem()`, как показано ниже:

```
EncryptedLocalStore.removeItem("firstName");
```

Все данные могут быть удалены из зашифрованного локального хранилища методом `EncryptedLocalStore.reset()`, как показано ниже:

```
EncryptedLocalStore.reset();
```

# Глава 40. Работа с локальными базами данных SQL в AIR

**Adobe AIR 1.0 и более поздних версий**

Adobe® AIR® включает возможности создания локальных баз данных SQL и работы с ними. Среда выполнения включает в себя механизм базы данных SQL с поддержкой многих стандартных функций SQL и использует открытую систему базы данных SQLite. Локальную базу данных SQL можно использовать для хранения локальных данных. Например, ее можно использовать для данных приложения, пользовательских настроек приложения, документов или других типов данных, которые приложение должно сохранять локально.

## О локальных базах данных SQL

**Adobe AIR 1.0 и более поздних версий**

Краткое разъяснение и примеры кодов с использованием баз данных SQL см. в следующих статьях по быстрому началу работы в центре Adobe Developer Connection:

- Асинхронная работа с локальной базой данных SQL (Flex)
- Синхронная работа с локальной базой данных SQL (Flex)
- Использование зашифрованных баз данных (Flex)
- Асинхронная работа с локальной базой данных SQL (Flash)
- Синхронная работа с локальной базой данных SQL (Flash)
- Использование зашифрованных баз данных (Flash)

Adobe AIR включает в себя механизм реляционной базы данных на основе SQL, который работает в среде выполнения, а данные хранятся в файлах базы данных на локальном компьютере, на котором выполняется приложение AIR (например, на жестком диске компьютера). За счет локального выполнения базы данных и локального хранения файлов данных приложение AIR может использовать базу данных независимо от наличия сетевого подключения. Таким образом, механизм локальной базы данных SQL среды выполнения обеспечивает удобство хранения постоянных локальных данных приложения, особенно если у вас есть опыт работы с реляционными базами данных и базами данных SQL.

### Применение локальных бах данных SQL

**Adobe AIR 1.0 и более поздних версий**

Функция локальной базы данных SQL в AIR может использоваться в любых целях, имеющих отношение к хранению данных приложения на локальном компьютере пользователя. Adobe AIR включает в себя несколько механизмов для локального хранения данных, каждый из которых имеет свои плюсы. Ниже приводятся возможные случаи использования локальной базы данных SQL в приложении AIR:

- В приложении для работы с данными (например, адресная книга) база данных может использоваться для хранения данных главного приложения.

- В приложении для работы с документами, с помощью которого пользователи создают документы для сохранения и, возможно, общего доступа, каждый документ можно сохранить в виде файла базы данных в местоположении, указанном пользователем. (Но обратите внимание, что, если база данных не зашифрована, любое приложение AIR сможет открыть файл этой базы данных. Шифрование рекомендуется для наиболее важных документов.)
- В приложении для работы по сети база данных может использоваться для хранения локального кэша данных приложения или для временного хранения данных, когда сетевое подключение недоступно. Можно создать механизм для синхронизации локальной базы данных с сетевым хранилищем данных.
- В любом приложении база данных может использоваться для хранения настроек приложения отдельных пользователей, например параметров пользователя или сведений о приложении, таких как размер и расположение окна.

### Дополнительные разделы справки

Кристоф Конрац (Christophe Coenraets): Каталог сотрудников на AIR for Android

Реймонд Камден (Raymond Camden): jQuery и AIR — переход от веб-страницы к приложению

## О базах данных AIR и файлах баз данных

**Adobe AIR 1.0 и более поздних версий**

Каждая отдельная локальная база данных SQL в Adobe AIR хранится в виде одного файла в файловой системе компьютера. Среда выполнения включает в себя механизм базы данных SQL, который управляет созданием и структурированием файлов базы данных, а также операциями с данными и извлечением данных из файла базы данных. Среда выполнения не указывает на то, как и где хранятся данные базы данных в файловой системе; каждая база данных полностью хранится в одном файле. Вы сами задаете местоположение в файловой системе для хранения файла базы данных. Одно приложение AIR может обращаться к одной или нескольким отдельным базам данных (т. е. к отдельным файлам баз данных). Поскольку среда выполнения хранит каждую базу данных в отдельном файле в файловой системе, базу данных можно найти в соответствии с заданной для приложения процедурой и ограничениями операционной системы на доступ к файлам. Каждый пользователь может иметь свой файл базы данных, либо к файлу базы данных могут обращаться все пользователи приложения с одного компьютера с общими данными. Поскольку данные являются локальными для каждого компьютера, пользователи с разных компьютеров не имеют автоматического общего доступа к этим данным. Механизм локальной базы данных SQL не имеет функции выполнения инструкций SQL в отношении удаленных или серверных баз данных.

## О реляционных базах данных

**Adobe AIR 1.0 и более поздних версий**

Реляционная база данных представляет собой механизм для хранения (и извлечения) данных на компьютере. Данные упорядочены в таблицы: в строках представлены записи или элементы, а столбцы (иногда называемые полями) разделяют отдельные значения каждой записи. Например, в приложении «адресная книга» может быть таблица «Друзья». В этом случае каждая строка таблицы будет представлять отдельного друга, информация о котором хранится в базе данных. Столбцы таблицы будут содержать такие сведения, как имя, фамилия, дата рождения и т. д. Для каждой строки, представляющей друга, в базе данных хранится отдельное значение каждого столбца.

Реляционные базы данных предназначены для хранения сложных данных, когда один элемент связан с элементами другого типа. В реляционной базе данных любые данные со связью «одна-с-несколькими» — когда одна запись связана с несколькими записями другого типа — делятся между различными таблицами. Например, вы хотите, чтобы в адресной книге хранилось несколько телефонных номеров каждого друга — это и есть связь «одна-с-несколькими». Таблица «друзья» будет содержать все личные данные о каждом друге. Другая таблица «Телефоны» будет содержать все телефонные номера всех друзей.

Помимо хранения данных о друзьях и телефонных номеров каждой таблице потребуются данные для отслеживания взаимосвязи между двумя таблицами, т. е. для поиска соответствия между отдельными записями о друзьях и номерами их телефонов. Такие данные называются первичным ключом — это уникальный идентификатор, благодаря которому каждая строка таблицы отличается от другой. Первичный ключ может быть «естественным», т. е. представлять собой один из элементов данных, который является естественным отличительным признаком каждой записи в таблице. В таблице «Друзья», если вы знаете, что ни у одного из друзей не совпадает дата рождения, можно использовать столбец даты рождения в качестве первичного ключа (естественного ключа) этой таблицы. Если естественный ключ отсутствует, создается отдельный столбец первичного ключа, например «Идентификатор друга» — искусственно созданное значение, используемое приложением для различения строк.

При помощи первичного ключа можно устанавливать связи между несколькими таблицами. Предположим, в таблице «Друзья» есть столбец «Идентификатор друга», который содержит уникальный номер для каждой строки (каждого друга). Связанная с ней таблица «Телефонные номера» может состоять из двух столбцов: один — с идентификатором друга, которому принадлежит этот телефонный номер, второй — с самим телефонным номером. Таким образом, независимо от количества телефонных номеров одного пользователя, все они могут храниться в таблице «Телефонные номера» и быть связанными с соответствующим другом при помощи первичного ключа «Идентификатор друга». Когда первичный ключ одной таблицы используется в связанной таблице для указания на связь между записями, значение связанной таблицы называется внешним ключом. В отличие от многих баз данных механизм локальной базы данных AIR не позволяет создавать ограничения внешнего ключа для автоматической проверки наличия соответствующей строки в таблице первичного ключа при вставке или обновлении внешнего ключа. Тем не менее связи с внешними ключами являются важной частью структуры реляционной базы данных, и внешние ключи необходимо использовать при создании связей между таблицами базы данных.

## О языке SQL

**Adobe AIR 1.0 и более поздних версий**

Язык структурированных запросов (SQL) используется в реляционных базах данных для выполнения операций над данными и извлечения данных. SQL скорее является описательным языком, чем процедурным. Вместо того, чтобы предписывать компьютеру способ извлечения данных, инструкция SQL описывает набор необходимых вам данных. Механизм базы данных определяет способ извлечения этих данных.

Язык SQL был стандартизирован Американским национальным институтом стандартов (ANSI). Локальная база данных SQL в Adobe AIR поддерживает большую часть стандарта SQL-92.

Конкретные описания языка SQL, поддерживаемого в Adobe AIR, см. в разделе ««Поддержка SQL в локальных базах данных» на странице 1194».

## О классах базы данных SQL

**Adobe AIR 1.0 и более поздних версий**

Для работы с локальными базами данных SQL в ActionScript 3.0 используются экземпляры этих классов, входящие в пакет flash.data:

| Класс | Описание |
|---|---|
| flash.data.SQLConnection | Позволяет создавать и открывать базы данных (файлы баз данных), а также содержит методы для выполнения операций на уровне базы данных и для управления транзакциями базы данных. |
| flash.data.SQLStatement | Представляет собой одну инструкцию SQL (один запрос или одну команду), выполняющуюся в отношении базы данных, включая определение текста инструкции и задание значений параметров. |
| flash.data.SQLResult | Позволяет получать информацию о выполнении инструкции и ее результатах, например строки результата инструкции `SELECT`, количество строк, задействованных в инструкции `UPDATE` или `DELETE`, и т. д. |

Чтобы получить данные о схеме, описывающей структуру базы данных, используются эти классы из пакета flash.data:

| Класс | Описание |
|---|---|
| flash.data.SQLSchemaResult | Служит контейнером для результатов схемы базы данных, полученных вызовом метода `SQLConnection.loadSchema()`. |
| flash.data.SQLTableSchema | Предоставляет описание отдельной таблицы базы данных. |
| flash.data.SQLViewSchema | Предоставляет описание отдельного представления базы данных. |
| flash.data.SQLIndexSchema | Предоставляет описание отдельного столбца таблицы или представления базы данных. |
| flash.data.SQLTriggerSchema | Предоставляет описание отдельного триггера базы данных. |

Другие классы пакета flash.data содержат константы, которые используются с классом SQLConnection и классом SQLColumnSchema:

| Класс | Описание |
|---|---|
| flash.data.SQLMode | Определяет набор констант, представляющих возможные значения параметра `openMode` методов `SQLConnection.open()` и `SQLConnection.openAsync()`. |
| flash.data.SQLColumnNameStyle | Определяет набор констант, представляющих возможные значения свойства `SQLConnection.columnNameStyle`. |
| flash.data.SQLTransactionLockType | Определяет набор констант, представляющих возможные значения параметра option метода `SQLConnection.begin()`. |
| flash.data.SQLCollationType | Определяет набор констант, представляющих возможные значения свойства `SQLColumnSchema.defaultCollationType` и параметра `defaultCollationType` конструктора `SQLColumnSchema()`. |

Кроме того, следующие классы пакета flash.events представляют используемые события (и соответствующие константы):

| Класс | Описание |
|---|---|
| flash.events.SQLEvent | Определяет события, отправляемые экземпляром SQLConnection или SQLStatement при успешном выполнении какой-либо из его операций. Каждой операции соответствует константа типа события, определенная в классе SQLEvent. |
| flash.events.SQLErrorEvent | Определяет событие, отправляемое экземпляром SQLConnection или SQLStatement в случае возникновения ошибки во время выполнения какой-либо из его операций. |
| flash.events.SQLUpdateEvent | Определяет событие, отправляемое экземплярами SQLConnection при изменении данных таблицы в одной из подключенных баз данных в результате выполнения инструкции SQL `INSERT`, `UPDATE` или `DELETE`. |

Наконец, следующие классы пакета flash.errors предоставляют сведения об ошибках в работе базы данных:

| Класс | Описание |
|---|---|
| flash.errors.SQLError | Предоставляет сведения об ошибке в работе базы данных, включая данные об операции, которая выполнялась, и причине сбоя. |
| flash.errors.SQLErrorOperation | Определяет набор констант, представляющих возможные значения свойства `operation` класса SQLError, указывающего на операцию базы данных, во время выполнения которой произошла ошибка. |

## О режимах синхронного и асинхронного выполнения

**Adobe AIR 1.0 и более поздних версий**

При написании кода для работы с локальной базой данных SQL указывается один из двух режимов выполнения операций этой базы данных: асинхронный или синхронный. В общем, примеры кода иллюстрируют выполнение операций в обоих режимах, чтобы вы могли использовать наиболее соответствующий вашим задачам пример.

В режиме асинхронного выполнения вы передаете среде выполнения инструкцию, и среда выполнения отправляет событие при успешном или неудачном завершении запрошенной вами операции. Сначала вы предписываете механизму базы данных выполнить операцию. Механизм базы данных работает в фоновом режиме, а приложение продолжает выполняться. Наконец, по завершении операции (или сбое) механизм базы данных отправляет событие. Написанный код, инициированный этим событием, выполняет последующие операции. Такой подход имеет существенное преимущество: среда выполнения выполняет операции базы данных в фоновом режиме, тогда как код главного приложения продолжает выполняться. Если операция базы данных требует значительного количества времени, приложение продолжает выполняться. Что самое важное, пользователь может продолжать взаимодействие с приложением без «застывания» экрана. Тем не менее, код асинхронной операции сложнее написать, чем другой код. Сложность возникает в случаях, когда несколько зависимых операций должны распределяться среди различных методов прослушивателей событий.

Концептуально, проще кодировать операции как единую последовательность шагов, т. е. набор синхронных операций, а не как набор операций, разделенных на несколько методов прослушивателей событий. Помимо асинхронных операций базы данных Adobe AIR также позволяет выполнять операции базы данных синхронно. В синхронном режиме операции выполняются не в фоновом режиме. Вместо этого они выполняются в той же последовательности, что и другой код приложения. Вы предписываете механизму базы данных выполнить операцию. На этом этапе код приостанавливает выполнение на время работы механизма базы данных. По завершении операции выполнение продолжается со следующей строки кода.

Асинхронное или синхронное выполнение операций задается на уровне SQLConnection. При использовании одного подключения базы данных невозможно выполнять одни операции или инструкции синхронно, а другие асинхронно. Задание синхронного или асинхронного режима выполнения для SQLConnection осуществляется путем вызова метода SQLConnection для открытия базы данных. При вызове `SQLConnection.open()` подключение использует синхронный режим выполнения, а при вызове `SQLConnection.openAsync()` используется асинхронный режим выполнения. Как только экземпляр SQLConnection подключается к базе данных при помощи метода `open()` или `openAsync()`, для него устанавливается синхронный или асинхронный режим выполнения, пока подключение к базе данных не будет прервано и установлено заново.

Каждый режим выполнения имеет свои преимущества. Хотя оба режима имеют много общих аспектов, при работе в каждом режиме следует учитывать и их отличия. Дополнительную информацию по этой теме, а также рекомендации по работе в каждом из режимов, см. в разделе «Использование синхронных и асинхронных операций базы данных» на странице 809.

# Создание и изменение базы данных

**Adobe AIR 1.0 и более поздних версий**

Для того чтобы приложение могло добавлять или извлекать данные, необходимо создать базу данных с таблицами, к которым будет обращаться приложение. В данном разделе приводится описание задач по созданию базы данных и структуры данных в этой базе данных. Хотя эти задачи используются менее часто, чем вставка и извлечение данных, они необходимы для большинства приложений.

### Дополнительные разделы справки

Mind the Flex: Обновление существующей базы данных AIR

## Создание базы данных

**Adobe AIR 1.0 и более поздних версий**

Для создания файла базы данных необходимо сначала создать экземпляр SQLConnection. Метод `open()` вызывается для его открытия в синхронном режиме выполнения, а метод `openAsync()` — для его открытия в асинхронном режиме. Методы `open()` и `openAsync()` используются для открытия подключения к базе данных. При передаче экземпляра File, который ссылается на несуществующее местоположение файла, параметру `reference` (первый параметр) метод `open()` или `openAsync()` создает файл базы данных в этом местоположении и открывает подключение к вновь созданной базе данных.

Независимо от того, вызывается метод `open()` или `openAsync()` для создания базы данных, имя файла базы данных может быть любым допустимым именем файла с любым расширением. При вызове метода `open()` или `openAsync()` со значением `null` для параметра `reference` создается новая база данных «в памяти», а не файл базы данных на диске.

Следующий код иллюстрирует процесс создания файла базы данных (новой базы данных) с использованием асинхронного режима выполнения. В данном случае файл базы данных сохраняется в «Указание на каталог хранилища приложения» на странице 722 под именем DBSample.db:

```
import flash.data.SQLConnection;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

conn.openAsync(dbFile);

function openHandler(event:SQLEvent):void
{
    trace("the database was created successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                conn.openAsync(dbFile);
            }

            private function openHandler(event:SQLEvent):void
            {
                trace("the database was created successfully");
            }

            private function errorHandler(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

***Примечание.*** *Хотя класс File позволяет указывать определенный в системе путь к файлу, этот путь может привести к приложениям, которые не работают на разных платформах. Например, путь C:\Documents and Settings\joe\test.db работает только в ОС Windows. По этим причинам самым оптимальным решением является использование статических свойств класса File, например* `File.applicationStorageDirectory`*, и метода* `resolvePath()` *(как показано в предыдущем примере). Дополнительные сведения см. в разделе ««Пути к объектам File» на странице 718».*

Для синхронного выполнения операций при открытии подключения к базе данных при помощи экземпляра SQLConnection вызовите метод `open()`. Следующий пример иллюстрирует создание и открытие экземпляра SQLConnection, выполняющего операции в синхронном режиме:

```
import flash.data.SQLConnection;
import flash.errors.SQLError;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

try
{
    conn.open(dbFile);
    trace("the database was created successfully");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.errors.SQLError;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                try
                {
                    conn.open(dbFile);
                    trace("the database was created successfully");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

## Создание таблиц базы данных

**Adobe AIR 1.0 и более поздних версий**

При создании таблицы базы данных требуется выполнить инструкцию SQL в отношении этой базы данных, следуя той же процедуре, что и для выполнения таких инструкций SQL, как `SELECT`, `INSERT` и т. п. Для создания таблицы используется инструкция `CREATE TABLE`, которая включает в себя определения столбцов и ограничений новой таблицы. Дополнительную информацию о выполнении инструкций SQL см. в разделе «Работа с инструкциями SQL» на странице 784.

Следующий пример иллюстрирует создание таблицы с именем «employees» в существующем файле базы данных с использованием асинхронного режима выполнения. Заметьте, что данный код предполагает наличие инициализированного и подключенного к базе данных экземпляра SQLConnection с именем `conn`.

```
import flash.data.SQLConnection;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

var createStmt:SQLStatement = new SQLStatement();
createStmt.sqlConnection = conn;

var sql:String =
    "CREATE TABLE IF NOT EXISTS employees (" +
    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "   firstName TEXT, " +
    "   lastName TEXT, " +
    "   salary NUMERIC CHECK (salary > 0)" +
    ")";
createStmt.text = sql;

createStmt.addEventListener(SQLEvent.RESULT, createResult);
createStmt.addEventListener(SQLErrorEvent.ERROR, createError);

createStmt.execute();

function createResult(event:SQLEvent):void
{
    trace("Table created");
}

function createError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                var createStmt:SQLStatement = new SQLStatement();
                createStmt.sqlConnection = conn;

                var sql:String =
                    "CREATE TABLE IF NOT EXISTS employees (" +
                    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    "   firstName TEXT, " +
                    "   lastName TEXT, " +
                    "   salary NUMERIC CHECK (salary > 0)" +
                    ")";
                createStmt.text = sql;

                createStmt.addEventListener(SQLEvent.RESULT, createResult);
                createStmt.addEventListener(SQLErrorEvent.ERROR, createError);

                createStmt.execute();
            }

            private function createResult(event:SQLEvent):void
            {
                trace("Table created");
            }

            private function createError(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Следующий пример иллюстрирует создание таблицы с именем «employees» в существующем файле базы данных с использованием синхронного режима выполнения. Заметьте, что данный код предполагает наличие инициализированного и подключенного к базе данных экземпляра SQLConnection с именем `conn`.

```
import flash.data.SQLConnection;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

var createStmt:SQLStatement = new SQLStatement();
createStmt.sqlConnection = conn;

var sql:String =
    "CREATE TABLE IF NOT EXISTS employees (" +
    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "   firstName TEXT, " +
    "   lastName TEXT, " +
    "   salary NUMERIC CHECK (salary > 0)" +
    ")";
createStmt.text = sql;

try
{
    createStmt.execute();
    trace("Table created");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                var createStmt:SQLStatement = new SQLStatement();
                createStmt.sqlConnection = conn;

                var sql:String =
                    "CREATE TABLE IF NOT EXISTS employees (" +
                    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    "   firstName TEXT, " +
                    "   lastName TEXT, " +
                    "   salary NUMERIC CHECK (salary > 0)" +
                    ")";
                createStmt.text = sql;

                try
                {
                    createStmt.execute();
                    trace("Table created");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

# Операции над данными базы данных SQL

**Adobe AIR 1.0 и более поздних версий**

Существует несколько стандартных задач, которые выполняются при работе с локальными базами данных SQL. В эти задачи входит подключение к базе данных, добавление данных в таблицы или извлечение данных из таблиц базы данных. Существуют также некоторые моменты, которые следует учитывать при выполнении этих задач, например работа с типами данных и обработка ошибок.

Существует также несколько менее часто выполняющихся задач по работе с базой данных, которые необходимо выполнить перед выполнением других, стандартных задач. Например, перед подключением к базе данных и извлечением данных из таблицы необходимо создать базу данных и структуру таблиц в этой базе данных. Такие менее часто выполняющиеся исходные задачи описаны в разделе «Создание и изменение базы данных» на странице 773.

Можно выбрать асинхронное выполнение операций базы данных, при котором механизм базы данных работает в фоновом режиме и оповещает при успешном или неудачном выполнении операции отправкой события. Эти операции могут выполняться и в синхронном режиме. В таком случае операции базы данных выполняются одна за другой, а приложение (включая обновление экрана) ожидает завершения операций для выполнения последующего кода. Дополнительную информацию о работе в синхронном и асинхронном режимах выполнения см. в разделе «Использование синхронных и асинхронных операций базы данных» на странице 809.

## Подключение к базе данных

**Adobe AIR 1.0 и более поздних версий**

Перед выполнением операций базы данных необходимо сначала открыть подключение к файлу базы данных. Экземпляр SQLConnection представляет подключение к одной или нескольким базам данных. Первая база данных, подключенная с использованием экземпляра SQLConnection, является «главной» базой данных. Эта база данных подключается с использованием метода `open()` (для использования синхронного режима выполнения) или метода `openAsync()` (для использования асинхронного режима выполнения).

При открытии базы данных с использованием асинхронной операции `openAsync()` зарегистрируйте событие `open` экземпляра SQLConnection, чтобы узнать о завершении операции `openAsync()`. Зарегистрируйте событие `error` экземпляра SQLConnection, чтобы узнать о сбое в выполнении операции.

Следующий пример иллюстрирует открытие существующего файла базы данных для асинхронного выполнения. Файл базы данных называется DBSample.db и находится в пользовательском «Указание на каталог хранилища приложения» на странице 722.

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

conn.openAsync(dbFile, SQLMode.UPDATE);

function openHandler(event:SQLEvent):void
{
    trace("the database opened successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLMode;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                conn.openAsync(dbFile, SQLMode.UPDATE);
            }

            private function openHandler(event:SQLEvent):void
            {
                trace("the database opened successfully");
            }

            private function errorHandler(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Следующий пример иллюстрирует открытие существующего файла базы данных для синхронного выполнения. Файл базы данных называется DBSample.db и находится в пользовательском «Указание на каталог хранилища приложения» на странице 722.

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.errors.SQLError;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

try
{
    conn.open(dbFile, SQLMode.UPDATE);
    trace("the database opened successfully");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLMode;
            import flash.errors.SQLError;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                try
                {
                    conn.open(dbFile, SQLMode.UPDATE);
                    trace("the database opened successfully");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Обратите внимание на то, что в вызове метода `openAsync()` в примере асинхронного выполнения и в вызове метода `open()` в примере синхронного выполнения вторым аргументом является константа `SQLMode.UPDATE`. При задании `SQLMode.UPDATE` для второго параметра (`openMode`) среда выполнения выдает ошибку, если указанный файл не существует. При передаче `SQLMode.CREATE` параметру `openMode` (или при отключении параметра `openMode`) среда выполнения пытается создать файл базы данных, если указанный файл не существует. Но если файл уже существует, он будет открыт, что равносильно использованию `SQLMode.Update`. Можно также задать `SQLMode.READ` для параметра `openMode`, чтобы открыть существующую базу данных только для чтения. В таком случае можно извлекать данные из базы данных, но добавлять, удалять или изменять данные нельзя.

## Работа с инструкциями SQL

**Adobe AIR 1.0 и более поздних версий**

Отдельная инструкция SQL (запрос или команда) представлена в среде выполнения в виде объекта SQLStatement. Выполните следующие действия для создания и выполнения инструкции SQL:

**Создайте экземпляр SQLStatement.**
Объект SQLStatement представляет инструкцию SQL в вашем приложении.

```
var selectData:SQLStatement = new SQLStatement();
```

**Укажите базу данных, в отношении которой выполняется запрос.**
Для этого задайте для свойства `sqlConnection` объекта SQLStatement экземпляр SQLConnection, подключенный к нужной базе данных.

```
// A SQLConnection named "conn" has been created previously
selectData.sqlConnection = conn;
```

**Укажите саму инструкцию SQL.**
Создайте текст инструкции в виде строки и присвойте его свойству `text` экземпляра SQLStatement.

```
selectData.text = "SELECT col1, col2 FROM my_table WHERE col1 = :param1";
```

**Определите функции для обработки результата операции execute (только для режима асинхронного выполнения).**
При помощи метода `addEventListener()` зарегистрируйте функции в качестве прослушивателей событий `result` и `error` экземпляра SQLStatement.

```
// using listener methods and addEventListener()

selectData.addEventListener(SQLEvent.RESULT, resultHandler);
selectData.addEventListener(SQLErrorEvent.ERROR, errorHandler);

function resultHandler(event:SQLEvent):void
{
    // do something after the statement execution succeeds
}

function errorHandler(event:SQLErrorEvent):void
{
    // do something after the statement execution fails
}
```

Вместо этого можно задать методы прослушивателей с использованием объекта Responder. В таком случае нужно создать экземпляр Responder и связать с ним методы прослушивателей.

```
// using a Responder (flash.net.Responder)

var selectResponder = new Responder(onResult, onError);

function onResult(result:SQLResult):void
{
    // do something after the statement execution succeeds
}

function onError(error:SQLError):void
{
    // do something after the statement execution fails
}
```

**Если текст инструкции содержит определения параметров, присвойте значения этим параметрам.**
Для присвоения значений параметрам используйте свойство ассоциативного массива `parameters` экземпляра SQLStatement.

```
selectData.parameters[":param1"] = 25;
```

**Выполните инструкцию SQL.**
Вызовите метод `execute()` экземпляра SQLStatement.

```
// using synchronous execution mode
// or listener methods in asynchronous execution mode
selectData.execute();
```

Более того, если вы используете Responder вместо прослушивателей событий в режиме асинхронного выполнения, передайте экземпляр Responder методу `execute()`.

```
// using a Responder in asynchronous execution mode
selectData.execute(-1, selectResponder);
```

Конкретные примеры, иллюстрирующие выполнение этих шагов, см. в разделах:

«Извлечение данных из базы данных» на странице 788

«Вставка данных» на странице 798

«Изменение или удаление данных» на странице 804

## Использование параметров в инструкциях

**Adobe AIR 1.0 и более поздних версий**

Использование параметров инструкции SQL позволяет создать инструкцию SQL для неоднократного использования. При использовании параметров инструкций значения в инструкциях могут меняться (например, значения, вставленные в инструкцию `INSERT`), но основной текст инструкции остается неизменным. Следовательно, использование параметров улучшает производительность и упрощает кодирование приложения.

## Основные сведения о параметрах инструкций

**Adobe AIR 1.0 и более поздних версий**

Часто приложение использует одну и ту же инструкцию SQL несколько раз с незначительными изменениями. Рассмотрим приложение для инвентаризации, работая в котором пользователь может добавить новые предметы хранения в базу данных. Код приложения, который добавляет предмет хранения в базу данных, выполняет инструкцию SQL `INSERT`, которая добавляет данные в базу данных. Однако каждый раз инструкция выполняется с некоторыми изменениями. Вставляемые в таблицу значения отличаются, так как они соответствуют конкретному добавляемому предмету хранения.

В случаях с инструкцией SQL, которая используется несколько раз с различными значениями, рекомендуется использовать инструкцию SQL, текст которой содержит параметры, а не литеральные значения. Параметр представляет собой заполнитель в тексте инструкции, который заменяется фактическим значением при каждом выполнении инструкции. Для использования параметров в инструкции SQL необходимо, как обычно, создать экземпляр SQLStatement. В инструкции SQL, присвоенной свойству `text`, используйте заполнители для параметров вместо литеральных значений. Затем определите значение каждого параметра, задав значение элемента в свойстве `parameters` экземпляра SQLStatement. Свойство `parameters` представляет собой ассоциативный массив, поэтому конкретное значение задается с использованием следующего синтаксиса:

```
statement.parameters[parameter_identifier] = value;
```

Элемент *parameter_identifier* является строкой, если используется именованный параметр, или целочисленным индексом, если используется неименованный параметр.

## Использование именованных параметров

**Adobe AIR 1.0 и более поздних версий**

Параметр может быть именованным. Именованный параметр имеет определенное имя, используемое базой данных для поиска соответствия между значением параметра и позицией заполнителя в тексте инструкции. Имя параметра состоит из символа «:» или «@», за которым следует имя, как в следующих примерах:

```
:itemName
@firstName
```

Следующий код иллюстрирует использование именованных параметров:

```
var sql:String =
    "INSERT INTO inventoryItems (name, productCode)" +
    "VALUES (:name, :productCode)";

var addItemStmt:SQLStatement = new SQLStatement();
addItemStmt.sqlConnection = conn;
addItemStmt.text = sql;

// set parameter values
addItemStmt.parameters[":name"] = "Item name";
addItemStmt.parameters[":productCode"] = "12345";

addItemStmt.execute();
```

## Использование неименованных параметров

**Adobe AIR 1.0 и более поздних версий**

Вместо именованных можно также использовать неименованные параметры. Неименованный параметр обозначается в инструкции SQL символом «?». Каждому параметру присваивается числовой индекс в соответствии с порядком следования параметров в инструкции, начиная с индекса 0 для первого параметра. Следующий пример является вариантом предыдущего примера с использованием неименованных параметров:

```
var sql:String =
    "INSERT INTO inventoryItems (name, productCode)" +
    "VALUES (?, ?)";

var addItemStmt:SQLStatement = new SQLStatement();
addItemStmt.sqlConnection = conn;
addItemStmt.text = sql;

// set parameter values
addItemStmt.parameters[0] = "Item name";
addItemStmt.parameters[1] = "12345";

addItemStmt.execute();
```

## Преимущества использования параметров

**Adobe AIR 1.0 и более поздних версий**

Использование параметров в инструкции SQL имеет несколько преимуществ:

**Лучшая производительность** Экземпляр SQLStatement, использующий параметры, работает более эффективно по сравнению с экземпляром, динамически создающим текст SQL при каждом выполнении. Производительность улучшается потому, что инструкция создается один раз и может использоваться многократно с различными значениями параметров без необходимости повторной компиляции инструкции SQL.

**Явный ввод данных** Параметры позволяют вводить подстановочные значения, которые неизвестны в момент построения инструкции SQL. Использование параметров является единственной гарантией создания класса хранилища значений, передаваемых в базу данных. Если параметры не используются, среда выполнения пытается преобразовать все значения из текстовых в класс хранилища на основе родственности типа соответствующего столбца.

Дополнительные сведения о классах хранилища и родственных столбцах см. в разделе ««Поддержка типов данных» на странице 1218».

**Лучшая безопасность** Использование параметров позволяет предотвратить зловредные атаки с внедрением SQL. При атаке с внедрением SQL пользователь вводит код SQL в местоположение, доступное для пользователя (например, в поле ввода данных). Если код приложения создаст инструкцию SQL, вставив введенные пользователем данные непосредственно в текст SQL, введенный пользователем код SQL выполняется в отношении базы данных. Следующий пример иллюстрирует вставку введенных пользователем данных в текст SQL. **Не используйте этот метод**:

```
// assume the variables "username" and "password"
// contain user-entered data

var sql:String =
    "SELECT userId " +
    "FROM users " +
    "WHERE username = '" + username + "' " +
    "    AND password = '" + password + "'";

var statement:SQLStatement = new SQLStatement();
statement.text = sql;
```

Использование параметров инструкции вместо вставки введенных пользователем значений в текст инструкции предотвращает атаки с внедрением SQL. Внедрение SQL не может произойти, так как значения параметров воспринимаются как подстановочные значения, а не как часть литерального текста инструкции. Далее приводится рекомендуемая альтернатива предыдущему примеру:

```
// assume the variables "username" and "password"
// contain user-entered data

var sql:String =
    "SELECT userId " +
    "FROM users " +
    "WHERE username = :username " +
    "    AND password = :password";

var statement:SQLStatement = new SQLStatement();
statement.text = sql;

// set parameter values
statement.parameters[":username"] = username;
statement.parameters[":password"] = password;
```

## Извлечение данных из базы данных

**Adobe AIR 1.0 и более поздних версий**

Извлечение данных из базы данных происходит в два этапа. Сначала необходимо выполнить инструкцию SQL `SELECT`, описывающую набор необходимых данных. Затем вы обращаетесь к извлеченным данным, отображаете их или выполняете над ними необходимые операции.

### Выполнение инструкции SELECT

**Adobe AIR 1.0 и более поздних версий**

Для извлечения существующих данных из базы данных используется экземпляр SQLStatement. Присвойте соответствующую инструкцию SQL `SELECT` свойству `text` экземпляра, затем вызовите его метод `execute()`.

Сведения о синтаксисе инструкции `SELECT` см. в разделе ««Поддержка SQL в локальных базах данных» на странице 1194».

Следующий пример иллюстрирует выполнение инструкции `SELECT` для извлечения данных из таблицы «products» с использованием режима асинхронного выполнения.

```
var selectStmt:SQLStatement = new SQLStatement();

// A SQLConnection named "conn" has been created previously
selectStmt.sqlConnection = conn;

selectStmt.text = "SELECT itemId, itemName, price FROM products";

selectStmt.addEventListener(SQLEvent.RESULT, resultHandler);
selectStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);

selectStmt.execute();

function resultHandler(event:SQLEvent):void
{
    var result:SQLResult = selectStmt.getResult();

    var numResults:int = result.data.length;
    for (var i:int = 0; i < numResults; i++)
    {
        var row:Object = result.data[i];
        var output:String = "itemId: " + row.itemId;
        output += "; itemName: " + row.itemName;
        output += "; price: " + row.price;
        trace(output);
    }
}

function errorHandler(event:SQLErrorEvent):void
{
    // Information about the error is available in the
    // event.error property, which is an instance of
    // the SQLError class.
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                var selectStmt:SQLStatement = new SQLStatement();

                // A SQLConnection named "conn" has been created previously
                selectStmt.sqlConnection = conn;

                selectStmt.text = "SELECT itemId, itemName, price FROM products";

                selectStmt.addEventListener(SQLEvent.RESULT, resultHandler);
                selectStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);
```

```
            selectStmt.execute();
        }

        private function resultHandler(event:SQLEvent):void
        {
            var result:SQLResult = selectStmt.getResult();

            var numResults:int = result.data.length;
            for (var i:int = 0; i < numResults; i++)
            {
                var row:Object = result.data[i];
                var output:String = "itemId: " + row.itemId;
                output += "; itemName: " + row.itemName;
                output += "; price: " + row.price;
                trace(output);
            }
        }

        private function errorHandler(event:SQLErrorEvent):void
        {
            // Information about the error is available in the
            // event.error property, which is an instance of
            // the SQLError class.
        }
    ]]>
  </mx:Script>
</mx:WindowedApplication>
```

Следующий пример иллюстрирует выполнение инструкции `SELECT` для извлечения данных из таблицы «products» в режиме синхронного выполнения.

```
var selectStmt:SQLStatement = new SQLStatement();

// A SQLConnection named "conn" has been created previously
selectStmt.sqlConnection = conn;

selectStmt.text = "SELECT itemId, itemName, price FROM products";

try
{
    selectStmt.execute();

    var result:SQLResult = selectStmt.getResult();

    var numResults:int = result.data.length;
    for (var i:int = 0; i < numResults; i++)
    {
        var row:Object = result.data[i];
        var output:String = "itemId: " + row.itemId;
        output += "; itemName: " + row.itemName;
        output += "; price: " + row.price;
        trace(output);
    }
}
catch (error:SQLError)
{
    // Information about the error is available in the
    // error variable, which is an instance of
    // the SQLError class.
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                var selectStmt:SQLStatement = new SQLStatement();

                // A SQLConnection named "conn" has been created previously
                selectStmt.sqlConnection = conn;

                selectStmt.text = "SELECT itemId, itemName, price FROM products";

                try
                {
                    selectStmt.execute();

                    var result:SQLResult = selectStmt.getResult();
```

```
                var numResults:int = result.data.length;
                for (var i:int = 0; i < numResults; i++)
                {
                    var row:Object = result.data[i];
                    var output:String = "itemId: " + row.itemId;
                    output += "; itemName: " + row.itemName;
                    output += "; price: " + row.price;
                    trace(output);
                }
            }
            catch (error:SQLError)
            {
                // Information about the error is available in the
                // error variable, which is an instance of
                // the SQLError class.
            }
        }
    ]]>
</mx:Script>
</mx:WindowedApplication>
```

В режиме асинхронного выполнения по завершении выполнения инструкции экземпляр SQLStatement отправляет событие `result` (`SQLEvent.RESULT`), указывая на успешное выполнение инструкции. Как вариант, если объект Responder передается в качестве аргумента методу `execute()`, вызывается функция обработчика результатов объекта Responder. В синхронном режиме выполнение приостанавливается до завершения операции `execute()`, затем возобновляется со следующей строки кода.

## Доступ к данным результатов выполнения инструкции SELECT

**Adobe AIR 1.0 и более поздних версий**

По завершении выполнения инструкции `SELECT` следующим шагом является получение доступа к извлеченным данным. Чтобы извлечь данные, полученные в результате выполнения инструкции `SELECT`, вызовите метод `getResult()` объекта SQLStatement.

```
var result:SQLResult = selectStatement.getResult();
```

Метод `getResult()` возвращает объект SQLResult. Свойство `data` объекта SQLResult представлено объектом Array, который содержит результаты выполнения инструкции `SELECT`.

```
var numResults:int = result.data.length;
for (var i:int = 0; i < numResults; i++)
{
    // row is an Object representing one row of result data
    var row:Object = result.data[i];
}
```

Каждая строка данных в наборе результатов `SELECT` становится экземпляром Object в массиве `data`. Этот объект имеет свойства, имена которых совпадают с названиями столбцов в наборе результатов. Свойства содержат значения из столбцов набора результатов. Предположим, инструкция `SELECT` имеет набор результатов с тремя столбцами: «itemId», «itemName» и «price». Для каждой строки в наборе результатов создается экземпляр Object со свойствами `itemId`, `itemName` и `price`. Эти свойства содержат значения из соответствующих столбцов.

Следующий код определяет экземпляр SQLStatement, текстом которого является инструкция `SELECT`. Инструкция извлекает строки, содержащие значения столбцов `firstName` и `lastName` всех строк таблицы `employees`. В данном примере используется режим асинхронного выполнения. По завершении выполнения вызывается метод `selectResult()`, а доступ к полученным строкам данных осуществляется посредством метода `SQLStatement.getResult()`, и строки отображаются с использованием метода `trace()`. Заметьте, что данный код предполагает наличие инициализированного и подключенного к базе данных экземпляра SQLConnection с именем `conn`. Он также предполагает, что таблица «employees» уже создана и заполнена данными.

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var selectStmt:SQLStatement = new SQLStatement();
selectStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "SELECT firstName, lastName " +
    "FROM employees";
selectStmt.text = sql;

// register listeners for the result and error events
selectStmt.addEventListener(SQLEvent.RESULT, selectResult);
selectStmt.addEventListener(SQLErrorEvent.ERROR, selectError);

// execute the statement
selectStmt.execute();

function selectResult(event:SQLEvent):void
{
    // access the result data
    var result:SQLResult = selectStmt.getResult();

    var numRows:int = result.data.length;
    for (var i:int = 0; i < numRows; i++)
    {
        var output:String = "";
        for (var columnName:String in result.data[i])
        {
            output += columnName + ": " + result.data[i][columnName] + "; ";
        }
        trace("row[" + i.toString() + "]\t", output);
    }
}

function selectError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var selectStmt:SQLStatement = new SQLStatement();
                selectStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "SELECT firstName, lastName " +
                    "FROM employees";
                selectStmt.text = sql;

                // register listeners for the result and error events
                selectStmt.addEventListener(SQLEvent.RESULT, selectResult);
                selectStmt.addEventListener(SQLErrorEvent.ERROR, selectError);

                // execute the statement
                selectStmt.execute();
            }

            private function selectResult(event:SQLEvent):void
            {
                // access the result data
                var result:SQLResult = selectStmt.getResult();

                var numRows:int = result.data.length;
                for (var i:int = 0; i < numRows; i++)
                {
                    var output:String = "";
                    for (var columnName:String in result.data[i])
                    {
                        output += columnName + ": " + result.data[i][columnName] + "; ";
                    }
                    trace("row[" + i.toString() + "]\t", output);
                }
            }

            private function selectError(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

Следующий код иллюстрирует тот же метод, что и в предыдущем примере, но использует режим синхронного выполнения. В примере определяется экземпляр SQLStatement, текстом которого является инструкция `SELECT`. Инструкция извлекает строки, содержащие значения столбцов `firstName` и `lastName` всех строк таблицы `employees`. Доступ к полученным строкам данных осуществляется посредством метода `SQLStatement.getResult()`, и строки отображаются при помощи метода `trace()`. Заметьте, что данный код предполагает наличие инициализированного и подключенного к базе данных экземпляра SQLConnection с именем `conn`. Он также предполагает, что таблица «employees» уже создана и заполнена данными.

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var selectStmt:SQLStatement = new SQLStatement();
selectStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "SELECT firstName, lastName " +
    "FROM employees";
selectStmt.text = sql;

try
{
    // execute the statement
    selectStmt.execute();

    // access the result data
    var result:SQLResult = selectStmt.getResult();

    var numRows:int = result.data.length;
    for (var i:int = 0; i < numRows; i++)
    {
        var output:String = "";
        for (var columnName:String in result.data[i])
        {
            output += columnName + ": " + result.data[i][columnName] + "; ";
        }
        trace("row[" + i.toString() + "]\t", output);
    }
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var selectStmt:SQLStatement = new SQLStatement();
                selectStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "SELECT firstName, lastName " +
                    "FROM employees";
                selectStmt.text = sql;

                try
                {
                    // execute the statement
                    selectStmt.execute();

                    // access the result data
                    var result:SQLResult = selectStmt.getResult();

                    var numRows:int = result.data.length;
                    for (var i:int = 0; i < numRows; i++)
                    {
                        var output:String = "";
                        for (var columnName:String in result.data[i])
                        {
                            output += columnName + ": ";
                            output += result.data[i][columnName] + "; ";
                        }
                        trace("row[" + i.toString() + "]\t", output);
                    }
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

## Определение типа данных результатов инструкции SELECT

**Adobe AIR 1.0 и более поздних версий**

По умолчанию каждая строка, возвращенная инструкцией `SELECT`, является экземпляром Object, свойства которого имеют те же имена, что и названия столбцов в наборе результатов, а значение каждого столбца является значением соответствующего свойства. Однако перед выполнением инструкции SQL `SELECT` можно присвоить класс свойству `itemClass` экземпляра SQLStatement. Задав свойство `itemClass`, каждая строка, возвращенная инструкцией `SELECT`, является экземпляром указанного класса. Среда выполнения присваивает значения столбцов результатов значениям свойств, устанавливая соответствие между названиями столбцов в наборе результатов `SELECT` и именами свойств в классе `itemClass`.

Каждый класс, являющийся значением свойства `itemClass`, должен иметь конструктор, не требующий параметров. Кроме того, этот класс должен иметь одно свойство для каждого столбца, возвращенного инструкцией `SELECT`. Если для столбца в списке `SELECT` не существует соответствующего имени свойства в классе `itemClass`, это считается ошибкой.

## Извлечение результатов SELECT по частям

**Adobe AIR 1.0 и более поздних версий**

По умолчанию инструкция `SELECT` извлекает все строки набора результатов одновременно. По завершении выполнения инструкции извлеченные данные обрабатываются, например создаются объекты или данные отображаются на экране. Если инструкция возвратила большое количество строк, одновременная обработка всех данных является очень трудоемким процессом для компьютера, что может привести к тому, что пользовательский интерфейс перестанет обновляться.

Улучшить производительность приложения можно, предписав среде выполнения возвращать одновременно определенное количество строк результатов. Таким образом, данные результатов будут возвращаться быстрее. Это также позволяет разделить строки результатов на части, чтобы пользовательский интерфейс мог обновляться после обработки каждого набора строк. Обратите внимание на то, что использование такого метода эффективно только при работе в режиме асинхронного выполнения.

Для извлечения результатов `SELECT` по частям задайте значение первого параметра метода `SQLStatement.execute()` (параметра `prefetch`). Параметр `prefetch` задает количество строк, которое нужно возвратить при первом выполнении инструкции. При вызове метода `execute()` экземпляра SQLStatement задайте значение параметра `prefetch`, чтобы возвратить определенное количество строк.

```
var stmt:SQLStatement = new SQLStatement();
stmt.sqlConnection = conn;

stmt.text = "SELECT ...";

stmt.addEventListener(SQLEvent.RESULT, selectResult);

stmt.execute(20); // only the first 20 rows (or fewer) are returned
```

Инструкция отправляет событие `result`, указывая на то, что доступен первый набор строк результатов. Полученное свойство `data` экземпляра SQLResult содержит строки данных, а его свойство `complete` указывает на то, есть ли еще строки для извлечения. Для извлечения дополнительных строк результатов вызовите метод `next()` экземпляра SQLStatement. Как и в случае с методом `execute()`, первый параметр метода `next()` используется для указания на то, сколько строк возвратить при следующей отправке события result.

```
function selectResult(event:SQLEvent):void
{
    var result:SQLResult = stmt.getResult();
    if (result.data != null)
    {
        // ... loop through the rows or perform other processing ...

        if (!result.complete)
        {
            stmt.next(20); // retrieve the next 20 rows
        }
        else
        {
            stmt.removeEventListener(SQLEvent.RESULT, selectResult);
        }
    }
}
```

SQLStatement отправляет событие `result` каждый раз, когда метод `next()` возвращает последующий набор строк результатов. Следовательно, ту же функцию прослушивателя можно использовать для продолжения обработки результатов (полученных вызовом метода `next()`) до тех пор, пока не будут возвращены все строки.

Дополнительные сведения см. в описаниях для метода `SQLStatement.execute()` (описание параметра `prefetch`) и метода `SQLStatement.next()`.

## Вставка данных

**Adobe AIR 1.0 и более поздних версий**

Добавление данных в базу данных предполагает выполнение инструкции SQL `INSERT`. По завершении выполнения инструкции можно получить доступ к первичному ключу вновь вставленной сроки, если такой ключ был создан базой данных.

### Выполнение инструкции INSERT

**Adobe AIR 1.0 и более поздних версий**

Для добавления данных в таблицу базы данных необходимо создать и выполнить экземпляр SQLStatement, текстом которого является инструкция SQL `INSERT`.

В следующем примере экземпляр SQLStatement используется для добавления строки данных в уже существующую таблицу «employees». Этот пример иллюстрирует вставку данных с использованием режима асинхронного выполнения. Заметьте, что данный код предполагает наличие инициализированного и подключенного к базе данных экземпляра SQLConnection с именем `conn`. Он также предполагает, что таблица «employees» уже создана.

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var insertStmt:SQLStatement = new SQLStatement();
insertStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "INSERT INTO employees (firstName, lastName, salary) " +
    "VALUES ('Bob', 'Smith', 8000)";
insertStmt.text = sql;

// register listeners for the result and failure (status) events
insertStmt.addEventListener(SQLEvent.RESULT, insertResult);
insertStmt.addEventListener(SQLErrorEvent.ERROR, insertError);

// execute the statement
insertStmt.execute();

function insertResult(event:SQLEvent):void
{
    trace("INSERT statement succeeded");
}

function insertError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var insertStmt:SQLStatement = new SQLStatement();
                insertStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "INSERT INTO employees (firstName, lastName, salary) " +
```

```
                "VALUES ('Bob', 'Smith', 8000)";
            insertStmt.text = sql;

            // register listeners for the result and failure (status) events
            insertStmt.addEventListener(SQLEvent.RESULT, insertResult);
            insertStmt.addEventListener(SQLErrorEvent.ERROR, insertError);

            // execute the statement
            insertStmt.execute();
        }

        private function insertResult(event:SQLEvent):void
        {
            trace("INSERT statement succeeded");
        }

        private function insertError(event:SQLErrorEvent):void
        {
            trace("Error message:", event.error.message);
            trace("Details:", event.error.details);
        }
    ]]>
</mx:Script>
</mx:WindowedApplication>
```

Следующий код вставляет строку данных в уже существующую таблицу «employees» с использованием режима синхронного выполнения. Заметьте, что данный код предполагает наличие инициализированного и подключенного к базе данных экземпляра SQLConnection с именем `conn`. Он также предполагает, что таблица «employees» уже создана.

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var insertStmt:SQLStatement = new SQLStatement();
insertStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "INSERT INTO employees (firstName, lastName, salary) " +
    "VALUES ('Bob', 'Smith', 8000)";
insertStmt.text = sql;

try
{
    // execute the statement
    insertStmt.execute();

    trace("INSERT statement succeeded");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var insertStmt:SQLStatement = new SQLStatement();
                insertStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "INSERT INTO employees (firstName, lastName, salary) " +
                    "VALUES ('Bob', 'Smith', 8000)";
                insertStmt.text = sql;

                try
                {
                    // execute the statement
                    insertStmt.execute();
                    trace("INSERT statement succeeded");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

## Извлечение созданного базой данных первичного ключа вставленной строки

**Adobe AIR 1.0 и более поздних версий**

Часто после вставки строки данных в таблицу код требует первичный ключ, созданный базой данных, или значение идентификатора строки для вновь вставленной строки. Например, при вставке строки в одну таблицу может потребоваться вставить строки в связанную таблицу. В таком случае потребуется вставить значение первичного ключа в качестве внешнего ключа в связанную таблицу. Первичный ключ вставленной строки можно извлечь при помощи объекта SQLResult, созданного при выполнении инструкции. Это тот же объект, что используется для доступа к данным результатов после выполнения инструкции `SELECT`. Как и в случае с любой другой инструкцией SQL, по завершении выполнения инструкции `INSERT` среда выполнения создает экземпляр SQLResult. Доступ к экземпляру SQLResult осуществляется путем вызова метода `getResult()` объекта SQLStatement, если используется прослушиватель событий или режим синхронного

выполнения. Наоборот, если используется режим асинхронного выполнения и экземпляр Responder передается вызову метода `execute()`, экземпляр SQLResult передается в качестве аргумента функции обработчика результатов. В любом случае экземпляр SQLResult имеет свойство, `lastInsertRowID`, которое содержит идентификатор последней вставленной строки, если выполненная инструкция SQL является инструкцией `INSERT`.

Следующий пример иллюстрирует доступ к первичному ключу вставленной строки в режиме асинхронного выполнения:

```
insertStmt.text = "INSERT INTO ...";

insertStmt.addEventListener(SQLEvent.RESULT, resultHandler);

insertStmt.execute();

function resultHandler(event:SQLEvent):void
{
    // get the primary key
    var result:SQLResult = insertStmt.getResult();

    var primaryKey:Number = result.lastInsertRowID;
    // do something with the primary key
}
```

Следующий пример иллюстрирует доступ к первичному ключу вставленной строки в режиме синхронного выполнения:

```
insertStmt.text = "INSERT INTO ...";

try
{
    insertStmt.execute();

    // get the primary key
    var result:SQLResult = insertStmt.getResult();

    var primaryKey:Number = result.lastInsertRowID;
    // do something with the primary key
}
catch (error:SQLError)
{
    // respond to the error
}
```

Заметьте, что идентификатор строки может представлять собой значение столбца, обозначенного как столбец первичного ключа в определении таблицы, а может и не представлять собой это значение. Это определяется в соответствии со следующими правилами:

- Если в определении таблицы задан столбец первичного ключа, который относится к типу данных `INTEGER`, свойство `lastInsertRowID` содержит значение, которое было вставлено в эту строку (или значение, созданное средой выполнения, если это столбец `AUTOINCREMENT`).
- Если в определении таблицы задано несколько столбцов первичного ключа (составной ключ) или один столбец первичного ключа с типом данных, отличным от типа `INTEGER`, база данных незаметно создает целочисленное значение идентификатора строки. Это созданное значение является значением свойства `lastInsertRowID`.

- Это значение всегда является идентификатором последней вставленной строки. Если инструкция `INSERT` запускает триггер, который вставляет строку, свойство `lastInsertRowID` содержит идентификатор последней вставленной триггером строки, а не строки, созданной инструкцией `INSERT`.

Следовательно, если вы хотите иметь явно определенный столбец первичного ключа, значение которого должно быть доступно после выполнения команды `INSERT` через свойство `SQLResult.lastInsertRowID`, столбец должен быть определен как `INTEGER PRIMARY KEY`. Даже если таблица не содержит столбец `INTEGER PRIMARY KEY`, можно также использовать созданный базой данных идентификатор строки в качестве первичного ключа таблицы для определения взаимосвязи со связанными таблицами. Значение столбца идентификатора строки доступно в любой инструкции SQL через одно из специальных имен столбцов `ROWID`, `_ROWID_` или `OID`. В связанной таблице можно создать столбец внешнего ключа и использовать значение идентификатора строки в качестве значения столбца внешнего ключа, как в случае с явно определенным столбцом `INTEGER PRIMARY KEY`. В этом случае при использовании произвольного первичного ключа вместо естественного ключа, и если вы не против создания средой выполнения значения первичного ключа, нет большой разницы в использовании столбца `INTEGER PRIMARY KEY` или идентификатора строки, созданного системой, в качестве первичного ключа таблицы для определения отношений внешнего ключа между двумя таблицами.

Дополнительные сведения о первичных ключах и генерируемых идентификаторах строк см. в разделе ««Поддержка SQL в локальных базах данных» на странице 1194».

## Изменение или удаление данных

**Adobe AIR 1.0 и более поздних версий**

Процесс выполнения других операций над данными идентичен процессу выполнения инструкции SQL `SELECT` или `INSERT`, описанных в разделе «Работа с инструкциями SQL» на странице 784. Просто присвойте другую инструкцию SQL свойству `text` экземпляра SQLStatement:

- Для изменения существующих в таблице данных используйте инструкцию `UPDATE`.
- Для удаления одной или нескольких строк данных из таблицы используйте инструкцию `DELETE`.

Описания этих инструкций см. в разделе ««Поддержка SQL в локальных базах данных» на странице 1194».

## Работа с несколькими базами данных

**Adobe AIR 1.0 и более поздних версий**

При помощи метода `SQLConnection.attach()` можно открыть подключение к дополнительной базе данных, применив его к экземпляру SQLConnection, который уже имеет открытую базу данных. Добавленной базе данных можно присвоить имя при помощи параметра name в вызове метода `attach()`. При создании инструкций для выполнения операций над этой базой данных можно использовать это имя в префиксе (в форме `database-name.table-name`) для квалификации имен таблиц в инструкциях SQL, показывая среде выполнения, что эту таблицу можно найти в указанной базе данных.

Можно выполнить одну инструкцию SQL, в которую включены таблицы из нескольких баз данных, подключенных к одному и тому же экземпляру SQLConnection. Если транзакция создана с использованием экземпляра SQLConnection, эта транзакция применяется ко всем инструкциям SQL, выполняемым с использованием этого экземпляра SQLConnection. Это правило действует независимо от того, в отношении какой из добавленных баз данных выполняется инструкция.

Вместо этого можно создать несколько экземпляров SQLConnection в приложении, каждый из которых будет подключен к одной или к нескольким базам данных. Однако при использовании нескольких подключений к одной и той же базе данных следует иметь в виду, что транзакция базы данных не является общей для экземпляров SQLConnection. Следовательно, при подключении к одной базе данных нескольких экземпляров SQLConnection нельзя рассчитывать на то, что изменения данных обоих подключений будут применены необходимым образом. Например, если две инструкции `UPDATE` или `DELETE` выполняются в отношении одной базы данных с использованием различных экземпляров SQLConnection, и во время первой операции возникает ошибка приложения, данные базы данных могут остаться в необратимом промежуточном состоянии, что может нарушить целостность базы данных (и, следовательно, приложения).

## Обработка ошибок базы данных

**Adobe AIR 1.0 и более поздних версий**

В общем, обработка ошибок базы данных аналогична обработке других ошибок времени выполнения. Необходимо написать код, готовый к возникновению ошибок, и реагировать на ошибки, а не предоставлять это дело среде выполнения. В общем, возможные ошибки базы данных можно разделить на три категории: ошибки подключения, ошибки синтаксиса SQL и ошибки ограничений.

### Ошибки подключения

**Adobe AIR 1.0 и более поздних версий**

Большинство ошибок базы данных — это ошибки подключения, которые могут возникнуть во время любой операции. Хотя существуют стратегии для предотвращения возникновения ошибок подключения, вряд ли существует простой способ без последствий восстановить работу после ошибки подключения, если база данных является важной частью приложения.

Большинство ошибок подключения относятся к взаимодействию среды выполнения с операционной системой, файловой системой и файлом базы данных. Например, ошибка подключения возникает, если у пользователя отсутствуют полномочия для создания файла базы данных в определенном местоположении файловой системы. Следующие стратегии помогают избежать ошибок подключения:

**Используйте файлы баз данных для конкретных пользователей.** Вместо использования одного файла базы данных для всех пользователей приложения на одном компьютере предоставьте каждому пользователю свой файл базы данных. Файл должен находиться в каталоге, связанном с учетной записью пользователя. Например, это может быть каталог хранилища приложения, папка документов пользователя, рабочий стол пользователя и т. д.

**Учитывайте различные типы пользователей.** Проверьте работу приложения с различными типами учетных записей пользователя и в различных операционных системах. Не предполагайте, что пользователь имеет права администратора для доступа к компьютеру. Также не предполагайте, что тот, кто установил приложение, является пользователем, выполняющим приложение.

**Учитывайте различные местоположения файлов.** Если разрешить пользователю указывать место сохранения файла базы данных или открывать файл по выбору, продумайте возможные местоположения файлов. Кроме того, продумайте ограничения для мест, куда пользователи могут сохранять файлы баз данных (или откуда они могут их открывать). Например, можно разрешить пользователям открывать только те файлы, которые находятся в пределах хранилищ их учетных записей.

Если возникает ошибка подключения, скорее всего это связано с первой попыткой создать или открыть базу данных. Это значит, что пользователь не может выполнять в приложении никакие операции, связанные с базой данных. Единственным методом устранения ошибок, связанных с режимом «только для чтения» или с

недостаточными полномочиями, является копирование файла базы данных в другое местоположение. Приложение может копировать файл базы данных в другое местоположение, которое позволяет пользователю создавать файлы и выполнять запись в них.

## Синтаксические ошибки

**Adobe AIR 1.0 и более поздних версий**

Синтаксическая ошибка возникает при неправильном построении инструкции SQL и попытке приложения выполнить эту инструкцию. Поскольку инструкции SQL локальной базы данных представляют собой строки, проверка синтаксиса SQL во время компиляции невозможна. Синтаксис инструкций SQL можно проверить, только выполнив их. Используйте следующие стратегии для предотвращения возникновения ошибок синтаксиса SQL:

**Тщательно тестируйте все инструкции SQL** По возможности во время разработки приложения тестируйте каждую инструкцию SQL отдельно перед их вставкой в качестве текста инструкции в код приложения. Кроме того, создайте набор тестов для проверки любых возможных вариаций кода.

**Используйте параметры инструкций и избегайте вставки (динамической генерации) SQL** Использование параметров вместо динамически создаваемых инструкций SQL означает, что при каждом выполнении инструкции SQL используется один и тот же текст инструкции. Это значительно облегчает тестирование инструкций и ограничивает возможные вариации. Если необходимо динамическое создание инструкций SQL, сократите до минимума динамические части инструкции. Кроме того, тщательно проверяйте введенные пользователем данные во избежание синтаксических ошибок.

Для устранения синтаксической ошибки приложение должно иметь сложную логику для проверки инструкции SQL и исправления ее синтаксиса. Следуя вышеназванным рекомендациям по предотвращению синтаксических ошибок, код может обнаружить любой потенциальный источник ошибок синтаксиса SQL (например, ввод данных пользователем в инструкцию). Предоставьте пользователям рекомендации по устранению синтаксических ошибок. Укажите на места, которые нужно исправить, для правильного выполнения инструкции.

## Ошибки ограничений

**Adobe AIR 1.0 и более поздних версий**

Ошибки ограничений возникают при попытке инструкции `INSERT` или `UPDATE` добавить данные в столбец. Ошибка возникает, если новые данные нарушают одно из установленных для таблицы или столбца ограничений. К возможным ограничениям относятся следующие:

**Ограничение unique** Указывает на запрет использования одинаковых значений в одном столбце для всех строк таблицы. Если для нескольких столбцов установлено ограничение уникальности, то в этих столбцах не должно встречаться повторяющееся сочетание значений. Другими словами, в рамках указанного уникального столбца или нескольких уникальных столбцов каждая строка должна отличаться от другой.

**Ограничение primary key** В отношении данных, которые разрешены или запрещены ограничением, ограничение primary key аналогично ограничению unique.

**Ограничение not null** Указывает на то, что отдельный столбец не может содержать значение `NULL` и, следовательно, в каждой строке этого столбца должно быть значение.

**Ограничение check** Позволяет задать произвольное ограничение для одной или нескольких таблиц. Типичным ограничением check является правило, согласно которому значение столбца должно находиться в определенных пределах (например, числовое значение столбца должно быть больше 0). Другим часто

использующимся типом ограничения check является указание отношений между значениями столбцов (например, значение столбца должно отличаться от значения другого столбца той же строки).

**Ограничение data type (родственности столбцов)** Среда выполнения определяет тип данных для значений столбцов, и при попытке сохранить в столбце значение неверного типа возникает ошибка. Однако во многих случаях значения преобразуются в соответствии с объявленным для столбца типом данных. См. раздел «Работа с типами данных базы данных» на странице 808 для получения дополнительной информации.

Среда выполнения не налагает ограничения на значения внешнего ключа. Другими словами, значения внешнего ключа не обязательно должны соответствовать существующему значению первичного ключа.

Помимо предустановленных типов ограничений механизм SQL среды выполнения поддерживает использование триггеров. Триггер подобен обработчику событий. Он представляет собой предустановленный набор инструкций, которые выполняются при наступлении какого-либо события. Например, можно определить триггер, который запускается при вставке данных в определенную таблицу или при удалении данных из нее. Одним из возможных применений триггера является проверка изменений данных. Если заданные условия не выполняются, возникает ошибка. Следовательно, триггер используется в тех же целях, что и ограничение, а стратегии по предотвращению и устранению ошибок ограничений также применимы к ошибкам, сгенерированным триггерами. Однако идентификаторы ошибок триггеров отличаются от идентификаторов ошибок ограничений.

Набор ограничений, применимых к определенной таблице, определяется при разработке приложения. Тщательное проектирование ограничений при разработке приложения облегчает предотвращение и устранение ошибок ограничений. Однако ошибки ограничений сложно предвидеть и устранять систематически. Ошибки ограничений трудно предвидеть, так как они не проявляются до добавления данных приложения. Ошибки ограничений возникают в связи с добавлением данных в базу данных после ее создания. Они часто являются следствием взаимосвязи между новыми и уже существующими данными базы данных. Следующие стратегии помогут избежать многих ошибок ограничений:

**Тщательно продумывайте структуру базы данных и ограничения** Целью ограничений является выполнение правил приложения и защита целостности данных в базе данных. При проектировании приложения продумайте структуру базы данных для поддержки приложения. В частности, определите правила для данных, например, когда требуются определенные значения, возможны ли значения по умолчанию, допускаются ли повторяющиеся значения и т. д. Эти правила помогут определить ограничения для базы данных.

**Указывайте имена столбцов явным образом** Инструкцию `INSERT` можно написать без явного указания столбцов, в которые нужно вставить значения, но такой подход представляет собой неоправданный риск. Явное именование столбцов для вставки данных разрешает автоматическое создание значений, наличие в столбцах значений по умолчанию и значений типа `NULL`. Кроме того, таким образом вы обеспечиваете вставку явных значений в столбцы `NOT NULL`.

**Используйте значения по умолчанию** При установке ограничения `NOT NULL` для столбца по возможности указывайте значение по умолчанию в определении этого столбца. Код приложения также может предоставить значения по умолчанию. Например, код может проверить, имеет ли переменная String значение `null`, и присвоить этой переменной значение перед ее использованием для задания значения параметра инструкции.

**Проверяйте введенные пользователем данные** Своевременно проверяйте введенные пользователем данные, чтобы убедиться, что они входят в пределы, заданные ограничением, особенно в случае с ограничениями `NOT NULL` и `CHECK`. Конечно, ограничение `UNIQUE` труднее всех поддается проверке, так как для этого потребуется выполнить инструкцию `SELECT`, чтобы определить уникальность данных.

**Используйте триггеры** Можно написать триггер, который будет проверять (и, возможно, заменять) вставленные данные и совершать другие действия для исправления недопустимых данных. Такая проверка и коррекция позволяет избежать возникновения ошибок ограничений.

Во многих случаях ошибки ограничений наиболее сложно предотвратить, чем другие типы ошибок. К счастью, существует несколько стратегий по устранению ошибок ограничений, которые не влияют на работу приложения:

**Используйте алгоритмы разрешения конфликтов** При установке ограничения для столбца и создании инструкции `INSERT` или `UPDATE` можно задать алгоритм разрешения конфликта. Алгоритм разрешения конфликта задает действие, которое выполняет база данных при нарушении ограничения. Механизм базы данных может выполнять несколько действий. Механизм базы данных может прервать выполнение отдельной инструкции или целой транзакции. Он может игнорировать ошибку. Он даже может удалить устаревшие данные и заменить их данными, которые пытается сохранить код.

Дополнительные сведения см. в разделе «ON CONFLICT (алгоритмы конфликтов)» приложения ««Поддержка SQL в локальных базах данных» на странице 1194».

**Обеспечивайте обратную связь в целях коррекции** Набор ограничений, которые могут повлиять на конкретную команду SQL, можно определить заранее. Следовательно, можно предугадать ошибки ограничений, которые могут возникнуть при выполнении инструкции. Учитывая рекомендации, можно создать приложение, способное реагировать на ошибки ограничений. Предположим, в приложение входит форма для ввода данных о новых продуктах. Если в базе данных для столбца с наименованиями продуктов установлено ограничение `UNIQUE`, вставка в базу данных строки с новым продуктом может привести к ошибке ограничения. Следовательно, приложение должно предугадывать ошибки ограничения. При возникновении ошибки приложение уведомляет пользователя о том, что продукт с таким наименованием уже существует и что необходимо выбрать другое наименование. Другой возможной реакцией является предоставление пользователю возможности просмотреть информацию о другом продукте с тем же наименованием.

## Работа с типами данных базы данных

**Adobe AIR 1.0 и более поздних версий**

При создании таблицы в базе данных инструкция SQL, создающая таблицу, задает родственность, или тип данных для каждого столбца таблицы. Явное объявление родственности столбцов в инструкции SQL `CREATE TABLE` не обязательно, но желательно.

Как правило, любой объект, сохраняемый в базу данных при помощи инструкции `INSERT`, возвращается в виде экземпляра того же типа данных при выполнении инструкции `SELECT`. Однако тип данных извлеченного значения может отличаться в зависимости от родственности столбца базы данных, в котором хранится это значение. Если тип данных хранящегося в столбце значения не соответствует родственности столбца, база данных пытается преобразовать это значение в соответствии с родственностью столбца. Например, если для столбца базы данных объявлена родственность `NUMERIC`, база данных пытается преобразовать вставляемые данные в числовой класс хранилища (`INTEGER` или `REAL`) перед сохранением данных. Если данные невозможно преобразовать, база данных выдает ошибку. Согласно этому правилу, если строка «12345» вставляется в столбец `NUMERIC`, база данных автоматически преобразует ее в целочисленное значение 12345 перед сохранением. При извлечении этого значения с использованием инструкции `SELECT` оно возвращается как экземпляр числового типа данных (Number), а не экземпляр String.

Следуя двум правилам, можно избежать нежелательного преобразования типов данных. Во-первых, задавайте для каждого столбца тип данных в соответствии с данными, которые будут в нем храниться. Во-вторых, вставляйте только те значения, тип данных которых соответствует заданной родственности. Следование этим двум правилам имеет два преимущества. При вставке данных не происходит их неожиданного преобразования (при этом, возможно, теряется их значение). Кроме того, при извлечении данных они возвращаются с исходным типом данных.

Дополнительные сведения о доступных типах сходства столбцов и использовании типов данных в инструкциях SQL см. в разделе ««Поддержка типов данных» на странице 1218».

# Использование синхронных и асинхронных операций базы данных

**Adobe AIR 1.0 и более поздних версий**

В предыдущих разделах говорилось о типичных операциях с базой данных, таких как извлечение, вставка, обновление и удаление данных, а также создание файла базы данных, таблиц и других объектов базы данных. Выполнение этих операций в синхронном и асинхронном режимах было проиллюстрировано примерами.

Напомним, что в режиме асинхронного выполнения вы предписываете механизму базы данных выполнить операцию. Механизм базы данных работает в фоновом режиме, а приложение продолжает выполняться. По завершении операции механизм базы данных отправляет соответствующее событие в целях уведомления о таком завершении. Основным преимуществом асинхронного выполнения является то, что среда выполнения выполняет операции базы данных в фоновом режиме, тогда как код главного приложения продолжает выполняться. Это особенно важно, когда выполнение операции требует значительного количества времени.

В синхронном режиме, напротив, операции не выполняются в фоновом режиме. Вы предписываете механизму базы данных выполнить операцию. На этом этапе код приостанавливает выполнение на время работы механизма базы данных. По завершении операции выполнение продолжается со следующей строки кода.

При использовании одного подключения базы данных невозможно выполнять одни операции или инструкции синхронно, а другие асинхронно. Задание синхронного или асинхронного режима выполнения для SQLConnection осуществляется при открытии подключения к базе данных. При вызове `SQLConnection.open()` подключение использует синхронный режим выполнения, а при вызове `SQLConnection.openAsync()` используется асинхронный режим выполнения. Как только экземпляр SQLConnection подключается к базе данных при помощи метода `open()` или `openAsync()`, для него устанавливается синхронный или асинхронный режим выполнения.

## Использование синхронных операций базы данных

**Adobe AIR 1.0 и более поздних версий**

Код, используемый для выполнения операций и реагирования на них при синхронном выполнении, мало отличается от кода, используемого при режиме асинхронного выполнения. Существует два ключевых отличия одного подхода от второго. Первое отличие заключается в выполнении операции, которая зависит от другой операции (например, строки результатов, полученные посредством инструкции `SELECT`, или первичный ключ строки, добавленной при помощи инструкции `INSERT`). Второе отличие связано с обработкой ошибок.

## Написание кода для синхронных операций

**Adobe AIR 1.0 и более поздних версий**

Ключевым отличием синхронного выполнения от асинхронного является то, что код для синхронного режима представляет собой единую последовательность шагов. В асинхронном режиме, наоборот, регистрируются прослушиватели событий и операции часто распределяются по методам прослушивателей. Когда база данных подключена в синхронном режиме, можно выполнить несколько операций базы данных подряд в одном блоке кода. Следующий пример иллюстрирует такой подход:

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, OpenMode.UPDATE);

// start a transaction
conn.begin();

// add the customer record to the database
var insertCustomer:SQLStatement = new SQLStatement();
insertCustomer.sqlConnection = conn;
insertCustomer.text =
    "INSERT INTO customers (firstName, lastName) " +
    "VALUES ('Bob', 'Jones')";
insertCustomer.execute();

var customerId:Number = insertCustomer.getResult().lastInsertRowID;

// add a related phone number record for the customer
var insertPhoneNumber:SQLStatement = new SQLStatement();
insertPhoneNumber.sqlConnection = conn;
insertPhoneNumber.text =
    "INSERT INTO customerPhoneNumbers (customerId, number) " +
    "VALUES (:customerId, '800-555-1234')";
insertPhoneNumber.parameters[":customerId"] = customerId;
insertPhoneNumber.execute();

// commit the transaction
conn.commit();
```

Как вы видите, одни и те же методы вызываются для выполнения операций базы данных независимо от использования синхронного или асинхронного режима. Основными отличиями одного подхода от другого является выполнение операции, которая зависит от другой операции, и обработка ошибок.

## Выполнение операции, зависящей от другой операции

**Adobe AIR 1.0 и более поздних версий**

В режиме синхронного выполнения нет необходимости составлять код, прослушивающий события, чтобы определить завершение операции. Вместо этого предполагается, что при успешном завершении операции в одной строке кода начинает выполняться операция в следующей строке. Следовательно, чтобы выполнить операцию, зависящую от успешного выполнения предыдущей операции, нужно написать зависимый код непосредственно после операции, от которой он зависит. Например, код приложения, включающий в себя начало транзакции, выполнение инструкции `INSERT`, получение первичного ключа вставленной строки, вставку первичного ключа в строку другой таблицы и фиксацию транзакции, может представлять собой последовательность инструкций. Следующий пример иллюстрирует эти операции:

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, SQLMode.UPDATE);

// start a transaction
conn.begin();

// add the customer record to the database
var insertCustomer:SQLStatement = new SQLStatement();
insertCustomer.sqlConnection = conn;
insertCustomer.text =
    "INSERT INTO customers (firstName, lastName) " +
    "VALUES ('Bob', 'Jones')";
insertCustomer.execute();

var customerId:Number = insertCustomer.getResult().lastInsertRowID;

// add a related phone number record for the customer
var insertPhoneNumber:SQLStatement = new SQLStatement();
insertPhoneNumber.sqlConnection = conn;
insertPhoneNumber.text =
    "INSERT INTO customerPhoneNumbers (customerId, number) " +
    "VALUES (:customerId, '800-555-1234')";
insertPhoneNumber.parameters[":customerId"] = customerId;
insertPhoneNumber.execute();

// commit the transaction
conn.commit();
```

## Обработка ошибок в синхронном режиме

**Adobe AIR 1.0 и более поздних версий**

В режиме синхронного выполнения нет необходимости прослушивать события ошибок для определения сбоя операции. Вместо этого нужно заключить код, который может привести к возникновению ошибок, в блоки кода `try..catch..finally`. Код, выдающий ошибку, заключается в блок `try`. Действия, выполняемые в ответ на каждый тип ошибки, заключаются в отдельный блок `catch`. Код, который выполняется независимо от успешного или неудачного выполнения операции (например, закрытие ненужного подключения к базе данных), заключается в блок `finally`. Следующий пример иллюстрирует использование блоков `try..catch..finally` для устранения ошибок. Он построен на предыдущем примере с добавлением кода обработки ошибок:

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, SQLMode.UPDATE);

// start a transaction
conn.begin();

try
{
    // add the customer record to the database
    var insertCustomer:SQLStatement = new SQLStatement();
    insertCustomer.sqlConnection = conn;
    insertCustomer.text =
        "INSERT INTO customers (firstName, lastName)" +
        "VALUES ('Bob', 'Jones')";

    insertCustomer.execute();

    var customerId:Number = insertCustomer.getResult().lastInsertRowID;

    // add a related phone number record for the customer
    var insertPhoneNumber:SQLStatement = new SQLStatement();
    insertPhoneNumber.sqlConnection = conn;
    insertPhoneNumber.text =
        "INSERT INTO customerPhoneNumbers (customerId, number)" +
        "VALUES (:customerId, '800-555-1234')";
    insertPhoneNumber.parameters[":customerId"] = customerId;

    insertPhoneNumber.execute();

    // if we've gotten to this point without errors, commit the transaction
    conn.commit();
}
catch (error:SQLError)
{
    // rollback the transaction
    conn.rollback();
}
```

## Основные сведения о модели асинхронного выполнения

**Adobe AIR 1.0 и более поздних версий**

Многие предполагают, что в режиме асинхронного выполнения нельзя начать выполнение экземпляра SQLStatement во время выполнения другого экземпляра SQLStatement в отношении одного и того же подключения к базе данных. На самом деле это предположение не совсем верное. Во время выполнения экземпляра SQLStatement невозможно изменить свойство `text` инструкции. Однако при использовании отдельного экземпляра SQLStatement для каждой инструкции SQL, которую нужно выполнить, можно вызвать метод `execute()` экземпляра SQLStatement во время выполнения другого экземпляра SQLStatement, что не приведет к ошибке.

При выполнении операций базы данных в асинхронном режиме каждое подключение базы данных (каждый экземпляр SQLConnection) имеет собственную внутреннюю очередь или список операций для выполнения. Среда выполняет операции последовательно в соответствии с порядком их добавления в очередь. При создании экземпляра SQLStatement и вызове его метода `execute()` операция выполнения этой инструкции добавляется в очередь подключения. Если нет выполняемых в данных момент операций для данного экземпляра SQLConnection, инструкция начинает выполняться в фоновом режиме. Предположим, в том же блоке кода создается другой экземпляр SQLStatement и вызывается его метод `execute()`. Операция выполнения второй инструкции ставится в очередь за первой инструкцией. Как только выполнение первой инструкции завершится, среда выполнения перейдет к следующей операции в очереди. Обработка последовательных операций в очереди происходит в фоновом режиме даже во время отправки события `result` первой операции в основной код приложения. Следующий код иллюстрирует такой подход:

```
// Using asynchronous execution mode
var stmt1:SQLStatement = new SQLStatement();
stmt1.sqlConnection = conn;

// ... Set statement text and parameters, and register event listeners ...

stmt1.execute();

// At this point stmt1's execute() operation is added to conn's execution queue.

var stmt2:SQLStatement = new SQLStatement();
stmt2.sqlConnection = conn;

// ... Set statement text and parameters, and register event listeners ...

stmt2.execute();

// At this point stmt2's execute() operation is added to conn's execution queue.
// When stmt1 finishes executing, stmt2 will immediately begin executing
// in the background.
```

Автоматическое выполнение базой данных последовательных инструкций в очереди имеет побочный эффект. Если выполнение инструкции зависит от результата выполнения предыдущей операции, невозможно добавить инструкцию в очередь (другими словами, невозможно вызвать ее метод `execute()`), пока не завершится выполнение первой операции. Это происходит потому, что после вызова метода `execute()` второй инструкции становится невозможным изменение свойств `text` или `parameters` этой инструкции. В таком случае необходимо дождаться получения события, указывающего на завершение первой операции, перед началом выполнения последующей операции. Например, если требуется выполнить инструкцию в контексте транзакции, выполнение инструкции зависит от выполнения операции по открытию транзакции.

После вызова метода `SQLConnection.begin()` для открытия транзакции необходимо дождаться отправки экземпляром SQLConnection события `begin`. Только после этого можно вызвать метод `execute()` экземпляра SQLStatement. В данном примере самым простым способом проверки правильного выполнения операций является создание метода, который регистрируется в качестве прослушивателя события `begin`. Код для вызова метода `SQLStatement.execute()` вставляется в метод прослушивателя.

# Использование шифрования с базами данных SQL

**Adobe AIR 1.5 и более поздних версий**

Все приложения Adobe AIR используют один и тот же механизм локальной базы данных. Следовательно, любое приложение AIR может подключаться к незашифрованному файлу данных для чтения и записи данных в него. Начиная с Adobe AIR версии 1.5 это приложение обеспечивает возможность создания зашифрованных файлов базы данных и подключения к ним. При использовании зашифрованной базы данных для подключения к ней приложение должно предоставить правильный ключ шифрования. Если предоставлен неверный ключ шифрования (или не предоставлен никакой ключ), приложение не подключится к базе данных. А значит приложение не сможет считывать данные из базы данных, а также записывать в нее данные или изменять их.

Для работы с зашифрованной базой данных должна быть создана такая база. Если зашифрованная база уже существует, можно установить соединение с ней. Также можно изменить ключ шифрования зашифрованной базы данных. За исключением процедур создания и подключения к зашифрованной базе данных, порядок работы с такой базой сходен с порядком работы с незашифрованной базой данных. В частности, выполнение инструкций SQL происходит точно также, независимо от того, зашифрована ли база данных или нет.

## Использование зашифрованной базы данных

**Adobe AIR 1.5 и более поздних версий**

Шифрование всегда полезно в тех случаях, когда требуется ограничить доступ к информации, хранимой в базе данных. Функция шифрования базы данных в Adobe AIR может использоваться в следующих целях. Далее приведены несколько примеров, когда может потребоваться использование зашифрованной базы данных:

- Доступный только для чтения кэш-буфер конфиденциальных прикладных данных, загруженных с сервера
- Локальное хранилище приложений, для закрытых данных, синхронизируемых с сервером (данные отправляются на сервер и загружаются с него)
- Зашифрованные файлы, которые используются в качестве формата файлов для документов, создаваемых и редактируемых приложением. Эти файлы могут быть конфиденциальными, предназначенными только для одного пользователя, а также могут быть предназначены для совместного использования всеми пользователями приложения.
- Любое другое использование локального хранилища данных, например, как это описано в разделе «Применение локальных бах данных SQL» на странице 768, при котором данные должны сохраняться конфиденциально, защищенные от других пользователей, имеющих доступ к компьютеру или файлам базы данных.

Понимание причины, зачем требуется использовать зашифрованную базу данных, помогает решить, как спроектировать приложение. В частности, это может повлиять на то, как приложение создает, получает и хранит ключ шифрования для базы данных. Дополнительные сведения об этих аспектах см. в разделе ««Предпосылки для использования зашифрованной базы данных» на странице 818».

Помимо зашифрованных баз данных существует альтернативный механизм конфиденциального хранения важных данных в зашифрованном локальном хранилище. Зашифрованное локальное хранилище позволяет хранить значение ByteArray с помощью ключа строки. Только то приложение AIR, которое хранит значение, может обращаться к нему, и только на том компьютере, где хранится это значение. Благодаря зашифрованному локальному хранилищу нет необходимости создавать собственный ключ шифрования. По этой причине зашифрованное локальное хранилище лучше всего подходит для удобного хранения отдельного значения или набора значений, которые легко могут быть зашифрованными в ByteArray. Зашифрованная база данных лучше всего подходит для больших наборов данных, где желательны структурированное хранение данных и выполнение запросов. Дополнительные сведения о зашифрованной локальной системе хранения данных см. в разделе ««Зашифрованная локальная система хранения данных» на странице 764».

## Создание зашифрованной базы данных

**Adobe AIR 1.5 и более поздних версий**

Для использования зашифрованной базы данных файл базы данных должен быть зашифрован при создании. Если база данных создана как незашифрованная, ее нельзя будет зашифровать в будущем. Точно так же нельзя будет расшифровать зашифрованную базу данных. Если необходимо, можно изменить ключ шифрования зашифрованной базы данных. Дополнительные сведения см. в разделе «Изменение ключа шифрования для базы данных» на странице 817. Если имеется существующая база данных, которая не зашифрована, и требуется использовать шифрование базы данных, можно создать новую зашифрованную базу данных и скопировать существующую структуру таблиц и данных в эту новую базу данных.

Создание зашифрованной базы данных практически полностью совпадает с процедурой создания незашифрованной базы данных, как это описано в разделе «Создание базы данных» на странице 773. Вначале создается экземпляр класса SQLConnection, который воспроизводит подключение к базе данных. База данных создается путем вызова метода `open()` или метода `openAsync()` объекта SQLConnection, указывающих базе данных местонахождение файла, который еще не существует. Единственным отличием при создании зашифрованной базы данных является то, что указывается значение для параметра `encryptionKey` (пятый параметр метода `open()` и шестой параметр метода `openAsync()`).

Допустимым значением параметра `encryptionKey` является объект ByteArray, содержащий ровно 16 байт.

Следующие примеры демонстрируют создание зашифрованной базы данных. Для простоты в этих примерах ключ шифрования жестко закодирован в коде приложения. Однако настоятельно не рекомендуется применять эту технологию, поскольку она не безопасна.

```
var conn:SQLConnection = new SQLConnection();

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

// Create an encrypted database in asynchronous mode
conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);

// Create an encrypted database in synchronous mode
conn.open(dbFile, SQLMode.CREATE, false, 1024, encryptionKey);
```

В качестве примера, демонстрирующего рекомендуемый способ создания ключа шифрования, см. раздел «Пример: создание и использование ключа шифрования» на странице 820.

## Подключение к зашифрованной базе данных

**Adobe AIR 1.5 и более поздних версий**

Подобно созданию зашифрованной базы данных, процедура открытия соединения с зашифрованной базой данных подобна процедуре открытия соединения с незашифрованной базой. Эта процедура подробно описана в разделе ««Подключение к базе данных» на странице 781». Метод `open()` используется для открытия соединения в синхронном режиме выполнения, а метод `openAsync()` — для открытия соединения в асинхронном режиме выполнения. Единственным отличием является то, что при открытии зашифрованной базы данных указывается допустимое значение для параметра `encryptionKey` (пятый параметр метода `open()` и шестой параметр метода `openAsync()`).

Если указанный ключ шифрования не верен, происходит ошибка. Для метода `open()` вызывается исключение SQLError. Для метода `openAsync()` объект SQLConnection отправляет событие SQLErrorEvent, свойство `error` которого содержит объект SQLError. В любом случае в объекте SQLError, созданном исключением, в свойстве `errorID` содержится значение 3138. Этот идентификатор ошибки соответствует сообщению об ошибке «Открытый файл не является файлом базы данных».

В следующем примере продемонстрировано открытие зашифрованной базы данных в асинхронном режиме выполнения. Для упрощения в этом примере ключ шифрования жестко закодирован в коде приложения. Однако настоятельно не рекомендуется применять эту технологию, поскольку она не безопасна.

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();
conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);
var dbFile:File = File.applicationStorageDirectory.resolvePath("DBSample.db");

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

conn.openAsync(dbFile, SQLMode.UPDATE, null, false, 1024, encryptionKey);

function openHandler(event:SQLEvent):void
{
    trace("the database opened successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    if (event.error.errorID == 3138)
    {
        trace("Incorrect encryption key");
    }
    else
    {
        trace("Error message:", event.error.message);
        trace("Details:", event.error.details);
    }
}
```

В следующем примере продемонстрировано открытие зашифрованной базы данных в синхронном режиме выполнения. Для упрощения в этом примере ключ шифрования жестко закодирован в коде приложения. Однако настоятельно не рекомендуется применять эту технологию, поскольку она не безопасна.

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();
var dbFile:File = File.applicationStorageDirectory.resolvePath("DBSample.db");

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

try
{
    conn.open(dbFile, SQLMode.UPDATE, false, 1024, encryptionKey);
    trace("the database was created successfully");
}
catch (error:SQLError)
{
    if (error.errorID == 3138)
    {
        trace("Incorrect encryption key");
    }
    else
    {
        trace("Error message:", error.message);
        trace("Details:", error.details);
    }
}
```

В качестве примера, демонстрирующего рекомендуемый способ создания ключа шифрования, см. раздел «Пример: создание и использование ключа шифрования» на странице 820.

## Изменение ключа шифрования для базы данных

**Adobe AIR 1.5 и более поздних версий**

Если база данных зашифрована, можно изменить ключ для ее шифрования в последующем. Для изменения ключа шифрования базы данных прежде всего откройте соединение к этой базе данных, создав экземпляр объекта SQLConnection и вызвав для него методы `open()` или `openAsync()`. Как только соединение с базой данных будет открыто, вызовите метод `reencrypt()`, передав ему в качестве аргумента новый ключ шифрования.

Так же, как и для большинства операций с базой данных, поведение метода `reencrypt()` зависит от того, будет ли соединение с базой данных использовать синхронный или асинхронный режимы выполнения. Если для соединения с базой данных используется метод `open()`, операция `reencrypt()` выполняется в синхронном режиме. Когда эта операция завершается, продолжается выполнение следующей строки программного кода:

```
var newKey:ByteArray = new ByteArray();
// ... generate the new key and store it in newKey
conn.reencrypt(newKey);
```

Однако, если соединение с базой данных открыто с помощью метода `openAsync()`, операция `reencrypt()` выполняется в асинхронном режиме. Вызов `reencrypt()` начинает процесс перешифрования. Когда эта операция завершается, объект SQLConnection отправляет событие `reencrypt`. Для определения того, не завершилось ли перешифрование, используется процесс прослушивания событий:

```
var newKey:ByteArray = new ByteArray();
// ... generate the new key and store it in newKey

conn.addEventListener(SQLEvent.REENCRYPT, reencryptHandler);

conn.reencrypt(newKey);

function reencryptHandler(event:SQLEvent):void
{
    // save the fact that the key changed
}
```

Операция `reencrypt()` выполняется в собственной отдельной транзакции. Если операция прерывается или завершается со сбоем (например, если приложение закрывается до завершения операции), выполняется откат этой транзакции. В этом случае исходный ключ шифрования по-прежнему является ключом шифрования базы данных.

Метод `reencrypt()` может использоваться для устранения шифрования из базы данных. Если передать значение `null` или ключ шифрования, который содержит не 16-байтовый ByteArray, в метод `reencrypt()`, произойдет ошибка.

## Предпосылки для использования зашифрованной базы данных

**Adobe AIR 1.5 и более поздних версий**

В разделе «Использование зашифрованной базы данных» на странице 814 представлено несколько случаев, в которых может потребоваться использование зашифрованной базы данных. Очевидно, что использование сценариев в различных приложениях (включая эти и другие сценарии) предполагает различные требования к конфиденциальности. Способ, которым проектируется использование шифрования в приложении, играет важную роль в управлении тем, насколько конфиденциальной будет база данных. Например, если зашифрованная база данных используется для конфиденциального хранения личных данных, защищая их даже от других пользователей данного компьютера, тогда каждому пользователю базы данных требуется свой собственный ключ шифрования. Для обеспечения максимального уровня безопасности приложение может создавать ключ шифрования из введенного пользователем пароля. Создание ключа безопасности на основе пароля обеспечивает уровень защиты, при котором даже если один пользователь попытается выдать себя за другого, он все равно не сможет обратиться к чужим данным. Альтернативным уровнем безопасности является ситуация, когда требуется, чтобы файлы данных были доступны на чтение другим пользователям этого приложения, но не других приложений. В этом случае каждой установленной копии приложения требуется доступ к совместно используемому ключу шифрования.

Можно спроектировать приложение, а в особенности технику создания ключа шифрования, в соответствии с уровнем безопасности, который требуется для данных приложения. В следующем списке представлены предпосылки для различных уровней конфиденциальности данных.

- Чтобы сделать базу данных доступной для любого пользователя, имеющего доступ к этому приложению на любом компьютере, используйте один ключ шифрования, который будет доступен для всех экземпляров приложения. Например, при первом запуске приложение может загружать общий ключ шифрования с сервера, используя защищенный протокол, например SSL. Затем приложение может сохранять ключ в зашифрованном локальном хранилище для последующего использования. В качестве альтернативы можно шифровать данные для каждого пользователя компьютера и синхронизировать их с удаленным хранилищем данных, например сервером, который обеспечивает перенос данных.
- Чтобы сделать базу данных доступной только для одного пользователя компьютера, создайте ключ шифрования из секретного слова пользователя (например, из пароля). В частности, для создания ключа не следует использовать какие-либо значения, связанные с данным компьютером (например, значения, хранимые в зашифрованном локальном хранилище). В качестве альтернативы можно шифровать данные для каждого пользователя компьютера и синхронизировать их с удаленным хранилищем данных, например сервером, который обеспечивает перенос данных.
- Чтобы сделать базу данных доступной только для одного пользователя с одного компьютера, создайте ключ шифрования из пароля. Пример этой методики см. в разделе «Пример: создание и использование ключа шифрования» на странице 820.

Далее изложены дополнительные соображения безопасности, которые важно учитывать при разработке приложения для работы с зашифрованной базой данных:

- Оценка защищенности системы всегда осуществляется по ее наиболее уязвимому месту. Если для создания ключа шифрования используется вводимый пользователем пароль, предусмотрите проверку ограничений на минимальную длину и сложность паролей. Короткие пароли, состоящие из простых символов, легко могут быть угаданы.
- Исходный код приложения AIR хранится на компьютере пользователя в виде плоского текста (для HTML-содержимого) или в виде легко декомпилируемого двоичного формата (для SWF-содержимого). Поскольку возможен доступ к исходному коду, необходимо учитывать два аспекта:
  - Никогда не кодируйте жестко ключ шифрования в исходный код
  - Всегда допускайте, что методика, используемая для создания ключа шифрования (например, генератор произвольных символов или определенный алгоритм хэширования) может быть легко воссоздана злоумышленником
- Шифрование в базах данных AIR использует стандарт AES в режиме CCM. Для обеспечения безопасности код шифрования требует комбинации вводимого пользователем ключа и солт-значения. Пример этой методики см. в разделе «Пример: создание и использование ключа шифрования» на странице 820.
- Если принято решение о шифровании базы данных, все файлы на диске, используемые ядром этой базы, шифруются. Но база данных временно удерживает некоторые данные в кэше памяти, чтобы повысить производительность чтения и записи при выполнении транзакций. Все находящиеся в памяти данные не зашифрованы. Если злоумышленник получит возможность доступа к памяти, используемой приложением AIR, например с помощью программы отладки, ему будет доступна та информация из базы данных, которая в данный момент открыта и не зашифрована.

## Пример: создание и использование ключа шифрования

**Adobe AIR 1.5 и более поздних версий**

Этот пример демонстрирует одну из методик создания ключа шифрования. Приложение спроектировано для обеспечения максимального уровня конфиденциальности и безопасности данных пользователя. Одним из важнейших аспектов защиты конфиденциальных данных является требование к пользователю вводить пароль при каждом подключении приложения к базе данных. Следовательно, как показано в этом примере, приложение, которому требуется этот уровень конфиденциальности, никогда не должно обычным способом сохранять ключ шифрования базы данных.

Приложение состоит из двух частей: класса ActionScript, создающего ключ шифрования (класс EncryptionKeyGenerator), и простейшего пользовательского интерфейса, демонстрирующего, как применять этот класс. Полный исходный код см. в разделе «Комплексный пример программного кода, создающего и использующего ключ шифрования» на странице 822.

### Использование класса EncryptionKeyGenerator для получения защищенного ключа шифрования

**Adobe AIR 1.5 и более поздних версий**

Для использования класса EncryptionKeyGenerator важно иметь четкое представление о принципах его работы. Дополнительные сведения о способах создания классом ключа шифрования см. в разделе ««Знакомство с классом EncryptionKeyGenerator» на странице 827».

Выполните следующие шаги, чтобы использовать класс EncryptionKeyGenerator в своем приложении:

1. Загрузите класс EncryptionKeyGenerator в качестве исходного кода или скомпилированного кода SWC. Класс EncryptionKeyGenerator включен в проект корневой библиотеки ActionScript 3.0 (as3corelib) с открытым кодом. Можно загрузить пакет as3corelib, включая исходный код и документацию к нему. SWC или файлы с исходным кодом можно загрузить со страницы проекта.
2. Поместите исходный код класса EncryptionKeyGenerator (или as3corelib SWC) в местоположение, где исходный код приложения сможет найти его.
3. Добавьте инструкцию `import` для класса EncryptionKeyGenerator в исходный код приложения.

   ```
   import com.adobe.air.crypto.EncryptionKeyGenerator;
   ```

4. До того места в коде, где создается база данных или открывается соединение с ней, добавьте код, создающий экземпляр EncryptionKeyGenerator, вызвав конструктор `EncryptionKeyGenerator()`.

   ```
   var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
   ```

5. Получение пароля от пользователя:

   ```
   var password:String = passwordInput.text;

   if (!keyGenerator.validateStrongPassword(password))
   {
       // display an error message
       return;
   }
   ```

Экземпляр EncryptionKeyGenerator использует этот пароль в качестве основы для ключа шифрования (показано в следующем шаге). Экземпляр EncryptionKeyGenerator проверяет пароль на соответствие определенным требованиям проверки надежности пароля. Если проверка не проходит, выдается сообщение об ошибке. В показанном демонстрационном коде можно заранее проверить пароль, вызвав метод `validateStrongPassword()` объекта EncryptionKeyGenerator. Таким способом можно определить, соответствует ли пароль минимальным требованиям надежности пароля и избежать ошибки.

**6** Создание ключа шифрования из пароля:

```
var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);
```

Метод `getEncryptionKey()` создает и возвращает ключ шифрования (16-байтовый ByteArray). Этот ключ шифрования можно использовать для создания новой зашифрованной базы данных или открытия существующей базы.

Метод `getEncryptionKey()` имеет один требуемый параметр, которым является пароль, получаемый в шаге 5.

***Примечание.*** *Для обеспечения максимального уровня безопасности и конфиденциальности данных приложение должно требовать от пользователя ввода пароля при каждом подключении приложения к базе данных. Никогда не сохраняйте пароль пользователя или ключ шифрования базы данных обычным способом. Из-за этого обычно возникают риски в обеспечении безопасности. Вместо этого, как продемонстрировано в данном примере, приложение должно использовать одну и ту же методику получения ключа шифрования из пароля как когда создает зашифрованную базу данных, так и когда соединяется с ней позже.*

Метод `getEncryptionKey()` также принимает второй (необязательный) параметр — `overrideSaltELSKey`. Класс EncryptionKeyGenerator создает произвольное значение (называемое *солт*-значением), которое используется как часть ключа шифрования. Чтобы можно было повторно создать ключ шифрования, солт-значение сохраняется в зашифрованном локальном хранилище (ELS) вашего приложения AIR. По умолчанию класс EncryptionKeyGenerator использует определенную строку в качестве ключа ELS. Хотя и маловероятно, но существует возможность, что этот ключ будет конфликтовать с другим ключом, используемым вашим приложением. Вместо использования ключа по умолчанию можно указать свой собственный ключ ELS. В этом случае укажите настраиваемый ключ, передав его вторым параметром `getEncryptionKey()`, как показано далее:

```
var customKey:String = "My custom ELS salt key";
var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password, customKey);
```

**7** Создание или открытие базы данных

С помощью ключа шифрования, возвращаемого методом `getEncryptionKey()`, в программном коде можно создать новую зашифрованную базу данных или попытаться открыть существующую зашифрованную базу данных. В любом случае можно использовать метод `open()` или `openAsync()` класса SQLConnection, как описано в разделах «Создание зашифрованной базы данных» на странице 815 и «Подключение к зашифрованной базе данных» на странице 816.

В этом примере приложение разработано так, чтобы открывать базу данных в асинхронном режиме выполнения. Программа устанавливает соответствующий процесс прослушивания событий и вызывает метод `openAsync()`, передавая ключ шифрования в качестве последнего аргумента:

```
conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, openError);

conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
```

В методах прослушивания программный код удаляет регистрации событий процессом прослушивания. Затем отображается сообщение о состоянии, показывающее, была ли база данных создана, открыта и не произошла ли ошибка. Особого упоминания из этих обработчиков событий заслуживает метод `openError()`. В этом методе инструкция `if` проверяет, существует ли база данных (подразумевает, что программа попыталась подключиться к существующей базе данных) и соответствует ли идентификатор ошибки константы `EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID`. Если оба этих условия выполняются, это скорее всего означает, что введенный пользователем пароль не верен. (Это также может означать, что указанный файл вообще не является файлом базы данных.) Далее показан код, который проверяет идентификатор ошибки:

```
if (!createNewDB && event.error.errorID ==
EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
{
    statusMsg.text = "Incorrect password!";
}
else
{
    statusMsg.text = "Error creating or opening database.";
}
```

Полный программный код с примером процессов прослушивания событий см. в разделе «Комплексный пример программного кода, создающего и использующего ключ шифрования» на странице 822.

## Комплексный пример программного кода, создающего и использующего ключ шифрования

**Adobe AIR 1.5 и более поздних версий**

Далее приведен полный код демонстрационного приложения «Создание и использование ключа шифрования». Этот код состоит из двух частей.

В примере используется класс EncryptionKeyGenerator для создания ключа шифрования из пароля. Класс EncryptionKeyGenerator включен в проект корневой библиотеки ActionScript 3.0 (as3corelib) с открытым кодом. Можно загрузить пакет as3corelib, включая исходный код и документацию к нему. SWC или файлы с исходным кодом можно загрузить со страницы проекта.

### Пример Flex

MXML-файл приложения содержит исходный код для простейшего приложения, создающего или открывающего соединение с зашифрованной базой данных:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="init();">
    <mx:Script>
        <![CDATA[
            import com.adobe.air.crypto.EncryptionKeyGenerator;

            private const dbFileName:String = "encryptedDatabase.db";

            private var dbFile:File;
            private var createNewDB:Boolean = true;
            private var conn:SQLConnection;

            // ------- Event handling -------

            private function init():void
            {
                conn = new SQLConnection();
                dbFile = File.applicationStorageDirectory.resolvePath(dbFileName);
                if (dbFile.exists)
                {
                    createNewDB = false;
                    instructions.text = "Enter your database password to open the encrypted
database.";
                    openButton.label = "Open Database";
                }
            }

            private function openConnection():void
            {
                var password:String = passwordInput.text;

                var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();

                if (password == null || password.length <= 0)
                {
                    statusMsg.text = "Please specify a password.";
                    return;
                }

                if (!keyGenerator.validateStrongPassword(password))
                {
                    statusMsg.text = "The password must be 8-32 characters long. It must
contain at least one lowercase letter, at least one uppercase letter, and at least one number
or symbol.";
                    return;
                }

                passwordInput.text = "";
                passwordInput.enabled = false;
                openButton.enabled = false;

                var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, openError);
```

```
                conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
            }

            private function openHandler(event:SQLEvent):void
            {
                conn.removeEventListener(SQLEvent.OPEN, openHandler);
                conn.removeEventListener(SQLErrorEvent.ERROR, openError);

                statusMsg.setStyle("color", 0x009900);
                if (createNewDB)
                {
                    statusMsg.text = "The encrypted database was created successfully.";
                }
                else
                {
                    statusMsg.text = "The encrypted database was opened successfully.";
                }
            }

            private function openError(event:SQLErrorEvent):void
            {
                conn.removeEventListener(SQLEvent.OPEN, openHandler);
                conn.removeEventListener(SQLErrorEvent.ERROR, openError);

                if (!createNewDB && event.error.errorID ==
EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
                {
                    statusMsg.text = "Incorrect password!";
                }
                else
                {
                    statusMsg.text = "Error creating or opening database.";
                }
            }
        ]]>
    </mx:Script>
    <mx:Text id="instructions" text="Enter a password to create an encrypted database. The next
time you open the application, you will need to re-enter the password to open the database
again." width="75%" height="65"/>
    <mx:HBox>
        <mx:TextInput id="passwordInput" displayAsPassword="true"/>
        <mx:Button id="openButton" label="Create Database" click="openConnection();"/>
    </mx:HBox>
    <mx:Text id="statusMsg" color="#990000" width="75%"/>
</mx:WindowedApplication>
```

### Пример Flash Professional

FLA-файл приложения содержит исходный код для простейшего приложения, создающего или открывающего соединение с зашифрованной базой данных. FLA-файл содержит четыре компонента, размещаемые в рабочей области:

| Имя экземпляра | Тип компонента | Описание |
|---|---|---|
| `instructions` | Label | Содержит инструкции для пользователя |
| `passwordInput` | TextInput | Поле ввода, куда пользователь вводит пароль |
| `openButton` | Button | Кнопка, которую нажимает пользователь после ввода пароля |
| `statusMsg` | Label | Отображает сообщения о состоянии (успешно или с ошибкой) |

Код для приложения определяется в ключевом кадре кадра 1 основной временной шкалы. Далее показан этот код для приложения:

```
import com.adobe.air.crypto.EncryptionKeyGenerator;

const dbFileName:String = "encryptedDatabase.db";

var dbFile:File;
var createNewDB:Boolean = true;
var conn:SQLConnection;

init();

// ------- Event handling -------

function init():void
{
    passwordInput.displayAsPassword = true;
    openButton.addEventListener(MouseEvent.CLICK, openConnection);
    statusMsg.setStyle("textFormat", new TextFormat(null, null, 0x990000));

    conn = new SQLConnection();
    dbFile = File.applicationStorageDirectory.resolvePath(dbFileName);

    if (dbFile.exists)
    {
        createNewDB = false;
        instructions.text = "Enter your database password to open the encrypted database.";
        openButton.label = "Open Database";
    }
    else
    {
        instructions.text = "Enter a password to create an encrypted database. The next time
you open the application, you will need to re-enter the password to open the database again.";
        openButton.label = "Create Database";
    }
}

function openConnection(event:MouseEvent):void
{
    var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();

    var password:String = passwordInput.text;

    if (password == null || password.length <= 0)
```

```
    {
        statusMsg.text = "Please specify a password.";
        return;
    }

    if (!keyGenerator.validateStrongPassword(password))
    {
        statusMsg.text = "The password must be 8-32 characters long. It must contain at least
one lowercase letter, at least one uppercase letter, and at least one number or symbol.";
        return;
    }

    passwordInput.text = "";
    passwordInput.enabled = false;
    openButton.enabled = false;

    var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);

    conn.addEventListener(SQLEvent.OPEN, openHandler);
    conn.addEventListener(SQLErrorEvent.ERROR, openError);

    conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
}

function openHandler(event:SQLEvent):void
{
    conn.removeEventListener(SQLEvent.OPEN, openHandler);
    conn.removeEventListener(SQLErrorEvent.ERROR, openError);

    statusMsg.setStyle("textFormat", new TextFormat(null, null, 0x009900));
    if (createNewDB)
    {
        statusMsg.text = "The encrypted database was created successfully.";
    }
    else
    {
        statusMsg.text = "The encrypted database was opened successfully.";
    }
}

function openError(event:SQLErrorEvent):void
{
    conn.removeEventListener(SQLEvent.OPEN, openHandler);
    conn.removeEventListener(SQLErrorEvent.ERROR, openError);

    if (!createNewDB && event.error.errorID ==
EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
    {
        statusMsg.text = "Incorrect password!";
    }
    else
    {
        statusMsg.text = "Error creating or opening database.";
    }
}
```

## Знакомство с классом EncryptionKeyGenerator

**Adobe AIR 1.5 и более поздних версий**

Необходимо понимать внутреннюю механику работы класса EncryptionKeyGenerator, чтобы использовать его для создания защищенных ключей шифрования для базы данных приложения. Процесс использования класса объясняется в разделе «Использование класса EncryptionKeyGenerator для получения защищенного ключа шифрования» на странице 820. Однако, возможно, покажется полезным ознакомиться с теми методиками, которые использует этот класс. Например, может потребоваться адаптация класса или реализация каких-либо из его методик в ситуациях, где необходим другой уровень конфиденциальности данных.

Класс EncryptionKeyGenerator включен в проект корневой библиотеки ActionScript 3.0 (as3corelib) с открытым кодом. Можно загрузить пакет as3corelib, включая исходный код и документацию. Исходный код можно также просмотреть на сервере проекта или загрузить его для большей наглядности объяснений.

Когда код создает экземпляр EncryptionKeyGenerator и вызывает его метод `getEncryptionKey()`, выполняется несколько шагов, обеспечивающих доступ к данным только уполномоченных на это пользователей. Процесс совпадает с процессом создания ключа шифрования из введенного пользователем пароля перед созданием базы данных, а также с пересозданием ключа шифрования для открытия базы данных.

### Получение и проверка надежного пароля

**Adobe AIR 1.5 и более поздних версий**

Когда код вызывает метод `getEncryptionKey()`, он передает пароль в качестве параметра. Этот пароль используется в качестве основы для ключа шифрования. Благодаря использованию информации, известной только пользователю, такая схема гарантирует, что только пользователь, знающий пароль, сможет обратиться к данным в базе данных. Даже если злоумышленнику удастся получить доступ к учетной записи пользователя на компьютере, он не сможет войти в базу данных, не зная пароля. В целях обеспечения максимальной безопасности приложение никогда не сохраняет этот пароль.

Код приложения создает экземпляр EncryptionKeyGenerator и вызывает его метод `getEncryptionKey()`, передавая введенный пользователем пароль в качестве аргумента (переменная `password` в этом примере):

```
var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);
```

Первым шагом, который выполняет класс EncryptionKeyGenerator при вызове метода `getEncryptionKey()`, является проверка введенного пользователем пароля на предмет его соответствия требованиям надежности пароля. Для класса EncryptionKeyGenerator необходимо использовать пароль, состоящий из 8-32 символов. Пароль должен содержать сочетание букв верхнего и нижнего регистров и хотя бы одной цифры или символа.

Регулярное выражение, проверяющее соответствие этому шаблону, определяется как константа с именем `STRONG_PASSWORD_PATTERN`:

```
private static const STRONG_PASSWORD_PATTERN:RegExp =
/(?=^.{8,32}$)((?=.*\d)|(?=.*\W+))(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$/;
```

Программный код, проверяющий пароль, размещается в методе `validateStrongPassword()` класса EncryptionKeyGenerator. Этот код имеет следующий вид:

```
public function vaidateStrongPassword(password:String):Boolean
{
    if (password == null || password.length <= 0)
    {
        return false;
    }

    return STRONG_PASSWORD_PATTERN.test(password))
}
```

При внутренней обработке метод `getEncryptionKey()` вызывает метод `validateStrongPassword()` класса EncryptionKeyGenerator и, если пароль недействителен, порождает исключение. Метод `validateStrongPassword()` — это общий метод, поэтому из программного кода можно выполнить проверку пароля без вызова метода `getEncryptionKey()`, чтобы избежать возникновения ошибки.

### Расширение пароля до 256 битов

**Adobe AIR 1.5 и более поздних версий**

Позднее в процессе обработки требуется, чтобы пароль был длиной 256 битов. Вместо того, чтобы требовать от каждого пользователя вводить пароль точно 256 битов (32 символа) длиной, в коде создается более длинный пароль путем повторения символов пароля.

Метод `getEncryptionKey()` вызывает метод `concatenatePassword()` для выполнения задачи создания длинного пароля.

```
var concatenatedPassword:String = concatenatePassword(password);
```

Следующий код предназначен для метода `concatenatePassword()`.

```
private function concatenatePassword(pwd:String):String
{
    var len:int = pwd.length;
    var targetLength:int = 32;

    if (len == targetLength)
    {
        return pwd;
    }

    var repetitions:int = Math.floor(targetLength / len);
    var excess:int = targetLength % len;

    var result:String = "";

    for (var i:uint = 0; i < repetitions; i++)
    {
        result += pwd;
    }

    result += pwd.substr(0, excess);

    return result;
}
```

Если длина пароля меньше 256 битов, код последовательно соединяет несколько экземпляров пароля, доводя длину до 256 битов. Если требуемую длину невозможно получить путем составления целого числа экземпляров пароля, последний экземпляр укорачивается, чтобы в результате получилось 256 битов.

### Создание или извлечение 256-разрядного солт-значения

**Adobe AIR 1.5 и более поздних версий**

Следующий шаг предназначен для получения 256-разрядного солт-значения, которое в последующих шагах соединяется паролем. Значение *солт* — это произвольное значение, которое добавляется или комбинируется с вводимым пользователем значением для получения пароля. Использование солт-значения с паролем гарантирует, что даже если пользователь выберет обычное слово или общеупотребимый термин в качестве пароля, комбинация пароль+солт, используемая системой, окажется произвольным значением. Эта произвольность помогает защититься от словарных атак, когда злоумышленник использует список слов в попытках угадать пароль. Создавая солт-значение и сохраняя его в зашифрованном локальном хранилище, выполняется привязка к учетной записи пользователя на компьютере, где находятся файлы базы данных.

Если приложение вызывает метод `getEncryptionKey()` в первый раз, программный код создает произвольное 256-разрядное солт-значение. В противном случае код загружает солт-значение из зашифрованного локального хранилища.

Солт-значение хранится в переменной с названием `salt`. Этот код определяет, не создано ли уже солт-значение, пытаясь загрузить его из зашифрованной локальной системы хранения.

```
var salt:ByteArray = EncryptedLocalStore.getItem(saltKey);
if (salt == null)
{
    salt = makeSalt();
    EncryptedLocalStore.setItem(saltKey, salt);
}
```

Если код создает новое солт-значение, метод `makeSalt()` создает 256-разрядное произвольное значение. Поскольку значение в конечном итоге хранится в зашифрованном локальном хранилище, оно создается как объект ByteArray. Метод `makeSalt()` использует метод `Math.random()` для произвольного создания значений. Метод `Math.random()` не может создавать 256 битов одновременно. Вместо этого в программном коде организован цикл, вызывающий метод `Math.random()` восемь раз. Каждый раз создается произвольное значение uint в интервале от 0 до 4294967295 (максимальное значение uint). Значение uint используется для удобства, поскольку в uint содержится ровно 32 бита. 256-разрядное значение создается путем записи восьми значений uint в ByteArray. Следующий код предназначен для метода `makeSalt()`:

```
private function makeSalt():ByteArray
{
    var result:ByteArray = new ByteArray;

    for (var i:uint = 0; i < 8; i++)
    {
        result.writeUnsignedInt(Math.round(Math.random() * uint.MAX_VALUE));
    }

    return result;
}
```

Когда код сохраняет солт-значение в зашифрованное локальное хранилище (ELS) или извлекает его из ELS, ему требуется ключ String, с которым сохраняется это солт-значение. Невозможно извлечь это солт-значение, не зная ключа. В этом случае ключ шифрования не может быть воссоздан для каждого открытия базы данных. По умолчанию EncryptionKeyGenerator использует предварительно определенный ключ ELS, определяемый константой `SALT_ELS_KEY`. Вместо использования ключа по умолчанию код приложения может также указать ключ ELS для использования при вызове метода `getEncryptionKey()`. Либо заданный по умолчанию, либо указанный приложением, ключ ELS хранится в переменной с именем `saltKey`. Эта переменная используется в вызовах `EncryptedLocalStore.setItem()` и `EncryptedLocalStore.getItem()`, как было показано ранее.

### Объединение 256-разрядного пароля и солт-значения с помощью оператора XOR

**Adobe AIR 1.5 и более поздних версий**

Теперь у кода есть 256-разрядный пароль и 256-разрядное солт-значение. Затем применяется побитовая операция XOR для объединения солт-значения и составного пароля в единое значение. В действительности эта методика позволяет создать 256-разрядный пароль, состоящий из целого спектра возможных символов. Это и происходит в действительности, несмотря на то, что в большинстве случаев фактически вводимый пароль состоит преимущественно из буквенно-цифровых символов. Подобное увеличение произвольности обеспечивает возможность расширения набора возможных паролей, не заставляя пользователя вводить сложные значения.

Результат операции XOR сохраняется в переменной `unhashedKey`. Фактический процесс выполнения побитовой операции XOR для двух значений происходит в методе `xorBytes()`:

```
var unhashedKey:ByteArray = xorBytes(concatenatedPassword, salt);
```

Побитовый оператор XOR ^) берет два значения uint и возвращает одно значение uint. (Значение uint содержит 32 бита.) Входные значения, передаваемые как аргументы методу `xorBytes()`, являются значениями String (пароль) и ByteArray (солт). Затем код приложения использует цикл, чтобы извлечь 32 бита из каждого вводимого значения и скомбинировать их с помощью оператора XOR.

```
private function xorBytes(passwordString:String, salt:ByteArray):ByteArray
{
    var result:ByteArray = new ByteArray();

    for (var i:uint = 0; i < 32; i += 4)
    {
        // ...
    }

    return result;
}
```

Внутри цикла первые 32 бита (4 байта) извлекаются из параметра `passwordString`. Эти биты извлекаются и преобразуются в uint (`o1`) с помощью двухфазного процесса. Вначале метод `charCodeAt()` получает цифровое значение для каждого символа. Затем это значение сдвигается в нужную позицию в uint с помощью оператора побитового сдвига влево (`<<`), после чего значение со сдвигом добавляется к `o1`. Например, первый символ (`i`) становится первыми 8 битами с помощью оператора побитового сдвига влево (`<<`); биты сдвигаются влево на 24 бита, получившееся значение назначается `o1`. Второй символ `(i + 1)` становится вторыми 8 битами путем сдвига влево на 16 битов и добавления результата к `o1`. Значения третьего и четвертого символов добавляются сходным образом.

```
        // ...

        // Extract 4 bytes from the password string and convert to a uint
        var o1:uint = passwordString.charCodeAt(i) << 24;
        o1 += passwordString.charCodeAt(i + 1) << 16;
        o1 += passwordString.charCodeAt(i + 2) << 8;
        o1 += passwordString.charCodeAt(i + 3);

        // ...
```

Переменная `o1` теперь содержит 32 бита из параметра `passwordString`. Следующие 32 бита извлекаются из параметра `salt` вызовом метода `readUnsignedInt()`. Эти 32 бита сохраняются в переменной uint с названием `o2`.

```
        // ...

        salt.position = i;
        var o2:uint = salt.readUnsignedInt();

        // ...
```

В завершение оба 32-разрядных (uint) значения объединяются с помощью оператора XOR, а результат записывается в значение ByteArray с названием `result`.

```
        // ...

        var xor:uint = o1 ^ o2;
        result.writeUnsignedInt(xor);
        // ...
```

После завершения цикла возвращается значение ByteArray, содержащее результат XOR.

```
        // ...
    }

    return result;
}
```

### Хэширование ключа

**Adobe AIR 1.5 и более поздних версий**

Как только составной пароль и солт-значение объединены, следующий шаг призван дополнительно обезопасить это значение путем хэширования с помощью алгоритма SHA-256. Хэширование значения означает создание дополнительного препятствия для злоумышленника при попытке восстановить это значение.

В этот момент у кода есть значение ByteArray с именем `unhashedKey`, содержащее составной пароль, объединенный с солт-значением. Проект корневой библиотеки ActionScript 3.0 (as3corelib) содержит класс SHA256 в пакете com.adobe.crypto. Метод `SHA256.hashBytes()` выполняет хэширование SHA-256 для ByteArray и возвращает значение String, содержащее 256-разрядный результат хэширования в виде шестнадцатеричного числа. Класс EncryptionKeyGenerator использует класс SHA256 для хэширования ключа.

```
var hashedKey:String = SHA256.hashBytes(unhashedKey);
```

### Извлечение ключа шифрования из хэша

**Adobe AIR 1.5 и более поздних версий**

Ключ шифрования должен быть значением ByteArray длиной ровно 16 байтов (128 битов). Результатом выполнения алгоритма хэширования SHA-256 всегда является 256-разрядное значение. Следовательно, финальным шагом является выбор 128 --битов из хэшированного результата для использования в качестве фактического ключа шифрования.

В классе EncryptionKeyGenerator код приложения уменьшает ключ до 128 битов, вызывая метод `generateEncryptionKey()`. Затем результат этого метода возвращается как результат метода `getEncryptionKey()`:

```
var encryptionKey:ByteArray = generateEncryptionKey(hashedKey);
return encryptionKey;
```

Совсем не обязательно использовать в качестве ключа шифрования первые 128 битов. Можно выбрать диапазон битов, начинающийся в некоторой произвольной точке, можно выбрать любой другой бит или использовать другие способы выбора битов. Важным является то, что программа выбирает 128 конкретных битов и именно эти 128 битов будут использоваться всегда.

В рассматриваемом случае метод `generateEncryptionKey()` использует в качестве ключа шифрования биты, начиная с 18-го байта. Как уже упоминалось ранее, класс SHA256 возвращает значение String, содержащее в 256-разрядном хэш-значении шестнадцатеричный номер. Отдельный блок из 128 битов содержит слишком много байтов для одновременного добавления к ByteArray. Поэтому программа использует цикл `for` для извлечения символов из шестнадцатеричного значения String, преобразует их в фактические цифровые значения и добавляет к ByteArray. Полученное в результате обработки алгоритмом SHA-256 значение String имеет длину 64 символа. Диапазон в 128 битов равен 32 символам в объекте String, каждый символ соответствует 4 битам. Наименьшее приращение данных, которое можно добавить к ByteArray, составляет 1 байт (8 битов), что эквивалентно двум символам в переменной `hash` объекта String. Поэтому счетчик цикла изменяется 0 до 31 (32 символа) с приращением 2 символа.

Внутри цикла программа вначале определяет начальную позицию для текущей пары символов. Поскольку требуемый диапазон начинается символом с позицией 17 в индексе (18-ый байт), то переменной `position` назначается текущее значение итератора (`i`) плюс 17. Код использует метод `substr()` объекта String для извлечения двух символов в текущей позиции. Эти символы хранятся в переменной `hex`. Затем программа использует метод `parseInt()` для преобразования переменной `hex` объекта String в десятеричное целое значение. Это значение сохраняется в целочисленной переменной `byte`. Наконец, программа добавляет значение из `byte` к переменной `result` объекта ByteArray, используя его метод `writeByte()`. Когда цикл завершается, переменная `result` объекта ByteArray содержит 16 байтов и уже готова к использованию в качестве ключа шифрования.

```
private function generateEncryptionKey(hash:String):ByteArray
{
    var result:ByteArray = new ByteArray();

    for (var i:uint = 0; i < 32; i += 2)
    {
        var position:uint = i + 17;
        var hex:String = hash.substr(position, 2);
        var byte:int = parseInt(hex, 16);
        result.writeByte(byte);
    }

    return result;
}
```

# Стратегии работы с базами данных SQL

**Adobe AIR 1.0 и более поздних версий**

Существует несколько способов получения приложением доступа к локальной базе данных SQL и работы с ней. Проект приложения может быть разным с точки зрения структуры кода приложения, последовательности и синхронизации выполнения операций и т. д. Выбранные методы влияют на простоту разработки приложения. Эти методы также влияют на то, насколько легко можно будет изменить приложение в будущем при его обновлении. Кроме того, они влияют на производительность приложения с точки зрения пользователя.

## Распространение заполненной базы данных

**Adobe AIR 1.0 и более поздних версий**

При использовании в приложении AIR локальной базы данных SQL приложение ожидает, что база данных будет иметь определенную структуру таблиц, столбцов и т. д. Некоторые приложения также ожидают, что файл базы данных будет заполнен определенными данными. Одним из способов обеспечения правильной структуры базы данных является создание базы данных в коде приложения. При загрузке приложения проверяется наличие файла базы данных в определенном местоположении. Если файл отсутствует, приложение выполняет ряд команд по созданию файла базы данных, структуры базы данных и заполнению таблиц исходными данными.

Код, создающий базу данных и ее таблицы, чаще всего является сложным. Как правило, он используется только один раз за время существования установленного приложения, но увеличивает размер приложения и усложняет его. Вместо создания базы данных, структуры и данных программным способом можно распространять предварительно заполненную данными базу данных вместе с приложением. Для распространения предварительно определенной базы данных вставьте файл базы данных в пакет AIR приложения.

Как и все файлы, входящие в пакет AIR, файл базы данных устанавливается в каталог приложения (каталог, представленный свойством `File.applicationDirectory`). Однако файлы в этом каталоге предназначены только для чтения. Используйте файл из пакета AIR в качестве «шаблона» базы данных. При первом запуске приложения пользователем скопируйте исходный файл базы данных в «Указание на каталог хранилища приложения» на странице 722 пользователя (или в другое местоположение) и используйте эту базу данных в приложении.

## Рекомендации по работе с локальными базами данных SQL

**Adobe AIR 1.0 и более поздних версий**

Далее приводится описание рекомендуемых методов, которые можно использовать для улучшения производительности, безопасности и обслуживания приложений при работе с локальными базами данных SQL.

### Предварительное создание подключений к базам данных

**Adobe AIR 1.0 и более поздних версий**

Даже если при первичной загрузке приложение не выполняет никаких инструкций, создайте объект SQLConnection и вызовите его метод `open()` или `openAsync()` заранее (например, после начального запуска приложения) во избежание задержек при выполнении инструкций. См. раздел ««Подключение к базе данных» на странице 781».

### Повторное использование подключений к базе данных

**Adobe AIR 1.0 и более поздних версий**

Если вы обращаетесь к определенной базе данных на протяжении всего времени работы приложения, рекомендуется сохранять ссылку на экземпляр SQLConnection и повторно использовать его в приложении, а не закрывать и снова открывать подключение. См. раздел ««Подключение к базе данных» на странице 781».

## Предпочтительное использование режима асинхронного выполнения

**Adobe AIR 1.0 и более поздних версий**

При написании кода доступа к данным возникает соблазн использовать синхронное, а не асинхронное выполнение операций, так как для синхронных операций требуется более короткий и менее сложный код. Однако, как было сказано в разделе «Использование синхронных и асинхронных операций базы данных» на странице 809, синхронные операции могут негативно сказаться на производительности, что очевидно для пользователей и омрачает их работу с приложением. Время, затраченное на выполнение операции, зависит от самой операции и особенно от объема обрабатываемых данных. Например, инструкция SQL `INSERT` , которая добавляет только одну строку в базу данных, требует меньше времени, чем инструкция `SELECT`, которая извлекает тысячи строк данных. Однако при использовании синхронного режима для выполнения нескольких операций эти операции объединяются. Даже если время, затрачиваемое каждой операцией, невелико, приложение «застывает» до тех пор, пока не будут выполнены все синхронные операции. В результате общее время выполнения нескольких операций может оказаться достаточным для остановки работы приложения.

Используйте асинхронные операции в качестве стандартного подхода, особенно с операциями, в которых задействовано большое количество строк. Существует метод, при помощи которого можно разделить на части обработку больших групп результатов, полученных выполнением инструкции `SELECT`. Этот метод описан в разделе «Извлечение результатов SELECT по частям» на странице 797. Однако этот метод можно применять только в режиме асинхронного выполнения. Используйте синхронные операции только в том случае, если в режиме асинхронного выполнения невозможно достичь той или иной функциональности, продумав при этом возможные проблемы с производительностью приложения, с которыми столкнутся пользователи. Асинхронное выполнение предполагает использование более сложного кода. Однако помните, что код потребуется написать только один раз, а пользователи будут работать с приложением регулярно, быстро или медленно.

Во многих случаях при использовании отдельного экземпляра SQLStatement для каждой инструкции SQL, которую нужно выполнить, можно поставить в очередь сразу несколько операций SQL, что делает асинхронный код подобным синхронному с точки зрения написания кода. Дополнительную информацию см. в разделе «Основные сведения о модели асинхронного выполнения» на странице 813.

## Используйте отдельные инструкции SQL и не изменяйте свойство экземпляра SQLStatement

**Adobe AIR 1.0 и более поздних версий**

Создавайте отдельный экземпляр SQLStatement для каждой инструкции SQL, которая выполняется в приложении несколько раз. Используйте этот экземпляр SQLStatement при каждом выполнении команды SQL. Предположим, вы создаете приложение, которое использует четыре разные операции SQL, выполняемые несколько раз. В таком случае создайте четыре экземпляра SQLStatement и вызовите метод `execute()` каждого экземпляра для его выполнения. Старайтесь не использовать один экземпляр SQLStatement для всех инструкций SQL, заново определяя свойство `text` каждый раз перед выполнением инструкции.

## Использование параметров инструкций

**Adobe AIR 1.0 и более поздних версий**

Используйте параметры SQLStatement — никогда не вставляйте введенные пользователем данные в текст инструкции. Использование параметров улучшает защиту приложения за счет предотвращения атак с внедрением SQL. Это позволяет использовать объекты в запросах (а не только литеральные значения SQL). Это также повышает эффективность выполнения инструкций, так как они могут использоваться многократно без необходимости повторной компиляции при каждом их выполнении. Дополнительные сведения см. в разделе «Использование параметров в инструкциях» на странице 785.

# Глава 41. Работа с байтовыми массивами

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс ByteArray позволяет считывать и записывать данные в двоичный поток данных — как правило, массив. Этот класс обеспечивает доступ к данным на самом элементарном уровне. Так как компьютерные данные измеряются в байтах, каждый из которых состоит из 8 бит, возможность чтения данных в байтах означает получение доступа к данным, существующим на ином уровне, нежели классы и методы доступа. Класс ByteArray позволяет разбирать любой поток данных, от растрового изображения до потока, передаваемого по сети, на уровне байтов.

Метод `writeObject()` позволяет записать объект в сериализованном формате AMF (Action Message Format) в ByteArray, а метод `readObject()` — считать сериализованный объект из ByteArray в переменную исходного типа данных. Сериализовать можно любые объекты, кроме экранных, т. е. объектов, которые можно добавить в список отображения. Сериализованный объект можно также назначить экземплярам заказного класса, если такой класс доступен во время выполнения. После преобразования объекта в формат AMF его можно легко передать по сети или сохранить в файл.

В приведенном примере приложение Adobe® AIR® читает zip-файл и обрабатывает байтовый поток, извлекая список файлов zip-архива и записывая их на рабочий стол.

### Дополнительные разделы справки

flash.utils.ByteArray

flash.utils.IExternalizable

Спецификация формата AMF (Action Message Format)

## Чтение и запись объектов ByteArray

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс ByteArray входит в пакет flash.utils. Чтобы создать объект ByteArray в ActionScript 3.0, импортируйте класс ByteArray и вызовите конструктор, как показано ниже:

```
import flash.utils.ByteArray;
var stream:ByteArray = new ByteArray();
```

## Методы ByteArray

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Любой осмысленный поток данных организован в таком формате, который можно проанализировать, чтобы найти нужную информацию. Например, запись в личной карточке сотрудника компании скорее всего будет содержать идентификационный номер, имя, адрес, номер телефона и т. д. MP3-файл содержит тег ID3, задающий название, исполнителя, альбом, дату выпуска и жанр загружаемого файла. Формат позволяет определять порядок данных в потоке. Таким образом можно читать поток и извлекать нужные данные.

Класс ByteArray содержит несколько методов, упрощающих чтение и запись данных в поток. К ним относятся методы `readBytes()` и `writeBytes()`, `readInt()` и `writeInt()`, `readFloat()` и `writeFloat()`, `readObject()` и `writeObject()`, а также `readUTFBytes()` и `writeUTFBytes()`. Они позволяют читать данные из потока в переменные отдельных типов данных и записывать из отдельных типов данных непосредственно в двоичный поток.

Например, в коде ниже читается простой строковой массив и числа с плавающей точкой, и каждый элемент записывается в объект ByteArray. Благодаря устройству массива код способен вызвать для записи данных нужные методы ByteArray (`writeUTFBytes()` и `writeFloat()`). Повторяющаяся структура данных позволяет читать массив с помощью цикла.

```
 // The following example reads a simple Array (groceries), made up of strings
// and floating-point numbers, and writes it to a ByteArray.

import flash.utils.ByteArray;

// define the grocery list Array
var groceries:Array = ["milk", 4.50, "soup", 1.79, "eggs", 3.19, "bread" , 2.35]
// define the ByteArray
var bytes:ByteArray = new ByteArray();
// for each item in the array
for (var i:int = 0; i < groceries.length; i++) {
        bytes.writeUTFBytes(groceries[i++]); //write the string and position to the next item
        bytes.writeFloat(groceries[i]);// write the float
        trace("bytes.position is: " + bytes.position);//display the position in ByteArray
}
trace("bytes length is: " +  bytes.length);// display the length
```

## Свойство position

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Свойство position сохраняет текущее положение указателя, определяющего ByteArray во время чтения или записи. Исходное значение свойства position равно 0 (нулю), как показано ниже:

```
var bytes:ByteArray = new ByteArray();
trace("bytes.position is initially: " + bytes.position); // 0
```

При чтении или записи в объект ByteArray используемый метод обновляет свойство position, чтобы указать положение, следующее сразу за последним считанным или записанным байтом. Например, в коде ниже строка записывается в ByteArray, а потом свойство position указывает на байт, следующий сразу за строкой в массиве ByteArray:

```
var bytes:ByteArray = new ByteArray();
trace("bytes.position is initially: " + bytes.position); // 0
bytes.writeUTFBytes("Hello World!");
trace("bytes.position is now: " + bytes.position);// 12
```

Таким же образом, операция чтения увеличивает значение свойства position на количество считанных байтов.

```
var bytes:ByteArray = new ByteArray();

trace("bytes.position is initially: " + bytes.position); // 0
bytes.writeUTFBytes("Hello World!");
trace("bytes.position is now: " + bytes.position);// 12
bytes.position = 0;
trace("The first 6 bytes are: " + (bytes.readUTFBytes(6)));//Hello
trace("And the next 6 bytes are: " + (bytes.readUTFBytes(6)));// World!
```

Обратите внимание, что свойство position может указывать и на конкретное положение в объекте ByteArray, из которого следует выполнять чтение или запись.

## Свойства bytesAvailable и length

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Свойства `length` и `bytesAvailable` сообщают длину ByteArray и количество байтов, оставшихся от текущего положения до конца. В примере ниже показано, как могут использоваться эти свойства. Строка текста записывается в объект ByteArray, а потом ByteArray читается по байтам, пока не будет обнаружен символ «a» или конец массива (`bytesAvailable <= 0`).

```
var bytes:ByteArray = new ByteArray();
var text:String = "Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vivamus etc.";

bytes.writeUTFBytes(text); // write the text to the ByteArray
trace("The length of the ByteArray is: " + bytes.length);// 70
bytes.position = 0; // reset position
while (bytes.bytesAvailable > 0 && (bytes.readUTFBytes(1) != 'a')) {
    //read to letter a or end of bytes
}
if (bytes.position < bytes.bytesAvailable) {
    trace("Found the letter a; position is: " + bytes.position); // 23
    trace("and the number of bytes available is: " + bytes.bytesAvailable);// 47
}
```

## Свойство endian

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Компьютеры отличаются друг от друга способами хранения многобайтовых чисел, т. е. чисел, требующих для хранения более одного байта памяти. Например, целое число может занять 4 байта или 32 бита. Одни компьютеры оставляют низший адрес памяти наиболее значимому байту, а другие — наименее значимому. Этот атрибут компьютера — или порядка данных — может называться *обратный порядок байтов* (сначала самый значимый байт) или *прямой порядок байтов* (сначала наименее значимый байт). Например, число 0x31323334 при обратном порядке байтов и прямом порядке байтов сохранялось бы следующим образом, исходя из того, что a0 — низший адрес памяти с 4 байтами, а a3 — высший:

| Обратный порядок байтов | Обратный порядок байтов | Обратный порядок байтов | Обратный порядок байтов |
|---|---|---|---|
| a0 | a1 | a2 | a3 |
| 31 | 32 | 33 | 34 |

| Прямой порядок байтов | Прямой порядок байтов | Прямой порядок байтов | Прямой порядок байтов |
|---|---|---|---|
| a0 | a1 | a2 | a3 |
| 34 | 33 | 32 | 31 |

Свойство `endian` класса ByteArray позволяет указать тот или иной порядок для многобайтовых чисел. Значением этого свойства может быть `"bigEndian"` или `"littleEndian"`, а класс Endian определяет константы `BIG_ENDIAN` и `LITTLE_ENDIAN` для задания свойства `endian` с помощью этих строк.

## Методы compress() и uncompress()

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `compress()` позволяет сжимать объект ByteArray согласно алгоритму сжатия, заданному соответствующим параметром. Метод `uncompress()` позволяет распаковывать сжатые объекты ByteArray в соответствии с алгоритмом сжатия. После вызова методов `compress()` и `uncompress()` длина байтового массива изменяется, а свойство position принимает в качестве значения конец массива.

Класс CompressionAlgorithm определяет константы, которые могут использоваться для указания алгоритмов сжатия. Класс ByteArray поддерживает алгоритмы deflate (только для AIR), zlib и lzma. Формат упакованных данных zlib описан в документе http://www.ietf.org/rfc/rfc1950.txt. Алгоритм lzma был добавлен для версий Flash Player 11.4 и AIR 3.4. Его описание доступно по адресу http://www.7-zip.org/7z.html.

Алгоритм сжатия deflate используется несколькими форматами: zlib, gzip и некоторыми вариантами zip. Алгоритм упаковки данных deflate описан в документе http://www.ietf.org/rfc/rfc1951.txt.

В примере ниже объект ByteArray с именем `bytes` сжимается с помощью алгоритма lzma:

```
bytes.compress(CompressionAlgorithm.LZMA);
```

В примере ниже объект ByteArray распаковывается с помощью алгоритма deflate:

```
bytes.uncompress(CompressionAlgorithm.LZMA);
```

## Чтение и запись объектов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Методы `readObject()` и `writeObject()` считывают и записывают объекты в ByteArray, кодируя их в формат AMF (Action Message Format). AMF — это запатентованный протокол передачи сообщений Adobe. Он используется различными классами языка ActionScript 3.0, включая Netstream, NetConnection, NetStream, LocalConnection и Shared Objects.

Однобайтовый маркер указывает на тип закодированных данных. В AMF используется 13 типов данных.

```
value-type = undefined-marker | null-marker | false-marker | true-marker | integer-type |
    double-type | string-type | xml-doc-type | date-type | array-type | object-type |
    xml-type | byte-array-type
```

Закодированные данные следуют за маркером типа; исключением являются случаи, когда маркер представляет единственно возможное значение, например «null», «true» или «false», и тогда кодирование не осуществляется.

Существуют две версии AMF: AMF0 и AMF3. AMF 0 поддерживает отправку комплексных объектов с помощью ссылки и позволяет восстанавливать отношения объектов в конечных точках. Формат AMF 3 дополняет AMF 0: он, помимо ссылок на объекты, способен отправлять ссылки на характеристики объекта и строки, а также он поддерживает новые типы данных, представленные в ActionScript 3.0. Свойство `ByteArray.objectEcoding` указывает на версию AMF, которая использовалась для кодирования данных объекта. Класс flash.net.ObjectEncoding определяет константы для указания версии AMF: `ObjectEncoding.AMF0` и `ObjectEncoding.AMF3`.

В примере ниже метод `writeObject()` вызывается для записи XML-объекта в ByteArray, который затем сжимает его с помощью алгоритма Deflate и записывает в файл `order` на рабочем столе. В примере надпись «Wrote order file to desktop!» («Файл записан на рабочий стол!») отображается в окне AIR после окончания.

```
import flash.filesystem.*;
import flash.display.Sprite;
import flash.display.TextField;
import flash.utils.ByteArray;
public class WriteObjectExample extends Sprite
{
    public function WriteObjectExample()
    {
        var bytes:ByteArray = new ByteArray();
        var myLabel:TextField = new TextField();
        myLabel.x = 150;
        myLabel.y = 150;
        myLabel.width = 200;
        addChild(myLabel);

        var myXML:XML =
            <order>
                <item id='1'>
                    <menuName>burger</menuName>
                    <price>3.95</price>
                </item>
                <item id='2'>
                    <menuName>fries</menuName>
                    <price>1.45</price>
                </item>
            </order>;

        // Write XML object to ByteArray
```

```
        bytes.writeObject(myXML);
        bytes.position = 0;//reset position to beginning
        bytes.compress(CompressionAlgorithm.DEFLATE);// compress ByteArray
        writeBytesToFile("order.xml", bytes);
        myLabel.text = "Wrote order file to desktop!";
    }

    private function writeBytesToFile(fileName:String, data:ByteArray):void
    {
        var outFile:File = File.desktopDirectory; // dest folder is desktop
        outFile = outFile.resolvePath(fileName);  // name of file to write
        var outStream:FileStream = new FileStream();
        // open output file stream in WRITE mode
        outStream.open(outFile, FileMode.WRITE);
        // write out the file
        outStream.writeBytes(data, 0, data.length);
        // close it
        outStream.close();
    }
}
```

Метод `readObject()` читает объект в сериализованном формате AMF из объекта ByteArray и сохраняет его в объекте заданного типа. В примере ниже файл `order` читается с рабочего стола в объект ByteArray (`inBytes`), распаковывается, а затем вызывается метод `readObject()`, сохраняющий его в XML-объекте `orderXML`. В примере используется конструкция цикла `for each()`, добавляющая каждый узел в текстовую область. В примере также отображается значение свойства `objectEncoding` и заголовок содержимого файла `order`.

```
import flash.filesystem.*;
import flash.display.Sprite;
import flash.display.TextField;
import flash.utils.ByteArray;

public class ReadObjectExample extends Sprite
{
    public function ReadObjectExample()
    {
        var inBytes:ByteArray = new ByteArray();
        // define text area for displaying XML content
        var myTxt:TextField = new TextField();
        myTxt.width = 550;
        myTxt.height = 400;
        addChild(myTxt);
        //display objectEncoding and file heading
        myTxt.text = "Object encoding is: " + inBytes.objectEncoding + "\n\n" + "order file: \n\n";
        readFileIntoByteArray("order", inBytes);

        inBytes.position = 0; // reset position to beginning
        inBytes.uncompress(CompressionAlgorithm.DEFLATE);
        inBytes.position = 0;//reset position to beginning
        // read XML Object
        var orderXML:XML = inBytes.readObject();
```

```
        // for each node in orderXML
        for each (var child:XML in orderXML)
        {
            // append child node to text area
            myTxt.text += child + "\n";
        }
    }

    // read specified file into byte array
    private function readFileIntoByteArray(fileName:String, data:ByteArray):void
    {
        var inFile:File = File.desktopDirectory; // source folder is desktop
        inFile = inFile.resolvePath(fileName);  // name of file to read
        var inStream:FileStream = new FileStream();
        inStream.open(inFile, FileMode.READ);
        inStream.readBytes(data);
        inStream.close();
    }
}
```

# Пример объекта ByteArray: чтение zip-файла

**Adobe AIR 1.0 и более поздних версий**

В этом примере показано, как читать простой zip-файл, содержащий несколько файлов разных типов. Для этого информация извлекается из метаданных каждого файла, каждый файл распаковывается в ByteArray и записывается на рабочий стол.

Общая структура zip-файла основана на спецификации компании PKWARE, опубликованной по адресу http://www.pkware.com/documents/casestudies/APPNOTE.TXT. Сначала идет заголовок файла и данные первого файла в zip-архиве, затем заголовок и данные второго файла и т. д. (Структура заголовка файла будет рассмотрена немного позднее). Кроме того, zip-файл может содержать запись дескриптора данных (как правило, если итоговый zip-файл был сохранен в память, а не на диск). Также включены несколько дополнительных элементов: заголовок дескриптора архива, запись дополнительных данных архива, структура центрального каталога, конец записи центрального каталога в формате Zip64, локатор центрального каталога в формате Zip64, конец записи центрального каталога.

Код в этом примере подходит только для разбора zip-файлов, не содержащих папок, и ему не требуются записи дескрипторов данных. Никакая информация после данных последнего файла в списке не учитывается.

Формат заголовка файла для каждого таков:

| | |
|---|---|
| подпись заголовка файла | 4 байта |
| требование к версии | 2 байта |
| флажок бита общего назначения | 2 байта |
| метод сжатия | 2 байта (8=DEFLATE; 0=UNCOMPRESSED) |
| время последнего изменения файла | 2 байта |
| дата последнего изменения файла | 2 байта |

| | |
|---|---|
| crc-32 | 4 байта |
| размер после сжатия | 4 байта |
| размер до сжатия | 4 байта |
| длина имени файла | 2 байта |
| длина дополнительного поля | 2 байта |
| имя файла | переменная |
| дополнительное поле | переменная |

За заголовком файла следуют собственно данные файла, который может быть в сжатом или несжатом виде, в зависимости от значения флажка метода сжатия. Флажок равен 0 (нулю), если данные не сжаты; 8, если данные сжаты с помощью алгоритма DEFLATE; любое другое значение говорит о том, что для сжатия использовались иные алгоритмы.

Пользовательский интерфейс в этом примере состоит из подписи и текстовой области (`taFiles`). Приложение записывает в текстовую область для каждого файла zip-архива следующую информацию: имя файла, размер после сжатия, размер до сжатия. Следующий документ MXML определяет интерфейс пользователя для версии Flex приложения:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="init();">
    <mx:Script>
        <![CDATA[
            // The application code goes here
        ]]>
    </mx:Script>
    <mx:Form>
        <mx:FormItem label="Output">
            <mx:TextArea id="taFiles" width="320" height="150"/>
        </mx:FormItem>
    </mx:Form>
</mx:WindowedApplication>
```

В начале программы выполняется ряд задач:

- Импорт требуемого класса

```
import flash.filesystem.*;
import flash.utils.ByteArray;
import flash.events.Event;
```

- Определение пользовательского интерфейса для Flash

```
import fl.controls.*;

//requires TextArea and Label components in the Library
var taFiles = new TextArea();
var output = new Label();
taFiles.setSize(320, 150);
taFiles.move(10, 30);
output.move(10, 10);
output.width = 150;
output.text = "Contents of HelloAir.zip";
addChild(taFiles);
addChild(output);
```

- Определение ByteArray с именем `bytes`

```
var bytes:ByteArray = new ByteArray();
```

- Определение переменных для хранения метаданных заголовка файла

```
// variables for reading fixed portion of file header
var fileName:String = new String();
var flNameLength:uint;
var xfldLength:uint;
var offset:uint;
var compSize:uint;
var uncompSize:uint;
var compMethod:int;
var signature:int;
```

- Определение объектов File (`zfile`) и FileStream (`zStream`), представляющих zip-файл, и указание положения zip-файла, из которого извлекаются файлы — файла HelloAIR.zip на рабочем столе.

```
// File variables for accessing .zip file
var zfile:File = File.desktopDirectory.resolvePath("HelloAIR.zip");
var zStream:FileStream = new FileStream();
```

В Flex программный код начинается с метода `init()`, вызываемого в качестве обработчика события `creationComplete` для корневого тега `mx:WindowedApplication`.

```
// for Flex
private function init():void
{
```

Программа начинается с открытия zip-файла в режиме READ.

```
zStream.open(zfile, FileMode.READ);
```

Затем свойству `endian` объекта `bytes` задается значение `LITTLE_ENDIAN`, чтобы числовые поля читались в прямом порядке байтов.

```
bytes.endian = Endian.LITTLE_ENDIAN;
```

Затем инструкция `while()` начинает цикл, который заканчивается, когда текущее положение в файловом потоке становится больше или равно размеру файла.

```
while (zStream.position < zfile.size)
{
```

Первая инструкция в цикле читает первые 30 байт файлового потока в ByteArray `bytes`. Первые 30 байт составляют часть заголовка файла с фиксированным размером.

```
        // read fixed metadata portion of local file header
        zStream.readBytes(bytes, 0, 30);
```

Затем код считывает целое число (`signature`) из первых байтов 30-байтового заголовка. В определении ZIP-формата указано, что подпись каждого заголовка файла содержится в шестнадцатеричном формате `0x04034b50`. Если подпись другая, это значит, что код вышел за границы файловой части zip-файла, а файлов для извлечения больше нет. В этом случае цикл `while` немедленно прекращается, не дожидаясь конца байтового массива.

```
        bytes.position = 0;
        signature = bytes.readInt();
        // if no longer reading data files, quit
        if (signature != 0x04034b50)
        {
            break;
        }
```

Следующая часть код читает байт заголовка со смещением — из положения 8 и сохраняет значение в переменной `compMethod`. Этот байт содержит значение, указывающее на метод сжатия, который был применен к файлу. Поддерживается ряд методов сжатия, но на практике почти все zip-файлы сжимаются с помощью алгоритма DEFLATE. Если текущий файл сжат с помощью алгоритма DEFLATE, значение `compMethod` равно 8; если файл не сжат, значение `compMethod` равно 0.

```
        bytes.position = 8;
        compMethod = bytes.readByte();  // store compression method (8 == Deflate)
```

За первыми 30 байтами следует часть заголовка переменной длины, содержащая имя файла и, возможно, дополнительное поле. Переменная `offset` хранит размер этой части. Размер подсчитывается путем добавления длины имени файла и длины дополнительного поля, которые читаются из заголовка в положениях 26 и 28.

```
        offset = 0;// stores length of variable portion of metadata
        bytes.position = 26;  // offset to file name length
        flNameLength = bytes.readShort();// store file name
        offset += flNameLength; // add length of file name
        bytes.position = 28;// offset to extra field length
        xfldLength = bytes.readShort();
        offset += xfldLength;// add length of extra field
```

Затем программа читает часть заголовка файла с переменной длиной для установления количества байтов, хранимых в переменной `offset`.

```
        // read variable length bytes between fixed-length header and compressed file data
        zStream.readBytes(bytes, 30, offset);
```

Программа читает имя файла из части заголовка с переменной длиной и отображает его в текстовой области вместе с данными о размере файла в сжатом и несжатом виде.

```
// Flash version
        bytes.position = 30;
        fileName = bytes.readUTFBytes(flNameLength); // read file name
        taFiles.appendText(fileName + "\n"); // write file name to text area
        bytes.position = 18;
        compSize = bytes.readUnsignedInt();  // store size of compressed portion
        taFiles.appendText("\tCompressed size is: " + compSize + '\n');
        bytes.position = 22;  // offset to uncompressed size
        uncompSize = bytes.readUnsignedInt();  // store uncompressed size
        taFiles.appendText("\tUncompressed size is: " + uncompSize + '\n');
```

```
// Flex version
        bytes.position = 30;
        fileName = bytes.readUTFBytes(flNameLength); // read file name
        taFiles.text += fileName + "\n"; // write file name to text area
        bytes.position = 18;
        compSize = bytes.readUnsignedInt();  // store size of compressed portion
        taFiles.text += "\tCompressed size is: " + compSize + '\n';
        bytes.position = 22;  // offset to uncompressed size
        uncompSize = bytes.readUnsignedInt();  // store uncompressed size
        taFiles.text += "\tUncompressed size is: " + uncompSize + '\n';
```

В примере программа читает остаток файла из файлового потока в объект `bytes` с целью установления длины, заданной в сжатом виде, и переписывает заголовок файла в первых 30 байтах. Размер сжатого файла задается точно, даже если файл не сжат, потому что в таком случае размеры сжатого и несжатого файлов совпадают.

```
    // read compressed file to offset 0 of bytes; for uncompressed files
    // the compressed and uncompressed size is the same
    if (compSize == 0) continue;
    zStream.readBytes(bytes, 0, compSize);
```

Затем в примере сжатый файл распаковывается, и вызывается функция `outfile()`, записывающая его в файловый поток вывода. Функции `outfile()` передается имя файла и байтовый массив, содержащий данные файла.

```
        if (compMethod == 8) // if file is compressed, uncompress
        {
            bytes.uncompress(CompressionAlgorithm.DEFLATE);
        }
        outFile(fileName, bytes);   // call outFile() to write out the file
```

В приведенном ранее примере `bytes.uncompress(CompressionAlgorithm.DEFLATE)` будет работать только для приложений AIR. Чтобы распаковать данные, сжатые с использованием алгоритма deflate, и для AIR, и для Flash Player, вызовите функцию `inflate()` класса ByteArray.

Закрывающие скобки указывают на окончание цикла `while`, метода `init()` и кода приложения Flex; исключением является метод `outFile()`. Код возвращается к началу цикла `while` и обрабатывает следующие байты в zip-файле. При этом либо извлекается следующий файл, либо, если последний файл уже обработан, обработка zip-файла завершается.

```
    } // end of while loop
} // for Flex version, end of init() method and application
```

Функция `outfile()` открывает файл вывода на рабочем столе в режиме WRITE и присваивает ему имя, заданное параметром `filename`. Затем данные файла из параметра `data` записываются в файловый поток вывода (`outStream`), а файл закрывается.

```
// Flash version
function outFile(fileName:String, data:ByteArray):void
{
    var outFile:File = File.desktopDirectory; // destination folder is desktop
    outFile = outFile.resolvePath(fileName);  // name of file to write
    var outStream:FileStream = new FileStream();
    // open output file stream in WRITE mode
    outStream.open(outFile, FileMode.WRITE);
    // write out the file
    outStream.writeBytes(data, 0, data.length);
    // close it
    outStream.close();
}

private function outFile(fileName:String, data:ByteArray):void
{
    var outFile:File = File.desktopDirectory; // dest folder is desktop
    outFile = outFile.resolvePath(fileName);  // name of file to write
    var outStream:FileStream = new FileStream();
    // open output file stream in WRITE mode
    outStream.open(outFile, FileMode.WRITE);
    // write out the file
    outStream.writeBytes(data, 0, data.length);
    // close it
    outStream.close();
}
```

# Глава 42. Основные сведения о работе с сетью и обмене данными

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При создании приложений в проигрывателе Flash Player или среде AIR часто требуется получать доступ к ресурсам вне приложения. Например, можно отправить запрос изображения на веб-сервер в Интернете и в ответ получить данные изображения. Или можно передавать сериализованные объекты туда и обратно по соединению через сокет с сервером приложений. API-интерфейсы Flash Player и AIR предоставляют несколько классов, позволяющих приложениям участвовать в этом обмене данными. Эти API-интерфейсы поддерживают взаимодействие по IP-сети для таких протоколов, как UDP, TCP, HTTP, RTMP и RTMFP.

Следующие классы можно использовать для отправки и получения данных по сети:

| Класс | Поддерживаемые форматы данных | Протоколы | Описание |
|---|---|---|---|
| Loader | SWF, PNG, JPEG, GIF | HTTP, HTTPS | Загружает поддерживаемые типы данных и преобразует данные в экранный объект.<br>См. раздел ««Динамическая загрузка содержимого для показа» на странице 214». |
| URLLoader | Любой (text, XML, binary и т. д.) | HTTP, HTTPS | Загружает произвольные форматы данных. Приложение отвечает за интерпретацию данных.<br>См. раздел ««Использование класса URLLoader» на странице 876» |
| FileReference | Любой | HTTP | Отправка и загрузка файлов.<br>См. раздел ««Использование класса FileReference» на странице 701» |
| NetConnection | Видео, аудио, формат ActionScript Message (AMF) | HTTP, HTTPS, RTMP, RTMFP | Подключает потоки видеоданных, аудиоданных и удаленных объектов.<br>См. раздел ««Работа с видео» на странице 508». |
| Sound | Аудио | HTTP | Загружает и воспроизводит поддерживаемые аудиоформаты.<br>См. раздел ««Загрузка внешних звуковых файлов» на странице 477». |
| XMLSocket | XML | TCP | Обменивается сообщениями XML с сервером XMLSocket.<br>См. раздел ««XML-сокеты» на странице 862». |
| Socket | Любой | TCP | Подключается к сокет-серверу TCP.<br>См. раздел ««Двоичные клиентские сокеты» на странице 857». |
| SecureSocket (AIR) | Любой | TCP с шифрованием SSLv3 или TLSv1 | Подключается к сокет-серверу TCP, для которого требуется защита шифрованием SSL или TLS.<br>См. раздел ««Безопасные сокеты клиента (AIR)» на странице 858». |
| ServerSocket (AIR) | Любой | TCP | Выступает в роли сервера для входящих подключений сокетов TCP.<br>См. раздел ««Сокеты сервера» на странице 866». |

| Класс | Поддерживаемые форматы данных | Протоколы | Описание |
|---|---|---|---|
| DatagramSocket (AIR) | Любой | UDP | Отправляет и получает пакеты UDP.<br>См. раздел ««UDP-сокеты (AIR)» на странице 868» |

Часто при создании веб-приложения полезно сохранять постоянные данные о состоянии приложения пользователя. На страницах и в приложениях HTML для этих целей часто используются файлы cookie. В проигрывателе Flash Player для этого можно использовать класс SharedObject. См. раздел ««Общие объекты» на странице 754». (Класс SharedObject можно использовать в приложениях AIR, но на сохранение данных в обычный файл накладывается меньше ограничений.)

Если приложению Flash Player или AIR требуется обмен данными с другим приложением Flash Player или AIR на одном компьютере, можно воспользоваться классом LocalConnection. Например, два (или более) SWF-файла на одной веб-странице могут обмениваться данными друг с другом. Подобным образом SWF-файл, выполняющийся на веб-странице, может обмениваться данными с приложением AIR. См. раздел ««Связь с другими экземплярами Flash Player и AIR» на странице 891».

Если необходима связь с другими процессами, отличными от SWF-файла, можно использовать класс NativeProcess, добавленный в среду AIR 2. Класс NativeProcess позволяет приложению AIR запускаться и обмениваться данными с другими приложениями. См. раздел ««Связь с собственными процессами в AIR» на странице 898».

Если требуется информация о сетевой среде компьютера, на котором выполняется приложение AIR, можно использовать следующие классы:

- NetworkInfo: предоставляет информацию о доступных сетевых интерфейсах, таких как IP-адрес компьютера. См. раздел ««Сетевые интерфейсы» на странице 850».
- DNSResolver: позволяет искать записи DNS. См. раздел ««Записи системы доменных имен (DNS)» на странице 854».
- ServiceMonitor: позволяет следить за доступностью сервера. См. раздел ««Мониторинг служб» на странице 852».
- URLMonitor: позволяет следить за доступностью ресурсов по определенному URL-адресу. См. раздел ««Мониторинг HTTP» на странице 853».
- SocketMonitor и SecureSocketMonitor: позволяют следить за доступностью ресурсов в сокете. См. раздел ««Мониторинг сокетов» на странице 854».

### Важные понятия и термины

Ниже приводится список важных терминов, встречающихся в этой главе при программировании сетевых приложений и приложений для обмена данными.

**Внешние данные** Данные, хранящиеся в какой-либо форме вне приложения и загружаемые в приложение при необходимости. Эти данные могут храниться в файле, который загружается непосредственно, либо в базе данных или иной форме, для доступа к которой требуется выполнение сценариев или программ на сервере.

**Переменные, закодированные как URL-адрес** Кодирование в виде URL-адреса позволяет представить несколько переменных (пары имен и значений переменных) в виде одной строки текста. Отдельные переменные записываются в формате name=value. Каждая переменная (т. е. каждая пара «имя-значение») отделяется символом амперсанд: variable1=value1&variable2=value2. Таким образом одним сообщением можно отправить любое количество переменных.

**Тип MIME** Стандартный код, предназначенный для определения типа данного файла при связи через Интернет. Тип каждого файла определяется по его специальному коду. При отправке файла или сообщения

компьютер (например, веб-сервер или пользовательский экземпляр Flash Player или AIR) указывает тип отправляемого файла.

**HTTP** протокол гипертекстовой передачи, стандартный формат для доставки веб-страниц и различных типов содержимого, отправляемого через Интернет.

**Метод запроса** Когда программа (например, приложение AIR или веб-обозреватель) отправляет сообщение (так называемый запрос HTTP) веб-серверу, отправляемые данные могут встраиваться в запрос двумя способами — с помощью методов запроса GET и POST. На сервере программа, получающая запрос, должна будет просмотреть соответствующую часть запроса, чтобы найти данные, поэтому метод запроса, используемый для отправки данных из вашего приложения, должен соответствовать методу запроса, используемому для чтения данных на сервере.

**Подключение через сокет** Постоянное подключение для осуществления связи между двумя компьютерами.

**Отправка** Отправка файла другому компьютеру.

**Загрузка** Получение файла с другого компьютера.

# Сетевые интерфейсы

**Adobe AIR 2 и более поздних версий**

Объект NetworkInfo можно использовать для обнаружения аппаратных и программных сетевых интерфейсов, доступных для приложения. Объект NetworkInfo является *одиночным* объектом, его не требуется создавать. Вместо этого используйте статическое свойство класса, `networkInfo`, чтобы получить доступ к одиночному объекту NetworkInfo. Объект NetworkInfo также отправляет событие `networkChange` при изменении одного из доступных интерфейсов.

Вызовите метод `findInterfaces()` для получения списка объектов NetworkInterface. Каждый объект NetworkInterface в списке описывает один из доступных интерфейсов. Объект NetworkInterface предоставляет такие данные, как IP-адрес, аппаратный адрес, максимальный размер пакета, а также сообщает, активен ли этот интерфейс.

Следующий код отслеживает свойства NetworkInterface для каждого интерфейса на клиентском компьютере.

```
package {
import flash.display.Sprite;
import flash.net.InterfaceAddress;
import flash.net.NetworkInfo;
import flash.net.NetworkInterface;

public class NetworkInformationExample extends Sprite
{
    public function NetworkInformationExample()
    {
        var networkInfo:NetworkInfo = NetworkInfo.networkInfo;
        var interfaces:Vector.<NetworkInterface> = networkInfo.findInterfaces();

        if( interfaces != null )
        {
            trace( "Interface count: " + interfaces.length );
            for each ( var interfaceObj:NetworkInterface in interfaces )
            {
                trace( "\nname: " + interfaceObj.name );
                trace( "display name: " + interfaceObj.displayName );
                trace( "mtu: "  + interfaceObj.mtu );
                trace( "active?: " + interfaceObj.active );
                trace( "parent interface: " + interfaceObj.parent );
                trace( "hardware address: " + interfaceObj.hardwareAddress );
                if( interfaceObj.subInterfaces != null )
                {
                    trace( "# subinterfaces: " + interfaceObj.subInterfaces.length );
                }
                trace("# addresses: " + interfaceObj.addresses.length );
                for each ( var address:InterfaceAddress in interfaceObj.addresses )
                {
                    trace( "  type: "           + address.ipVersion );
                    trace( "  address: " + address.address );
                    trace( "  broadcast: " + address.broadcast );
                    trace( "  prefix length: " + address.prefixLength );
                }
            }
        }
    }
}
}
```

Дополнительные сведения см. в разделе:

- NetworkInfo
- NetworkInterface
- InterfaceAddress
- Flexpert: определение типа сетевого подключения с помощью Flex 4.5

# Изменения сетевого подключения

**Adobe AIR 1.0 и более поздних версий**

Приложение AIR может работать в средах с нестабильными или переменчивыми условиями сетевого подключения. Чтобы приложение могло управлять подключением к интернет-ресурсам, Adobe AIR отправляет событие изменения сетевого состояния, когда подключение становится доступно или наоборот недоступно. Объект NetworkInfo, как и объект приложения NativeApplication, отправляет событие `networkChange`. Чтобы отвечать на это событие, добавьте прослушиватель:

```
NetworkInfo.networkInfo.addEventListener(Event.NETWORK_CHANGE, onNetworkChange);
```

Также нужно определить функцию обработчика события:

```
function onNetworkChange(event:Event)
{
    //Check resource availability
}
```

Событие `networkChange` говорит не об изменении сетевой активности в целом, а лишь о том, что изменилось отдельное подключение к сети. AIR не предпринимает попыток интерпретировать характер изменения. Компьютер, подключенный к сети, может иметь множество реальных и виртуальных подключений, поэтому разрыв соединения не всегда означат перекрытие доступа к ресурсу. С другой стороны, новые подключения не гарантируют лучшей доступности ресурса. Порой новое соединение может даже заблокировать доступ к ранее доступным ресурсам (например, при подключении к VPN).

В целом, единственным способом определить возможность подключения к удаленному ресурсу является практическая проверка. Инфраструктура мониторинга служб предоставляет основанные на событиях средства для реагирования на изменения сетевых подключений к определенному хосту.

***Примечание.*** *Инфраструктура мониторинга служб обнаруживает, отвечает ли сервер должным образом. Успешная проверка не гарантирует полнофункционального подключения. Масштабируемые веб-службы часто используют средства кэширования и распределения нагрузки, перенаправляющие интернет-трафик группе серверов. В такой ситуации поставщики услуг предоставляют лишь частичный диагноз подключения.*

## Мониторинг служб

**Adobe AIR 1.0 и более поздних версий**

Инфраструктура мониторинга служб, отдельно от инфраструктуры AIR, содержится в файле aircore.swc. Чтобы использовать инфраструктуру, файл aircore.swc необходимо включить в процесс сборки.

Adobe® Flash® Builder автоматически включает эту библиотеку.

Класс ServiceMonitor реализует инфраструктуру для мониторинга сетевых служб и обеспечивает основную функциональность для его осуществления. По умолчанию экземпляр класса ServiceMonitor отправляет события, связанные с сетевым подключением. Объект ServiceMonitor отправляет эти события при создании экземпляра и каждый раз, когда среда выполнения обнаруживает изменение в сети. Кроме того, свойство `pollInterval` экземпляра ServiceMonitor можно настроить для проверки подключения через заданные интервалы в миллисекундах, в дополнение к общим событиям подключения. Объект ServiceMonitor не проверяет подключения по сети, пока не будет вызван метод `start()`.

Класс URLMonitor, подкласс класса ServiceMonitor, обнаруживает изменения подключения по HTTP-протоколу для конкретных запросов URLRequest.

Класс SocketMonitor, также подкласс класса ServiceMonitor, обнаруживает изменения подключения к тому или иному хосту или порту.

***Примечание.*** *В AIR до версии 2 инфраструктура мониторинга служб была опубликована в библиотеке servicemonitor.swc. Теперь эта библиотека не используется. Вместо нее следует использовать библиотеку aircore.swc.*

**Flash CS4 и CS5 Professional**

Порядок использования этих классов в Adobe® Flash® CS4 или CS5 Professional.

1. Выберите в меню «Файл» команду «Параметры публикации».
2. Нажмите кнопку «Параметры» для ActionScript 3.0. Выберите «Путь к библиотеке».
3. Нажмите кнопку «Найти SWC» и откройте папку AIK в каталоге установки Flash Professional.
4. В этой папке найдите каталог /frameworks/libs/air/aircore.swc (для AIR 2) или /frameworks/libs/air/servicemonitor.swc (для AIR 1.5).
5. Нажмите кнопку «OK».
6. Добавьте следующую инструкцию импорта к своему коду ActionScript 3.0.

```
import air.net.*;
```

**Flash CS3 Professional**

Для использования этих классов в Adobe® Flash® CS3 Professional перетащите компонент ServiceMonitorShim из панели «Компоненты» в «Библиотеку». Затем добавьте следующую инструкцию `import` в код ActionScript 3.0.

```
import air.net.*;
```

## Мониторинг HTTP

**Adobe AIR 1.0 и более поздних версий**

Класс URLMonitor определяет, могут ли HTTP-запросы отправляться по заданным адресам на порте 80 (стандартный порт для HTTP-соединений). В коде ниже экземпляр класса URLMonitor используется для определения изменений подключения к сайту Adobe:

```
import air.net.URLMonitor;
import flash.net.URLRequest;
import flash.events.StatusEvent;
var monitor:URLMonitor;
monitor = new URLMonitor(new URLRequest('http://www.example.com'));
monitor.addEventListener(StatusEvent.STATUS, announceStatus);
monitor.start();
function announceStatus(e:StatusEvent):void {
    trace("Status change. Current status: " + monitor.available);
}
```

## Мониторинг сокетов

**Adobe AIR 1.0 и более поздних версий**

Приложения AIR также могут использовать подключения сокетов в рамках пассивных моделей соединений. Брандмауэры и сетевые маршрутизаторы обычно ограничивают сетевые соединения на неавторизованных портах из соображений безопасности. По этой причине разработчикам необходимо учитывать, что у пользователей не всегда есть возможность устанавливать подключения к сокетам.

В коде ниже экземпляр класса SocketMonitor используется для определения изменений подключения к сокету. Осуществляется мониторинг порта 6667, который часто используется для службы IRC.

```
import air.net.ServiceMonitor;
import flash.events.StatusEvent;

socketMonitor = new SocketMonitor('www.example.com',6667);
socketMonitor.addEventListener(StatusEvent.STATUS, socketStatusChange);
socketMonitor.start();

function announceStatus(e:StatusEvent):void {
    trace("Status change. Current status: " + socketMonitor.available);
}
```

Если для сокет-сервер требуется защищенное подключение, вместо класса SocketMonitor можно использовать класс SecureSocketMonitor.

# Записи системы доменных имен (DNS)

**Adobe AIR 2.0 и более поздних версий**

Поиск записей ресурсов DNS можно выполнять с помощью класса DNSResolver. Записи ресурсов DNS предоставляют такую информацию, как IP-адрес доменного имени и доменное имя IP-адреса. Можно выполнять поиск следующих записей ресурсов DNS:

- ARecord — IPv4-адрес для хоста;
- AAAARecord — IPv6-адрес для хоста;
- MXRecord — запись почтового обменника для хоста;
- PTRRecord — имя хоста для IP-адреса.
- SRVRecord — запись службы для службы..

Чтобы найти запись, необходимо передать методу `lookup()` объекта DNSResolver строку запроса и объекта класса, представляющего тип записи. Строка запроса варьируется в зависимости от типа записи.

| Класс записи | Строка запроса | Пример строки запроса |
|---|---|---|
| ARecord | имя хоста | "example.com" |
| AAAARecord | имя хоста | "example.com" |
| MXRecord | имя хоста | "example.com" |
| PTRRecord | IP-адрес | “208.77.188.166” |
| SRVRecord | Идентификатор службы: _service._protocol.host | "_sip._tcp.example.com" |

Следующий пример выполняет поиск IP-адреса для хоста example.com.

```
package
{
    import flash.display.Sprite;
    import flash.events.DNSResolverEvent;
    import flash.events.ErrorEvent;
    import flash.net.dns.ARecord;
    import flash.net.dns.DNSResolver;

    public class DNSResolverExample extends Sprite
    {

        public function DNSResolverExample()
        {
            var resolver:DNSResolver = new DNSResolver();
            resolver.addEventListener( DNSResolverEvent.LOOKUP, lookupComplete );
            resolver.addEventListener( ErrorEvent.ERROR, lookupError );

            resolver.lookup( "example.com.", ARecord );
        }

        private function lookupComplete( event:DNSResolverEvent ):void
        {
            trace( "Query string: " + event.host );
            trace( "Record count: " + event.resourceRecords.length );
            for each( var record:* in event.resourceRecords )
            {
                if( record is ARecord ) trace( record.address );
            }

        }

        private function lookupError( error:ErrorEvent ):void
        {
            trace("Error: " + error.text );
        }
    }
}
```

Дополнительные сведения см. в разделе:

- DNSResolver
- DNSResolverEvent
- ARecord
- AAAARecord
- MXRecord
- PTRRecord
- SRVRecord

# Глава 43. Сокеты

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Сокет — это тип сетевого подключения, устанавливаемого между двумя компьютерными процессами. Как правило, процессы выполняются на двух разных компьютерах, подключенных к IP-сети. Однако подключенные процессы могут выполняться на одном компьютере при использовании IP-адреса «локального хоста».

Adobe Flash Player поддерживает клиентские сокеты TCP (Transport Control Protocol, протокол управления транспортировкой). Приложение Flash Player может подключаться к другому процессу, выступающему в роли сокет-сервера, однако не может принимать запросы входящих подключений от других процессов. Другими словами, приложение Flash Player может подключаться к серверу TCP, но не может выступать в его роли.

API-интерфейс Flash Player также включает класс XMLSocket. Класс XMLSocket использует протокол Flash Player, который позволяет обмениваться сообщениями XML с сервером, поддерживающим работу по этому протоколу. Класс XMLSocket впервые появился в ActionScript 1 и существует до сих пор для обеспечения обратной совместимости. Для новых приложений следует использовать класс Socket, если необходимость подключиться к серверу, настроенному на работу с Flash XMLSocket, отсутствует.

В среду Adobe AIR добавлен ряд дополнительных классов для сетевого программирования на основе сокетов. Приложения AIR могут работать как сокет-серверы TCP при использовании класса ServerSocket и могут подключаться к серверам, для которых требуется защита SSL или TLS, при использовании класса SecureSocket. Приложения AIR могут также отправлять и получать сообщения по протоколу UDP (Universal Datagram Protocol, универсальный протокол передачи данных) при использовании класса DatagramSocket.

### Дополнительные разделы справки

Пакет flash.net

«Подключение к сокетам» на странице 1152

## TCP-сокеты

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Протокол Transmission Control Protocol (TCP) предоставляет возможность обмена сообщениями через постоянное сетевое подключение. При использовании протокола TCP гарантируется доставка сообщений в том порядке, в котором они были отправлены (за исключением случаев возникновения серьезных неполадок в сети). Для соединений TCP требуется «клиент» и «сервер». Проигрыватель Flash Player может создавать сокеты клиента. В дополнение к этому Adobe AIR может создавать сокеты сервера.

Подключения TCP обеспечивают следующие API-интерфейсы ActionScript.

- Socket позволяет клиентскому приложению подключаться к серверу. Класс Socket не прослушивает входящие соединения.
- SecureSocket (AIR) позволяет клиентскому приложению подключаться к доверенному серверу и обмениваться данными с использованием шифрования.

- ServerSocket (AIR) позволяет приложению прослушивать входящие подключения и выступать в роли сервера.
- XMLSocket позволяет клиентскому приложению подключаться к серверу XMLSocket.

## Двоичные клиентские сокеты

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Подключение к двоичному сокету похоже на подключение к XML-сокету, только клиенту и серверу можно обмениваться не только XML-пакетами. Вместо этого подключение может передавать данные в виде двоичной информации. Это позволяет подключаться к различным службам, в том числе к почтовым серверам (POP3, SMTP и IMAP) и серверам новостей (NNTP).

### класс Socket

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс Socket позволяет создавать подключения к сокету, а также считывать и записывать необработанные двоичные данные. Класс Socket используется для взаимодействия с серверами, использующими двоичные протоколы. Используя подключения через двоичный сокет, можно написать код, который взаимодействует с несколькими различными интернет-протоколами, например POP3, SMTP, IMAP и NNTP. Это взаимодействие, в свою очередь, позволяет приложениям подключаться к почтовым серверам и серверам новостей.

Проигрыватель Flash Player может взаимодействовать с сервером напрямую, с использованием двоичного протокола этого сервера. Одни серверы используют прямой порядок байтов, а другие — обратный порядок байтов. Большинство серверов в Интернете используют обратный порядок байтов, так как он является «сетевым порядком байтов». Порядок байтов от младшего к старшему популярен потому, что используется в архитектуре Intel® x86. Необходимо использовать тот порядок байтов, который соответствует порядку байтов сервера, отправляющего или получающего данные. Все операции, выполняемые интерфейсами IDataInput и IDataOutput, и классы, реализующие эти интерфейсы (ByteArray, Socket и URLStream), по умолчанию кодируются в формате от старшего к младшему, то есть начиная с самого значимого байта. Этот порядок байтов по умолчанию был выбран для обеспечения соответствия порядку байтов Java и официальному сетевому порядку байтов. Чтобы поменять используемый порядок байтов с одного на другой, можно задать свойству `endian` значение `Endian.BIG_ENDIAN` или `Endian.LITTLE_ENDIAN`.

*Класс Socket наследует все методы, определенные в интерфейсах IDataInput и IDataOutput (которые находятся flash.utils). Эти методы необходимо использовать для записи и чтения данных объекта Socket.*

Дополнительные сведения см. в разделе:

- Socket
- IDataInput
- IDataOutput
- Событие socketData

## Безопасные сокеты клиента (AIR)

**Adobe AIR 2 и более поздних версий**

Класс SecureSocket можно использовать для подключения к серверам сокетов, которые используют SSL4 или TLS1. Использование защищенного сокета обеспечивает три преимущества: проверку подлинности сервера, целостность данных и конфиденциальность сообщений. Среда выполнения проверяет подлинность сервера с помощью сертификата сервера и его отношения корневому сертификату или сертификату промежуточного центра сертификации в хранилище сертификатов пользователя. Среда выполнения использует алгоритмы шифрования протоколов SSL и TLS, чтобы обеспечивать целостность данных и конфиденциальность сообщений.

При подключении к серверу с помощью объекта SecureSocket среда выполнения проверяет сертификат сервера с помощью хранилища доверенных сертификатов. На платформах Windows и Mac операционная система предоставляет хранилище доверия. В ОС Linux среда выполнения предоставляет собственное хранилище доверенных сертификатов.

Если сертификат сервера недействительный или недоверенный, среда выполнения отправляет событие `ioError`. Чтобы определить, почему не удалось пройти проверку, можно проверить свойство `serverCertificateStatus` объекта SecureSocket. Подключиться к серверу, у которого нет действующего и доверенного сертификата, невозможно.

Класс CertificateStatus определяет строковые константы, которые представляют возможные результаты проверки.

- Expired (истек): прошла дата окончания срока действия сертификата.
- Invalid (недействительный): существует несколько причин, по которым сертификат может быть недействительным. Например, сертификат мог быть изменен, поврежден или иметь неправильный тип.
- Invalid chain (недействительная цепочка): недействителен один или несколько сертификатов в цепочке сертификатов сервера.
- Principal mismatch (принципиальное несоответствие): имя хоста сервера и общее имя сертификата не соответствуют. Другими словами, сервер использует неправильный сертификат.
- Revoked (отозванный): сертификат отозван выпустившим его центром сертификации.
- Trusted (доверенный): сертификат является действительным и доверенным. Объект SecureSocket может подключаться только к серверу, который использует действующий и доверенный сертификат.
- Unknown (неизвестный): объект SecureSocket еще не проверил сертификат. Свойство `serverCertificateStatus` имеет это значение состояние до вызова метода `connect()` и до отправки события `connect` или `ioError`.
- Untrusted signers (недоверенные подписчики): сертификат не восходит по цепочке к доверенному корневому сертификату в хранилище сертификатов на клиентском компьютере.

Для связи посредством объекта SecureSocket требуется сервер, использующий защищенный протокол и имеющий действительный доверенный сертификат. В остальных отношениях объект SecureSocket используется так же, как объект Socket.

Объект SecureSocket поддерживается не всеми платформами. Используйте свойство `isSupported` класса SecureSocket, чтобы проверить, поддерживает ли среда выполнения использование объекта SecureSocket на текущем клиентском компьютере.

Дополнительные сведения см. в разделе:

- SecureSocket

- CertificateStatus
- IDataInput
- IDataOutput
- Событие socketData

## Пример сокета TCP: создание клиента Telnet

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

На примере создания клиента Telnet демонстрируются приемы подключения к удаленному серверу и передачи данных с помощью класса Socket. В данном примере рассматриваются следующие приемы:

- создание пользовательского клиента Telnet с помощью класса Socket;
- отправка текста удаленному серверу с помощью объекта ByteArray;
- обработка данных, полученных от удаленного сервера.

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения Telnet находятся в папке Samples/Telnet. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| TelnetSocket.fla<br>или<br>TelnetSocket.mxml | Основной файл приложения, состоящий из пользовательского интерфейса для Flex (MXML) или Flash (FLA). |
| TelnetSocket.as | Класс документа, отвечающий за поведение интерфейса (только Flash). |
| com/example/programmingas3/Telnet/Telnet.as | Обеспечивает функции клиента Telnet для приложения, включая подключение к удаленному серверу, отправку, получение и отображение данных. |

### Обзор приложения Telnet на базе сокетов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Основной файл TelnetSocket.mxml отвечает за создание пользовательского интерфейса для всего приложения.

Помимо пользовательского интерфейса, этот файл определяет два метода, `login()` и `sendCommand()`, для подключения пользователя к заданному серверу.

В следующем коде указан ActionScript, используемый в основном файле приложения.

```
import com.example.programmingas3.socket.Telnet;

private var telnetClient:Telnet;
private function connect():void
{
    telnetClient = new Telnet(serverName.text, int(portNumber.text), output);
    console.title = "Connecting to " + serverName.text + ":" + portNumber.text;
    console.enabled = true;
}
private function sendCommand():void
{
    var ba:ByteArray = new ByteArray();
    ba.writeMultiByte(command.text + "\n", "UTF-8");
    telnetClient.writeBytesToSocket(ba);
    command.text = "";
}
```

Первая строка кода импортирует класс Telnet из пользовательского пакета com.example.programmingas.socket. Вторая строка кода объявляет экземпляр класса Telnet, `telnetClient`, который инициализируется позже с помощью метода `connect()`. Затем объявляется метод `connect()`, который вводит переменную `telnetClient`, объявленную ранее. Этот метод передает указанные пользователем имя и порт сервера Telnet, а также ссылку на компонент TextArea в списке отображения, который используется для отображения текстовых ответов от сокет-сервера. Последние две строки метода `connect()` задают свойство `title` для компонента Panel и включают Panel, чтобы пользователь мог отправлять данные на удаленный сервер. Последний метод в основном файле приложения, `sendCommand()`, служит для отправки команд пользователя удаленному серверу в виде объекта ByteArray.

### Обзор класса Telnet

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс Telnet отвечает за подключение к удаленному серверу Telnet, а также за отправку и получение данных.

Класс Telnet объявляет следующие личные переменные.

```
private var serverURL:String;
private var portNumber:int;
private var socket:Socket;
private var ta:TextArea;
private var state:int = 0;
```

Первая переменная, `serverURL`, содержит указанный пользователем адрес сервера, к которому нужно подключиться.

Вторая переменная, `portNumber` — это номер порта, в данный момент используемый сервером Telnet. По умолчанию служба Telnet использует порт 23.

Третья переменная, `socket` — это экземпляр Socket, который пытается подключиться к серверу, определенному переменными `serverURL` и `portNumber`.

Четвертая переменная, `ta` — это ссылка на экземпляр компонента TextArea, добавленный в рабочую область. Этот компонент служит для отображения ответов удаленного сервера Telnet или сообщений об ошибках.

Последняя переменная, `state` — это числовое значение, которое служит для определения параметров, поддерживаемых клиентом Telnet.

Как показано в примере, конструктор класса Telnet вызывается методом `connect()` в основном файле приложения.

Конструктор Telnet принимает три параметра: `server`, `port` и `output`. Параметры `server` и `port` указывают имя и номер порта сервера Telnet. Последний параметр, `output` — это ссылка на экземпляр компонента TextArea, добавленный в рабочую область, в котором для пользователей отображается вывод сервера.

```
public function Telnet(server:String, port:int, output:TextArea)
{
    serverURL = server;
    portNumber = port;
    ta = output;
    socket = new Socket();
    socket.addEventListener(Event.CONNECT, connectHandler);
    socket.addEventListener(Event.CLOSE, closeHandler);
    socket.addEventListener(ErrorEvent.ERROR, errorHandler);
    socket.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
    socket.addEventListener(ProgressEvent.SOCKET_DATA, dataHandler);
    Security.loadPolicyFile("http://" + serverURL + "/crossdomain.xml");
    try
    {
        msg("Trying to connect to " + serverURL + ":" + portNumber + "\n");
        socket.connect(serverURL, portNumber);
    }
    catch (error:Error)
    {
        msg(error.message + "\n");
        socket.close();
    }
}
```

### Запись данных для сокета

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы записать данные в подключение к сокету, необходимо вызвать любой метод записи класса Socket. К таким методам записи относятся `writeBoolean()`, `writeByte()`, `writeBytes()`, `writeDouble()` и другие. После этого удалите данные из буфера обмена вывода с помощью метода `flush()`. На сервере Telnet данные для подключения к сокету записываются с помощью метода `writeBytes()`, который принимает в качестве параметра массив байтов и отправляет его в буфер вывода. Метод `writeBytesToSocket()` реализуется следующим образом:

```
public function writeBytesToSocket(ba:ByteArray):void
{
    socket.writeBytes(ba);
    socket.flush();
}
```

Этот метод вызывается методом `sendCommand()` основного файла приложения.

### Отображение сообщений от сокет-сервера

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При получении каждого события или сообщения от сокет-сервера вызывается метод `msg()`. Этот метод добавляет строку в компонент TextArea в рабочей области и вызывает пользовательский метод `setScroll()`, который выполняет прокрутку компонента TextArea до конца. Метод `msg()` реализуется следующим образом:

```
private function msg(value:String):void
{
    ta.text += value;
    setScroll();
}
```

Если не добавлена автоматическая прокрутка содержимого компонента TextArea, пользователям придется вручную перетаскивать полосы прокрутки в текстовой области, чтобы увидеть последний ответ сервера.

### Прокрутка компонента TextArea

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `setScroll()` содержит одну строку кода ActionScript, который прокручивает содержимое компонента TextArea по вертикали, чтобы пользователь мог видеть последнюю строку возвращенного текста. Следующий код демонстрирует метод `setScroll()`:

```
public function setScroll():void
{
    ta.verticalScrollPosition = ta.maxVerticalScrollPosition;
}
```

Этот метод задает свойство `verticalScrollPosition`, которое представляет собой номер строки верхнего ряда символов, отображаемого в данный момент, и присваивает ему значение свойства `maxVerticalScrollPosition`.

## XML-сокеты

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Сокет XML позволяет создать подключение к удаленному серверу, которое остается отрытым до принудительного закрытия. Можно осуществлять обмен данными строк, например XML, между сервером и клиентом. Преимущество сервера с XML-сокетом заключается в том, что пользователю не нужно специально запрашивать данные. Сервер может отправлять данные без получения запроса любому подключенному клиенту.

В проигрывателе Flash Player и в содержимом Adobe AIR за пределами изолированной среды приложения для подключений XML-сокета требуется наличие файла политики сокета на сервере назначения. Дополнительные сведения см. в разделах ««Элементы управления веб-сайта (файлы политики)» на странице 1132» и ««Подключение к сокетам» на странице 1152».

Класс XMLSocket не может туннелировать вход через брандмауэры автоматически, так как в отличие от протокола Real-Time Messaging Protocol (RTMP) XMLSocket не имеет функции HTTP-туннелирования. Если требуется использовать HTTP-туннелирование, попробуйте использовать ПО Flash Remoting или Flash Media Server (поддерживающие протокол RTMP).

На то, каким образом и где содержимое, не входящее в Flash Player или изолированную программную среду приложения AIR, может использовать объект XMLSocket для подключения к серверу, накладываются следующие ограничения.

- В отношении содержимого вне изолированной программной среды приложения метод `XMLSocket.connect()` может устанавливать соединение только с TCP-портами, номера которых начинаются с 1024. Следствием такого ограничения является то, что для серверных управляющих программ, которые обмениваются данными с объектом `XMLSocket`, также должны быть назначены номера портов, начиная с 1024 и выше. Номера портов ниже 1024 часто используются системными службами, например FTP (21), Telnet (23), SMTP (25), HTTP (80) и POP3 (110), поэтому объекты XMLSocket не используют эти порты из соображений безопасности. Ограничение по номерам портов снижает риск несанкционированного доступа к ресурсам и их неправомочного использования.
- В отношении содержимого вне изолированной программной среды приложения метод `XMLSocket.connect()` может устанавливать соединение только с компьютерами, входящими в тот же домен, что и содержимое. (Это ограничение аналогично правилам безопасности, применимым к методу `URLLoader.load()`.) Для подключения к серверной управляющей программе, выполняющейся не в том домене, в который входит содержимое, можно создать на сервере файл междоменной политики, в котором будет прописан доступ определенных доменов. Дополнительные сведения о файлах междоменной политики см. в разделе ««Безопасность AIR» на странице 1160».

***Примечание.*** *Настройка сервера на обмен данными с объектом XMLSocket может представлять некоторую трудность. Если вашему приложению не требуется осуществлять взаимодействие в реальном времени, используйте класс URLLoader вместо класса XMLSocket.*

При помощи методов `XMLSocket.connect()` и `XMLSocket.send()` класса XMLSocket можно передавать данные XML на сервер и получать данные с сервера, используя подключение через сокет. Метод `XMLSocket.connect()` устанавливает подключение к порту веб-сервера через сокет. Метод `XMLSocket.send()` передает объект XML на сервер, указанный в подключении через сокет.

При вызове метода `XMLSocket.connect()` приложение открывает подключение TCP/IP к серверу и не закрывает его до наступления одного из следующих событий.

- Вызов метода `XMLSocket.close()` класса XMLSocket.
- Больше не существует ссылок на объект XMLSocket.
- Завершается работа проигрывателя Flash Player.
- Соединение прервано (например, отключился модем).

## Подключение к серверу при помощи класса XMLSocket

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для создания подключения через сокет необходимо создать серверное приложение, которое будет ожидать запрос на подключение через сокет и отправлять ответ проигрывателю Flash Player или приложению AIR. Такое серверное приложение может быть создано в среде AIR или написано на таком языке программирования, как Java, Python или Perl. Для работы с классом XMLSocket на серверном компьютере необходимо запустить скрытый процесс (daemon), который понимает простой протокол, используемый классом XMLSocket.

- Сообщения XML отправляются с использованием подключения через потоковый сокет TCP/IP в полнодуплексном режиме.
- Каждое сообщение XML представляет собой полноценный документ XML, заканчивающийся нулевым (0) байтом.

- Через одно подключение XMLSocket можно отправить и получить неограниченное количество сообщений XML.

### Создание сокет-сервера Java XML и подключение к нему

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Следующий код иллюстрирует простой сервер XMLSocket, написанный на языке Java, который принимает входящие соединения и отображает полученные сообщения в окне запроса команд. По умолчанию новому серверу назначается порт 8080 локального компьютера, но можно указать и другой номер порта при запуске сервера из командной строки.

Создайте новый текстовый документ и добавьте следующий код.

```
import java.io.*;
import java.net.*;

class SimpleServer
{
    private static SimpleServer server;
    ServerSocket socket;
    Socket incoming;
    BufferedReader readerIn;
    PrintStream printOut;

    public static void main(String[] args)
    {
        int port = 8080;

        try
        {
            port = Integer.parseInt(args[0]);
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            // Catch exception and keep going.
        }

        server = new SimpleServer(port);
    }

    private SimpleServer(int port)
    {
        System.out.println(">> Starting SimpleServer");
        try
        {
            socket = new ServerSocket(port);
            incoming = socket.accept();
            readerIn = new BufferedReader(new InputStreamReader(incoming.getInputStream()));
            printOut = new PrintStream(incoming.getOutputStream());
            printOut.println("Enter EXIT to exit.\r");
            out("Enter EXIT to exit.\r");
            boolean done = false;
            while (!done)
            {
                String str = readerIn.readLine();
                if (str == null)
                {
```

```
                    done = true;
                }
                else
                {
                    out("Echo: " + str + "\r");
                    if(str.trim().equals("EXIT"))
                    {
                        done = true;
                    }
                }
                incoming.close();
            }
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }

    private void out(String str)
    {
        printOut.println(str);
        System.out.println(str);
    }
}
```

Сохраните документ на жесткий диск под именем SimpleServer.java и скомпилируйте его при помощи компилятора Java. При этом создается файл класса Java с именем SimpleServer.class.

Запустить сервер XMLSocket можно, открыв окно запроса команд и введя `java SimpleServer`. Файл SimpleServer.class может размещаться в любом месте на локальном компьютере или в локальной сети; его не обязательно помещать в корневой каталог веб-сервера.

*Если сервер запустить не удается из-за того, что файлы находятся не в пути к классам Java, попробуйте запустить сервер с помощью `java -classpath . SimpleServer`.*

Чтобы подключиться к объекту XMLSocket из приложения, необходимо создать новый экземпляр класса XMLSocket и вызвать метод `XMLSocket.connect()`, передав имя хоста и номер порта, как показано ниже.

```
var xmlsock:XMLSocket = new XMLSocket();
xmlsock.connect("127.0.0.1", 8080);
```

Каждый раз при получении данных с сервера отправляется событие `data` (`flash.events.DataEvent.DATA`).

```
xmlsock.addEventListener(DataEvent.DATA, onData);
private function onData(event:DataEvent):void
{
    trace("[" + event.type + "] " + event.data);
}
```

Чтобы отправить данные серверу XMLSocket, следует использовать метод `XMLSocket.send()` и передать объект XML или строку. Проигрыватель Flash Player преобразует переданный параметр в объект String и отправляет его содержимое серверу XMLSocket, добавив в конце нулевой байт (0).

```
xmlsock.send(xmlFormattedData);
```

Метод `XMLSocket.send()` не возвращает значение, указывающее на то, успешно ли были переданы данные. Если при передаче данных произошла ошибка, выдается ошибка IOError.

*Каждое сообщение, отправляемое сокет-серверу XML, должно заканчиваться символом новой строки (`\n`).*

Дополнительные сведения см. в разделе «XMLSocket».

## Сокеты сервера

**Adobe AIR 2 и более поздних версий**

Используйте класс ServerSocket, чтобы позволить другим процессам подключаться к приложению с помощью сокета TCP. Процесс подключения может выполняться на локальном компьютере или на другом сетевом компьютере. Когда объект ServerSocket получает запрос на подключение, он отправляет событие `connect`. Объект ServerSocketConnectEvent, отправляемый вместе с событием, содержит объект Socket. Этот объект Socket можно использовать для последующей связи с другим процессом.

Чтобы прослушивать входящие подключения сокетов, выполните следующие действия.

1. Создайте объект ServerSocket и свяжите его с локальным портом.
2. Добавьте прослушиватели для события `connect`.
3. Вызовите метод `listen()`.
4. Отвечайте на событие `connect`, которое передает объект Socket для каждого входящего подключения.

Объект ServerSocket продолжает прослушивать новые подключения, пока не будет вызван метод `close()`.

Следующий код демонстрирует создание приложения для сервера сокетов. В этом примере прослушиваются входящие подключения для порта 8087. При получении подключения код выводит сообщение (строку «Connected.» (Подключено.)) клиентскому сокету. После этого сервер пересылает полученные сообщения обратно клиенту.

```
package
{
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.ProgressEvent;
    import flash.events.ServerSocketConnectEvent;
    import flash.net.ServerSocket;
    import flash.net.Socket;

    public class ServerSocketExample extends Sprite
    {
        private var serverSocket:ServerSocket;
        private var clientSockets:Array = new Array();

        public function ServerSocketExample()
        {
            try
            {
                // Create the server socket
                serverSocket = new ServerSocket();

                // Add the event listener
                serverSocket.addEventListener( Event.CONNECT, connectHandler );
                serverSocket.addEventListener( Event.CLOSE, onClose );
```

```
            // Bind to local port 8087
            serverSocket.bind( 8087, "127.0.0.1" );

            // Listen for connections
            serverSocket.listen();
            trace( "Listening on " + serverSocket.localPort );

        }
        catch(e:SecurityError)
        {
            trace(e);
        }
    }

    public function connectHandler(event:ServerSocketConnectEvent):void
    {
        //The socket is provided by the event object
        var socket:Socket = event.socket as Socket;
        clientSockets.push( socket );

        socket.addEventListener( ProgressEvent.SOCKET_DATA, socketDataHandler);
        socket.addEventListener( Event.CLOSE, onClientClose );
        socket.addEventListener( IOErrorEvent.IO_ERROR, onIOError );

        //Send a connect message
        socket.writeUTFBytes("Connected.");
        socket.flush();

        trace( "Sending connect message" );
    }

    public function socketDataHandler(event:ProgressEvent):void
    {
        var socket:Socket = event.target as Socket

        //Read the message from the socket
        var message:String = socket.readUTFBytes( socket.bytesAvailable );
        trace( "Received: " + message);

        // Echo the received message back to the sender
        message = "Echo -- " + message;
        socket.writeUTFBytes( message );
```

```
            socket.flush();
            trace( "Sending: " + message );
        }

        private function onClientClose( event:Event ):void
        {
            trace( "Connection to client closed." );
            //Should also remove from clientSockets array...
        }

        private function onIOError( errorEvent:IOErrorEvent ):void
        {
            trace( "IOError: " + errorEvent.text );
        }

        private function onClose( event:Event ):void
        {
            trace( "Server socket closed by OS." );
        }
}}
```

Дополнительные сведения см. в разделе:

- ServerSocket
- ServerSocketConnectEvent
- Socket

# UDP-сокеты (AIR)

**Adobe AIR 2 и более поздних версий**

Протокол Universal Datagram Protocol (UDP) предоставляет возможность обмена сообщениями через сетевое подключение, не использующее информацию о состоянии. UDP не гарантирует, что сообщения будут доставляться по порядку и что они будут вообще доставлены. При использовании UDP сетевой код операционной системы обычно затрачивает меньше времени на передачу, отслеживание и подтверждение сообщений. Поэтому сообщения UDP обычно доходят до целевого приложения раньше, чем сообщения TCP.

Связь через сокеты UDP применяется, когда необходимо отправлять информацию в реальном времени, например для обновления местоположения в игре или передачи звуковых пакетов в приложении аудиочата. В подобных приложениях допускается некоторый процент потери данных, и высокая скорость передачи важнее гарантированного получения. Однако почти для всех остальных применений лучше использовать сокеты TCP.

Приложение AIR может отправлять и получать сообщения UDP с помощью классов DatagramSocket и DatagramSocketDataEvent. Чтобы отправить или получить сообщение UDP, выполните следующие действия.

1. Создайте объект DatagramSocket.
2. Добавьте прослушиватель для события `data`.
3. Свяжите сокет с локальным IP-адресом и портом с помощью метода `bind()`.
4. Отправьте сообщения, вызвав метод `send()` и передав IP-адрес и порт целевого компьютера.

5 Получите сообщения, ответив на событие `data`. Объект DatagramSocketDataEvent, отправляемый для этого события, содержит объект ByteArray с данными сообщения.

Следующий пример демонстрирует, как приложение может отправлять и получать сообщения UDP. Код отправляет целевому компьютеру одно сообщение, содержащее строку «Hello» (Здравствуйте). Он также отслеживает содержимое получаемых сообщений.

```
package
{
import flash.display.Sprite;
import flash.events.DatagramSocketDataEvent;
import flash.events.Event;
import flash.net.DatagramSocket;
import flash.utils.ByteArray;

public class DatagramSocketExample extends Sprite
{
    private var datagramSocket:DatagramSocket;

    //The IP and port for this computer
    private var localIP:String = "192.168.0.1";
    private var localPort:int = 55555;

    //The IP and port for the target computer
    private var targetIP:String = "192.168.0.2";
    private var targetPort:int = 55555;

    public function DatagramSocketExample()
    {
        //Create the socket
        datagramSocket = new DatagramSocket();
        datagramSocket.addEventListener( DatagramSocketDataEvent.DATA, dataReceived );

        //Bind the socket to the local network interface and port
        datagramSocket.bind( localPort, localIP );

        //Listen for incoming datagrams
        datagramSocket.receive();

        //Create a message in a ByteArray
        var data:ByteArray = new ByteArray();
        data.writeUTFBytes("Hello.");

        //Send the datagram message
        datagramSocket.send( data, 0, 0, targetIP, targetPort);
    }

    private function dataReceived( event:DatagramSocketDataEvent ):void
    {
        //Read the data from the datagram
        trace("Received from " + event.srcAddress + ":" + event.srcPort + "> " +
            event.data.readUTFBytes( event.data.bytesAvailable ) );
    }
}}
```

При использовании сокетов UDP необходимо учитывать следующее.

- Размер одного пакета данных не должен превышать максимального размера пакета (MTU), установленного для сетевого интерфейса или любых промежуточных сетевых узлов между отправителем и получателем. Все данные в объекте ByteArray, передаваемом методу send(), отправляется как один пакет. (При использовании TCP большие сообщения разбиваются на отдельные пакеты.)
- Между отправителем и получателем не выполняется квитирование установки связи. Сообщения отвергаются без вывода ошибки, если целевой узел не существует или для указанного порта нет активного прослушивателя.
- Когда используется метод `connect()`, сообщения, отправляемые из других источников, игнорируются. Подключение UDP обеспечивает только удобную фильтрацию пакетов. Однако это не означает, что для целевого адреса и порта обязательно есть активный процесс прослушивания.
- Трафик UDP может перегружать сеть. Сетевым администраторам может потребоваться внедрить элементы управления качеством обслуживания в случае такой перегрузки. (Протокол TCP имеет встроенное управление трафиком для предотвращения перегрузки каналов связи.)

Дополнительные сведения см. в разделе:

- DatagramSocket
- DatagramSocketDataEvent
- ByteArray

# Адреса IPv6

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Проигрыватель Flash Player 9.0.115.0 и более поздние версии поддерживают IPv6 (интернет-протокол версии 6). IPv6 — это версия Internet Protocol (интернет-протокола), поддерживающего 128-битные адреса (усовершенствование протокола IPv4, поддерживающего только 32-битные адреса). Вам может потребоваться активировать IPv6 на используемых интерфейсах сетевых соединений. Дополнительные сведения см. в справочных документах операционной системы, в которой размещены ваши данные.

Если система хостинга поддерживает IPv6, вы можете указывать цифровые адреса IPv6 в URL-адресах, заключенные в квадратные скобки ([]), как показано ниже:

```
[2001:db8:ccc3:ffff:0:444d:555e:666f]
```

Проигрыватель Flash Player возвращает буквенные значения IPv6 согласно следующим правилам.

- Проигрыватель Flash Player возвращает длинную форму строки для адресов IPv6.
- Значение IP-адреса не имеет сокращений с двойным двоеточием.
- Шестнадцатеричные знаки могут быть только в нижнем регистре.
- Адреса IPv6 заключаются в квадратные скобки ([]).
- Каждый квартет адреса выводится как 0-4 шестнадцатеричных знаков, начальные нули опускаются.
- Квартет адреса из четырех нулей выводится как один ноль (не как двойное двоеточие), за исключением случаев, перечисленных в следующем списке.

Значения IPv6, возвращаемые проигрывателем Flash Player, имеют следующие исключения.

- Неопределенный адрес IPv6 (все нули) выводится как [::].
- Адрес IPv6 кольцевой проверки или локального хоста выводится как [::1].
- Соотнесенные адреса IPv4 (преобразованные в IPv6) выводятся как [::ffff:a.b.c.d], где a.b.c.d представляет типичное значение IPv4 (десятичного адреса с разделительными точками).
- Адреса, совместимые с IPv4, выводятся как [::a.b.c.d], где a.b.c.d представляет типичное значение IPv4 (десятичного адреса с разделительными точками).

# Глава 44. Связь HTTP

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Приложения Adobe® AIR® и Adobe® Flash® Player могут связываться с серверами HTTP для загрузки данных, изображений, видео, а также для обмена сообщениями.

### Дополнительные разделы справки

flash.net.URLLoader

flash.net.URLStream

flash.net.URLRequest

flash.net.URLRequestDefaults

flash.net.URLRequestHeader

flash.net.URLRequestMethod

flash.net.URLVariables

## Загрузка внешних данных

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

ActionScript 3.0 предусматривает механизмы для загрузки данных из внешних источников. Эти источники могут предоставлять статическое содержимое, например текстовые файлы, или динамическое содержимое, созданное веб-сценарием. Данные могут иметь различный формат, и среда выполнения предоставляет функции для декодирования данных и получения доступа к ним. Можно также выполнять передачу данных на внешний сервер как часть процесса получения данных.

### Использование класса URLRequest

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Во многих API-интерфейсах, загружающих внешние данные, класс URLRequest используется для определения свойств необходимого сетевого запроса.

#### Свойства URLRequest

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В изолированной программной среде можно задать следующие свойства объекта URLRequest:

| Свойство | Описание |
|---|---|
| contentType | MIME-тип содержимого, передаваемого с запросом URL. Если объект contentType не задан, значения отправляются в виде `application/x-www-form-urlencoded`. |
| data | Объект, содержащий данные, которые должны быть переданы с запросом URL. |
| digest | Строка, уникальным образом идентифицирующая подписанный компонент платформы Adobe, который должен быть сохранен (или извлечен) в кэш-памяти проигрывателя Adobe® Flash® Player. |
| method | Метод запроса HTTP, например GET или POST. (Содержимое, выполняющееся в домене безопасности приложения AIR, может указывать строки, отличные от `"GET"` или `"POST"`, в качестве свойства `method`. Допускается использование любого глагола HTTP, а по умолчанию используется метод `"GET"`. См. раздел «Безопасность AIR» на странице 1160.) |
| requestHeaders | Массив заголовков запроса HTTP, который должен быть добавлен к запросу HTTP. Обратите внимание, что права на задание некоторых заголовков ограничены в проигрывателе Flash Player, а также в содержимом AIR, выполняемом за пределами изолированной программной среды приложения. |
| url | Задает запрашиваемый URL-адрес. |

В среде AIR можно задать дополнительные свойства класса URLRequest, которые доступны только для содержимого AIR, выполняемого в изолированной программной среде приложения. Содержимое в изолированной программной среде приложения может также определять URL-адреса при помощи новых схем URL-адресов (в дополнение к таким стандартным схемам, как `file` и `http`).

| Свойство | Описание |
|---|---|
| followRedirects | Указывает, выполнять переход по указанному перенаправлению (`true`, значение по умолчанию) или нет (`false`). Эта возможность поддерживается только в изолированной программной среде AIR. |
| manageCookies | Указывает, должен стек протокола HTTP управлять файлами cookie этого запроса (`true`, значение по умолчанию) или нет (`false`). Настройка этого свойства поддерживается только в изолированной программной среде AIR. |
| authenticate | Указывает, должны ли обрабатываться запросы на проверку подлинности (`true`) данного запроса. Настройка этого свойства поддерживается только в изолированной программной среде AIR. По умолчанию проверка подлинности запросов выполняется — при этом может открыться диалоговое окно проверки подлинности, если для сервера необходимо указать учетные данные. Можно также задать имя пользователя и пароль с использованием класса URLRequestDefaults (см. раздел ««Настройка значений по умолчанию для объекта URLRequest (только в среде AIR)» на странице 874»). |
| cacheResponse | Указывает, должны ли сохраняться в кэше данные ответа на этот запрос. Настройка этого свойства поддерживается только в изолированной программной среде AIR. По умолчанию ответ кэшируется (`true`). |
| useCache | Указывает, следует ли обращаться к локальному кэшу перед выборкой данных этим URLRequest. Настройка этого свойства поддерживается только в изолированной программной среде AIR. По умолчанию (значение `true`) используется локальная кэшированная версия, если таковая имеется. |
| userAgent | Указывает строку агента пользователя, используемую в запросе HTTP. |

***Примечание.*** *Класс HTMLLoader имеет связанные свойства для настроек, относящихся к содержимому, загружаемому объектом HTMLLoader. Дополнительные сведения см. в разделе ««О классе HTMLLoader» на странице 1054».*

## Настройка значений по умолчанию для объекта URLRequest (только в среде AIR)

**Adobe AIR 1.0 и более поздних версий**

Класс URLRequestDefaults позволяет задать программные настройки по умолчанию для объектов URLRequest. Например, следующий код задает значения по умолчанию для свойств `manageCookies` и `useCache`. Во всех новых объектах URLRequest для этих свойств будут использоваться указанные значения вместо значений по умолчанию:

```
URLRequestDefaults.manageCookies = false;
URLRequestDefaults.useCache = false;
```

***Примечание.*** *Класс URLRequestDefaults определяется для содержимого, выполняемого только в среде Adobe AIR. Он не поддерживается в приложении Flash Player.*

Класс URLRequestDefaults содержит метод `setLoginCredentialsForHost()`, позволяющий задать имя пользователя и пароль, используемые по умолчанию для определенного узла. Узел, заданный параметром hostname этого метода, может представлять собой домен, например `"www.example.com"`, или домен и номер порта, например `"www.example.com:80"`. Обратите внимание на то, что `"example.com"`, `"www.example.com"` и `"sales.example.com"` считаются уникальными узлами.

Эти учетные данные используются только по требованию сервера. Если пользователь уже прошел проверку подлинности (например, с помощью диалогового окна проверки подлинности), то при вызове метода `setLoginCredentialsForHost()` этот пользователь не изменяется.

Следующий код позволяет задать имя пользователя и пароль по умолчанию, используемые для отправки запросов на веб-сайт www.example.com:

```
URLRequestDefaults.setLoginCredentialsForHost("www.example.com", "Ada", "love1816$X");
```

Параметры URLRequestDefaults применяются только к текущему домену приложения, но с одним исключением. Учетные данные, переданные в метод `setLoginCredentialsForHost()`, используются для запросов, выполняемых в любом программном домене в приложении AIR.

Дополнительные сведения о классе URLRequestDefaults см. в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

## Схемы URI

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Стандартные схемы URI, например описанную ниже, можно использовать в запросах, выполняемых из любой изолированной программной среды:

**http: и https:**

Используйте их для стандартных URL-адресов Интернета (так же, как они используются в веб-браузере).

**file:**

Используйте схему `file:` для указания URL-адреса файла, находящегося в локальной файловой системе. Например:

```
file:///c:/AIR Test/test.txt
```

В среде AIR можно также использовать следующие схемы при определении URL-адреса для содержимого, выполняющегося в изолированной программной среде приложения:

**app:**
Используйте схему `app:`, чтобы указать путь относительно корневого каталога установленного приложения. Например, следующий путь указывает на каталог ресурсов, вложенный в каталог установленного приложения.

```
app:/resources
```

Когда приложение AIR запускается с использованием средства AIR Debug Launcher (ADL), каталогом приложения является каталог, содержащий файл дескриптора приложения.

В URL-адресе (и в свойстве `url`) объекта File, созданного при помощи `File.applicationDirectory`, используется схема URI `app`, как показано ниже:

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("assets");
trace(dir.url); // app:/assets
```

**app-storage:**
Используйте схему `app-storage:`, чтобы указать путь относительно каталога системы хранения данных приложения. Для каждого установленного приложения (и пользователя) среда AIR создает уникальный каталог хранилища приложения, в котором сохраняются все данные, относящиеся к этому приложению. Например, следующий путь указывает на файл prefs.xml в подкаталоге параметров каталога хранилища приложения:

```
app-storage:/settings/prefs.xml
```

В URL-адресе (и в свойстве `url`) объекта File, созданного при помощи `File.applicationStorageDirectory`, используется схема URI `app-storage`, как показано ниже:

```
var prefsFile:File = File.applicationStorageDirectory;
prefsFile = prefsFile.resolvePath("prefs.xml");
trace(dir.prefsFile); // app-storage:/prefs.xml
```

**Адрес получателя mailto:**
Можно использовать схему mailto с указанием адреса получателя в объектах URLRequest, переданных функции `navigateToURL()`. См. раздел ««Открытие URL-адреса в другом приложении» на странице 889».

При помощи объекта URLRequest, использующего любую из этих схем URI, можно определить запрос URL для различных объектов, например объекта FileStream или Sound. Эти схемы можно также использовать для содержимого HTML, выполняющегося в AIR, например в атрибуте `src` тега `img`.

Однако эти схемы URI для среды AIR (`app:` и `app-storage:`) можно использовать только для содержимого в изолированной программной среде приложения. Дополнительные сведения см. в разделе ««Безопасность AIR» на странице 1160».

## Задание переменных URL-адреса

И хотя переменные можно добавить в строку URL-адреса напрямую, проще использовать класс URLVariables для задания любых необходимых в запросе переменных.

Существует три способа добавления параметров в объект URLVariables:

- В конструкторе URLVariables
- С использованием метода `URLVariables.decode()`
- В качестве динамических свойств самого объекта URLVariables

В следующем примере показаны все три метода, а также способ назначения переменных объекту URLRequest:

```
var urlVar:URLVariables = new URLVariables( "one=1&two=2" );
urlVar.decode("amp=" + encodeURIComponent( "&" ) );
urlVar.three = 3;
urlVar.amp2 = "&&";
trace(urlVar.toString()); //amp=%26&amp2=%26%26&one=1&two=2&three=3

var urlRequest:URLRequest = new URLRequest( "http://www.example.com/test.cfm" );
urlRequest.data = urlVar;
```

При определении переменных в конструкторе URLVariables или методе `URLVariables.decode()` убедитесь, что выполнено URL-кодирование символов, имеющих специальное значением в строке URI. Например, при использовании амперсанда в имени или значении параметра необходимо закодировать амперсанд, заменив его символ с `&` на `%26`, поскольку амперсанд используется в качестве разделителя для параметров. Для этого можно использовать функцию `encodeURIComponent()` верхнего уровня.

## Использование класса URLLoader

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс URLLoader позволяет отправлять запрос на сервер и получать доступ к возвращенной информации. Класс URLLoader можно также использовать для доступа к файлам в локальной файловой системе в контекстах, в которых доступ к локальным файлам разрешен (например, в локальной изолированной программной среде с файловой системой Flash Player и изолированной программной среде приложения AIR). Класс URLLoader загружает данные с URL-адреса в виде текста, двоичных данных или переменных в кодировке URL. Класс URLLoader отправляет такие события, как `complete`, `httpStatus`, `ioError`, `open`, `progress` и `securityError`.

Модель обработки событий ActionScript 3.0 значительно отличается от модели ActionScript 2.0, в которой использовались обработчики событий `LoadVars.onData`, `LoadVars.onHTTPStatus` и `LoadVars.onLoad`. Дополнительные сведения об обработке событий в среде ActionScript 3.0 см. в разделе ««Обработка событий» на странице 134»

Загруженные данные доступны только после завершения загрузки. Можно отслеживать ход выполнения загрузки (количество загруженных байтов и общее количество байтов) путем прослушивания отправляемого события `progress`. Однако, если файл загружается достаточно быстро, событие `progress` может не отправляться. При успешной загрузке файла отправляется событие `complete`. Путем задания свойства `dataFormat` объекта URLLoader можно получать данные в виде текста, двоичных данных без сжатия или объекта URLVariables.

Метод `URLLoader.load()` (и дополнительно конструктор класса URLLoader) имеет один параметр, `request`, который является объектом URLRequest. Объект URLRequest содержит всю информацию для одного HTTP-запроса, такую как целевой URL-адрес, метод запроса (`GET` или `POST`), дополнительную информацию заголовка и тип MIME.

Например, для выгрузки пакета XML в серверный сценарий можно использовать такой код:

```
var secondsUTC:Number = new Date().time;
var dataXML:XML =
    <clock>
        <time>{secondsUTC}</time>
    </clock>;
var request:URLRequest = new URLRequest("http://www.yourdomain.com/time.cfm");
request.contentType = "text/xml";
request.data = dataXML.toXMLString();
request.method = URLRequestMethod.POST;
var loader:URLLoader = new URLLoader();
loader.load(request);
```

Предыдущий фрагмент кода создает документ XML с именем `dataXML`, который содержит пакет XML для отправки на сервер. В примере свойству `contentType` объекта URLRequest задается значение `text/xml`, а документ XML назначается свойству `data` объекта URLRequest. Наконец, в примере создается объект URLLoader и осуществляется отправка запроса в удаленный сценарий с использованием метода `load()`.

## Использование класса URLStream

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс URLStream предоставляет доступ к загружаемым данным по мере их получения. Класс URLStream также позволяет закрыть поток до завершения загрузки. Загруженные данные доступны в виде двоичных данных без сжатия.

Перед чтением данных из объекта URLStream используйте свойство `bytesAvailable` для определения наличия достаточного объема данных. Исключение EOFError создается при попытке чтения данных, объем которых превышает доступный объем.

**Событие httpResponseStatus (AIR)**

В среде Adobe AIR класс URLStream отправляет событие `httpResponseStatus` в добавление к событию `httpStatus`. Событие `httpResponseStatus` доставляется перед какими-либо данными ответа. Событие `httpResponseStatus` (представленное классом HTTPStatusEvent) содержит свойство `responseURL`, которое является URL-адресом, с которого был возвращен ответ, и свойство `responseHeaders`, которое является массивом объектов URLRequestHeader, представляющих собой возвращенные заголовки ответов.

---

## Загрузка данных из внешних документов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

При создании динамических приложений может потребоваться загрузка данных из внешних файлов или серверных сценариев. Это позволяет разрабатывать динамические приложения без редактирования и повторной компиляции приложения. Например, чтобы создать приложение типа «совет дня», можно создать сценарий, выполняемый на стороне сервера, который получает случайно выбранный совет и сохраняет его в текстовом файле раз в день. Затем приложение может загружать содержимое статического текстового файла, чтобы каждый раз не отправлять запрос в базу данных.

Следующий код создает экземпляр URLRequest и объект URLLoader, загружающий содержимое внешнего текстового файла params.txt:

```
var request:URLRequest = new URLRequest("params.txt");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

Если не определить метод запроса, Flash Player и Adobe AIR по умолчанию загружают содержимое с помощью метода HTTP `GET`. Чтобы отправить запрос с помощью метода `POST`, задайте для свойства `request.method` значение `POST` с использованием статической константы `URLRequestMethod.POST`, как показано в следующем коде:

```
var request:URLRequest = new URLRequest("sendfeedback.cfm");
request.method = URLRequestMethod.POST;
```

Внешний документ, params.txt, загружаемый во время выполнения, содержит следующие данные:

```
monthNames=January,February,March,April,May,June,July,August,September,October,November,Dece
mber&dayNames=Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
```

Файл содержит два параметра: `monthNames` и `dayNames`. Каждый параметр содержит список значений, разделенных запятыми, который анализируется как строки. Можно разбить список на массивы при помощи метода `String.split()`.

*Старайтесь не использовать зарезервированные слова или языковые конструкции в качестве имен переменных в файлах внешних данных, так как это усложняет чтение и отладку кода.*

После загрузки данных отправляется событие `complete`, и содержимое внешнего документа становится доступным для использования в свойстве `data` объекта URLLoader, как показано в следующем коде:

```
function completeHandler(event)
{
    var loader2 = event.target;
    air.trace(loader2.data);
}
```

Если удаленный документ содержит пары «имя-значение», разбор данных можно выполнить при помощи класса URLVariables, передав ему содержимое загруженного файла, как показано ниже:

```
private function completeHandler(event:Event):void
{
    var loader2:URLLoader = URLLoader(event.target);
    var variables:URLVariables = new URLVariables(loader2.data);
    trace(variables.dayNames);
}
```

Каждая пара «имя-значение» во внешнем файле воссоздается в виде свойства в объекте URLVariables. Каждое свойство в объекте переменных в предыдущем примере кода рассматривается как строка. Если пара «имя-значение» представляет собой список элементов, можно преобразовать строку в массив при помощи метода `String.split()`, как показано ниже:

```
var dayNameArray:Array = variables.dayNames.split(",");
```

*Если загружаются числовые данные из внешних текстовых файлов, значения необходимо преобразовать в числовые с помощью функции верхнего уровня, такой как `int()`, `uint()` или `Number()`.*

Вместо того, чтобы загружать содержимое удаленного файла в виде строки и создавать новый объект URLVariables, можно задать в качестве значения свойства `URLLoader.dataFormat` одно из статических свойств класса URLLoaderDataFormat. Свойство `URLLoader.dataFormat` может иметь одно из трех значений.

- `URLLoaderDataFormat.BINARY` — свойство `URLLoader.data` будет содержать бинарные данные, сохраненные в объекте ByteArray;

- `URLLoaderDataFormat.TEXT` — свойство `URLLoader.data` будет содержать текст в объекте String;
- `URLLoaderDataFormat.VARIABLES` — свойство `URLLoader.data` будет содержать переменные в кодировке URL, сохраненные в объекте URLVariables.

На примере следующего кода показано, как свойство `URLLoader.dataFormat` в значении `URLLoaderDataFormat.VARIABLES` позволяет автоматически разбирать загруженные данные и передавать их в объект URLVariables.

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class URLLoaderDataFormatExample extends Sprite
    {
        public function URLLoaderDataFormatExample()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/params.txt");
            var variables:URLLoader = new URLLoader();
            variables.dataFormat = URLLoaderDataFormat.VARIABLES;
            variables.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                variables.load(request);
            }
            catch (error:Error)
            {
                trace("Unable to load URL: " + error);
            }
        }
        private function completeHandler(event:Event):void
        {
        var loader:URLLoader = URLLoader(event.target);
        trace(loader.data.dayNames);
        }
    }
}
```

***Примечание.*** *По умолчанию свойство `URLLoader.dataFormat` имеет значение `URLLoaderDataFormat.TEXT`.*

Как показано в следующем примере, загрузка XML из внешнего файла не отличается от загрузки URLVariables. Можно создать экземпляр URLRequest и экземпляр URLLoader и использовать их для загрузки удаленного документа XML. После завершения загрузки файла, отправляется событие `Event.COMPLETE`, и содержимое внешнего файла преобразуется в экземпляр XML, который можно разобрать с помощью методов и свойств XML.

```
package
{
    import flash.display.Sprite;
    import flash.errors.*;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLRequest;

    public class ExternalDocs extends Sprite
    {
        public function ExternalDocs()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/data.xml");
            var loader:URLLoader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                loader.load(request);
            }
            catch (error:ArgumentError)
            {
                trace("An ArgumentError has occurred.");
            }
            catch (error:SecurityError)
            {
                trace("A SecurityError has occurred.");
            }
        }
        private function completeHandler(event:Event):void
        {
            var dataXML:XML = XML(event.target.data);
            trace(dataXML.toXMLString());
        }
    }
}
```

## Связь с внешними сценариями

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс URLVariables можно использовать не только для загрузки внешних файлов данных, но и для отправки переменных сценарию, выполняемому на стороне сервера, и обработки ответа сервера. Это используется, например, при программировании какой-либо игры, когда необходимо отправить количество набранных пользователем очков серверу, чтобы определить, нужно ли включать это количество в список лучших результатов, или когда нужно отправить серверу данные пользователя для входа в систему с целью проверки. Сценарий со стороны сервера может обработать имя пользователя и пароль, проверить их по базе данных и возвратить подтверждение подлинности предоставленных пользователем учетных данных.

Следующий код создает объект URLVariables с именем `variables`, который создает новую переменную с именем `name`. Затем создается объект URLRequest, задающий URL-адрес серверного сценария, которому нужно отправить переменные. После этого задается свойство `method` объекта URLRequest для отправки переменных в качестве запроса HTTP `POST`. Чтобы добавить объект URLVariables в запрос URL, свойству `data` объекта URLRequest присваивается созданный ранее объект URLVariables. Наконец, создается экземпляр URLLoader и вызывается метод `URLLoader.load()`, который инициирует запрос.

```
var variables:URLVariables = new URLVariables("name=Franklin");
var request:URLRequest = new URLRequest();
request.url = "http://www.[yourdomain].com/greeting.cfm";
request.method = URLRequestMethod.POST;
request.data = variables;
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.VARIABLES;
loader.addEventListener(Event.COMPLETE, completeHandler);
try
{
    loader.load(request);
}
catch (error:Error)
{
    trace("Unable to load URL");
}

function completeHandler(event:Event):void
{
    trace(event.target.data.welcomeMessage);
}
```

Следующий код включает содержимое документа Adobe ColdFusion® greeting.cfm, использованное в предыдущем примере.

```
<cfif NOT IsDefined("Form.name") OR Len(Trim(Form.Name)) EQ 0>
    <cfset Form.Name = "Stranger" />
</cfif>
<cfoutput>welcomeMessage=#UrlEncodedFormat("Welcome, " & Form.name)#
</cfoutput>
```

# Запросы веб-службы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Существует множество основанных на протоколе HTTP веб-служб. Основные типы включают:

- REST
- XML-RPC
- Протокол SOAP

Чтобы использовать веб-службу в среде ActionScript 3, создайте объект URLRequest, определите вызов веб-службы с использованием переменных URL-адреса или документа XML и передайте вызов в службу с помощью объекта URLLoader. Среда Flex содержит несколько классов, которые облегчают использование веб-служб и особенно полезны при доступе к сложным службам SOAP. Начиная с версии Flash Professional CS3, классы Flex можно использовать в приложениях, разработанных в среде Flash Professional, а также приложениях, разработанных в среде Flash Builder.

В основанных на коде HTML приложениях AIR можно использовать классы URLRequest и URLLoader или класс Javascript XMLHttpRequest. При необходимости можно также создать библиотеку SWF, распространяющую компоненты служб среды Flex в код Javascript.

Когда приложение выполняется в браузере, можно использовать только веб-службы, находящиеся в том же интернет-домене, что и вызывающий SWF-файл, если только на сервере, на котором размещена веб-служба, также не содержится файл политики взаимодействия доменов, который разрешает доступ из других доменов. Методом, который часто используется, когда файл политики взаимодействия доменов недоступен, является представление запросов через собственный сервер. Adobe Blaze DS и Adobe LiveCycle поддерживают представление веб-службы.

В приложениях AIR файл политики взаимодействия доменов не требуется, если вызов веб-службы выполняется из изолированной программной среды приложения. Содержимое приложения AIR никогда не обслуживается из удаленного домена, поэтому оно не участвует в типах атак, запрещенных политиками взаимодействия доменов. В основанных на коде HTML приложениях AIR содержимое в изолированной программной среде приложения может выполнять запросы XMLHttpRequests между доменами. Можно разрешить содержимому в других изолированных программных средах выполнять междоменные запросы XMLHttpRequests, если содержимое загружено в iframe.

### Дополнительные разделы справки

«Элементы управления веб-сайта (файлы политики)» на странице 1132

Adobe BlazeDS

Adobe LiveCycle ES2

Архитектура REST

XML-RPC

Протокол SOAP

## Запросы веб-служб REST

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Веб-службы REST используют команды метода HTTP для назначения основного действия и переменные URL для указания сведений о действии. Например, в запросе для получения данных элемента может использоваться команда GET и переменные URL для указания имени метода и идентификатора элемента. Получающаяся в результате строка URL может иметь следующий вид:

```
http://service.example.com/?method=getItem&id=d3452
```

Для доступа к веб-службе REST с помощью кода ActionScript можно использовать классы URLRequest, URLVariables и URLLoader. В коде Javascript в приложении AIR можно также использовать класс XMLHttpRequest.

Программирование вызова веб-службы REST на языке ActionScript обычно включает следующие действия:

1. Создание объекта URLRequest.
2. Задание команд методов URL и HTTP службы в объекте запроса.
3. Создание объекта URLVariables.
4. Задание параметров вызова службы в качестве динамических свойств объекта переменных.
5. Назначение объекта переменных свойству данных объекта запроса.
6. Отправка вызова в службу с помощью объекта URLLoader.

7 Обработка события `complete`, отправленного объектом URLLoader и обозначающего завершение вызова службы. Также мудрым решением является прослушивание различных событий ошибок, которые могут быть отправлены объектом URLLoader.

Например, примите во внимание веб-службу, которая распространяет метод тестирования, возвращающий те же параметры вызова обратно отправителю запроса. Следующий код ActionScript можно использовать для вызова службы:

```
import flash.events.Event;
import flash.events.ErrorEvent;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.net.URLRequestMethod;
import flash.net.URLVariables;

private var requestor:URLLoader = new URLLoader();
public function restServiceCall():void
{
    //Create the HTTP request object
    var request:URLRequest = new URLRequest( "http://service.example.com/" );
    request.method = URLRequestMethod.GET;

    //Add the URL variables
    var variables:URLVariables = new URLVariables();
    variables.method = "test.echo";
    variables.api_key = "123456ABC";
    variables.message = "Able was I, ere I saw Elba.";
    request.data = variables;

    //Initiate the transaction
    requestor = new URLLoader();
    requestor.addEventListener( Event.COMPLETE, httpRequestComplete );
    requestor.addEventListener( IOErrorEvent.IOERROR, httpRequestError );
    requestor.addEventListener( SecurityErrorEvent.SECURITY_ERROR, httpRequestError );
    requestor.load( request );
}
private function httpRequestComplete( event:Event ):void
{
    trace( event.target.data );
}

private function httpRequestError( error:ErrorEvent ):void{
    trace( "An error occured: " + error.message );
}
```

В сценарии Javascript в приложении AIR этот запрос можно выполнить с помощью объекта XMLHttpRequest:

```
<html>
<head><title>RESTful web service request</title>
<script type="text/javascript">

function makeRequest()
{
    var requestDisplay = document.getElementById( "request" );
    var resultDisplay  = document.getElementById( "result" );

    //Create a conveninece object to hold the call properties
    var request = {};
    request.URL = "http://service.example.com/";
    request.method = "test.echo";
    request.HTTPmethod = "GET";
    request.parameters = {};
    request.parameters.api_key = "ABCDEF123";
    request.parameters.message = "Able was I ere I saw Elba.";
    var requestURL = makeURL( request );
    xmlhttp = new XMLHttpRequest();
    xmlhttp.open( request.HTTPmethod, requestURL, true);
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            resultDisplay.innerHTML = xmlhttp.responseText;
        }
    }
    xmlhttp.send(null);

    requestDisplay.innerHTML = requestURL;
}
//Convert the request object into a properly formatted URL
function makeURL( request )
{
    var url = request.URL + "?method=" + escape( request.method );
    for( var property in request.parameters )
    {
        url += "&" + property + "=" + escape( request.parameters[property] );
    }

    return url;
}
</script>
</head>
<body onload="makeRequest()">
<h1>Request:</h1>
<div id="request"></div>
<h1>Result:</h1>
<div id="result"></div>
</body>
</html>
```

## Запросы веб-служб XML-RPC

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Служба XML-RPC использует параметры вызова как документ XML, а не как набор переменных URL. Для выполнения транзакции с использованием службы XML-RPC создайте правильно отформатированное сообщение XML и отправьте его в веб-службу с помощью HTTP-метода `POST`. Помимо этого, следует задать заголовок `Content-Type` запроса, чтобы сервер обрабатывал данные запроса в формате XML.

Следующие пример показывает способ использования вызова веб-службы, показанного в примере REST, но на этот раз как службы XML-RPC:

```
import flash.events.Event;
import flash.events.ErrorEvent;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.net.URLRequestMethod;
import flash.net.URLVariables;
public function xmlRPCRequest():void
{
    //Create the XML-RPC document
    var xmlRPC:XML = <methodCall>
                                    <methodName></methodName>
                                    <params>
                                        <param>
                                            <value>
                                                <struct/>
                                            </value>
                                        </param>
                                    </params>
                                </methodCall>;

    xmlRPC.methodName = "test.echo";

    //Add the method parameters
    var parameters:Object = new Object();
    parameters.api_key = "123456ABC";
    parameters.message = "Able was I, ere I saw Elba.";

    for( var propertyName:String in parameters )
    {
        xmlRPC..struct.member[xmlRPC..struct.member.length + 1] =
            <member>
                <name>{propertyName}</name>
                <value>
                    <string>{parameters[propertyName]}</string>
                </value>
            </member>;
    }

    //Create the HTTP request object
    var request:URLRequest = new URLRequest( "http://service.example.com/xml-rpc/" );
    request.method = URLRequestMethod.POST;
    request.cacheResponse = false;
    request.requestHeaders.push(new URLRequestHeader("Content-Type", "application/xml"));
```

```
    request.data = xmlRPC;

    //Initiate the request
    requestor = new URLLoader();
    requestor.dataFormat = URLLoaderDataFormat.TEXT;
    requestor.addEventListener( Event.COMPLETE, xmlRPCRequestComplete );
    requestor.addEventListener( IOErrorEvent.IO_ERROR, xmlRPCRequestError );
    requestor.addEventListener( SecurityErrorEvent.SECURITY_ERROR, xmlRPCRequestError );
    requestor.load( request );
}

private function xmlRPCRequestComplete( event:Event ):void
{
    trace( XML(event.target.data).toXMLString() );
}

private function xmlRPCRequestError( error:ErrorEvent ):void
{
    trace( "An error occurred: " + error );
}
```

WebKit в среде AIR не поддерживает синтаксис E4X, поэтому метод, используемый для создания документа XML в предыдущем примере, не работает в коде Javascript. Вместо этого необходимо использовать методы DOM для создания документа XML или создать документ в качестве строки и использовать класс Javascript DOMParser для преобразования строки в формат XML.

В следующем примере методы DOM используются для создания сообщения XML-RPC и объекта XMLHttpRequest для выполнения транзакции веб-службы:

```
<html>
<head>
<title>XML-RPC web service request</title>
<script type="text/javascript">

function makeRequest()
{
    var requestDisplay = document.getElementById( "request" );
    var resultDisplay  = document.getElementById( "result" );

    var request = {};
    request.URL = "http://services.example.com/xmlrpc/";
    request.method = "test.echo";
    request.HTTPmethod = "POST";
    request.parameters = {};
    request.parameters.api_key = "123456ABC";
    request.parameters.message = "Able was I ere I saw Elba.";
    var requestMessage = formatXMLRPC( request );

    xmlhttp = new XMLHttpRequest();
    xmlhttp.open( request.HTTPmethod, request.URL, true);
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            resultDisplay.innerText = xmlhttp.responseText;
        }
    }
    xmlhttp.send( requestMessage );
```

```
    requestDisplay.innerText = xmlToString( requestMessage.documentElement );
}

//Formats a request as XML-RPC document
function formatXMLRPC( request )
{
    var xmldoc = document.implementation.createDocument( "", "", null );
    var root = xmldoc.createElement( "methodCall" );
    xmldoc.appendChild( root );
    var methodName = xmldoc.createElement( "methodName" );
    var methodString = xmldoc.createTextNode( request.method );
    methodName.appendChild( methodString );

    root.appendChild( methodName );

        var params = xmldoc.createElement( "params" );
        root.appendChild( params );

        var param = xmldoc.createElement( "param" );
        params.appendChild( param );
        var value = xmldoc.createElement( "value" );
        param.appendChild( value );
        var struct = xmldoc.createElement( "struct" );
        value.appendChild( struct );

        for( var property in request.parameters )
        {
            var member = xmldoc.createElement( "member" );
            struct.appendChild( member );

            var name = xmldoc.createElement( "name" );
            var paramName = xmldoc.createTextNode( property );
            name.appendChild( paramName )
            member.appendChild( name );

            var value = xmldoc.createElement( "value" );
            var type = xmldoc.createElement( "string" );
            value.appendChild( type );
            var paramValue = xmldoc.createTextNode( request.parameters[property] );
            type.appendChild( paramValue )
            member.appendChild( value );
        }
    return xmldoc;
}

//Returns a string representation of an XML node
function xmlToString( rootNode, indent )
{
    if( indent == null ) indent = "";
    var result = indent + "<" + rootNode.tagName + ">\n";
    for( var i = 0; i < rootNode.childNodes.length; i++)
    {
        if(rootNode.childNodes.item( i ).nodeType == Node.TEXT_NODE )
        {
            result += indent + "    " + rootNode.childNodes.item( i ).textContent + "\n";
        }
    }
```

```
    if( rootNode.childElementCount > 0 )
    {
        result += xmlToString( rootNode.firstElementChild, indent + "    " );
    }
    if( rootNode.nextElementSibling )
    {
        result += indent + "</" + rootNode.tagName + ">\n";
        result += xmlToString( rootNode.nextElementSibling, indent );
    }
    else
    {
        result += indent +"</" + rootNode.tagName + ">\n";
    }
    return result;
}

</script>
</head>
<body onload="makeRequest()">
<h1>Request:</h1>
<pre id="request"></pre>
<h1>Result:</h1>
<pre id="result"></pre>
</body>
</html>
```

## Запросы веб-служб SOAP

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Протокол SOAP создан на основе общей концепции веб-служб XML-RPC и обеспечивает более широкий, хотя и более сложный, набор средств для передачи типизированных данных. Веб-службы SOAP обычно предоставляют файл WSDL (Web Service Description Language), в котором указаны вызовы веб-службы, типы данных и URL-адрес службы. И хотя ActionScript 3 не предоставляет прямую поддержку SOAP, можно создать сообщение SOAP XML «вручную», опубликовать его на сервере и выполнить синтаксический анализ результатов. Однако для любых веб-служб SOAP, за исключением простейших, вероятно, можно значительно уменьшить время разработки за счет использования существующей библиотеки SOAP.

Среда Flex включает библиотеки для доступа к веб-службам SOAP. В среде Flash Builder библиотека (файл rpc.swc) автоматически включается в проекты Flex, поскольку она является компонентом среды Flex. В среде Flash Professional можно добавить файлы Flex framework.swc и rpc.swc в путь к библиотеке в проекте, а затем получать доступ к классам Flex в коде ActionScript.

### Дополнительные разделы справки

Использование компонента веб-служб Flex в среде Flash Professional

Кристоф Конрац (Christophe Coenraets): Real-time Trader Desktop для Android

# Открытие URL-адреса в другом приложении

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Функцию `navigateToURL()` можно использовать для открытия URL-адреса в веб-браузере или другом приложении. Для содержимого, выполняемого в AIR, функция `navigateToURL()` открывает страницу в системном веб-браузере по умолчанию.

С объектом URLRequest, передаваемым в качестве параметра `request` этой функции, используется только свойство `url`.

Первым параметром функции `navigateToURL()`, параметром `navigate`, является объект URLRequest (см. раздел ««Использование класса URLRequest» на странице 872»). Вторым является дополнительный параметр `window`, в котором можно задать имя окна. Например, следующий код позволяет открыть веб-страницу www.adobe.com:

```
var url:String = "http://www.adobe.com";
var urlReq:URLRequest = new URLRequest(url);
navigateToURL(urlReq);
```

***Примечание.*** *При использовании функции `navigateToURL()` среда выполнения обрабатывает объект URLRequest, использующий метод POST (т. е. тот, у которого для свойства `method` задано `URLRequestMethod.POST`) так же, как при использовании метода GET.*

При использовании функции `navigateToURL()` использование схем URI разрешается в зависимости от изолированной программной среды кода, вызывающего функцию `navigateToURL()`.

Некоторые API-интерфейсы позволяют запускать содержимое в веб-обозревателе. В целях защиты некоторые схемы URI недопустимы при использовании этих API-интерфейсов в среде AIR. Перечень недопустимых схем зависит от изолированной программной среды кода, использующего API-интерфейс. (Дополнительные сведения об изолированных программных средах безопасности см. в разделе «Безопасность AIR» на странице 1160.)

### Изолированная программная среда приложения (только для AIR)

В URL-адресе, который запущен содержимым, выполняемым в изолированной среде безопасности приложения AIR, можно использовать любую схему URI. Приложение должно быть зарегистрированным, чтобы можно было обработать схему URI. В противном случае не будет результата. Следующие схемы поддерживаются на многих компьютерах и устройствах.

- `http:`
- `https:`
- `file:`
- `mailto:` — AIR направляет эти запросы зарегистрированному системному почтовому приложению
- `sms:` — AIR направляет запросы `sms:` в стандартное приложение обмена текстовыми сообщениями. URL-адреса должны быть указаны в формате, соответствующем правилам системы, в которой приложение выполняется. Например, на платформе Android схема URI должна включать символы в нижнем регистре.

  ```
  navigateToURL( new URLRequest( "sms:+15555550101") );
  ```

- `tel:` — AIR направляет запросы `tel:` стандартному приложению набора телефонных номеров. URL-адреса должны быть указаны в формате, соответствующем правилам системы, в которой приложение выполняется. Например, на платформе Android схема URI должна включать символы в нижнем регистре.

  ```
  navigateToURL( new URLRequest( "tel:5555555555") );
  ```

- `market:` — AIR направляет запросы `market:` приложению«Маркет», которое обычно поддерживается на устройствах Android.

```
navigateToURL( new URLRequest( "market://search?q=Adobe Flash") );
navigateToURL( new URLRequest( "market://search?q=pname:com.adobe.flashplayer") );
```

Если позволяет операционная система, приложения могут определять и регистрировать пользовательские схемы URI. Можно создавать URL с использованием схемы для запуска приложения из AIR.

**Удаленные изолированные программные среды**

Допускается использование следующих схем. Используйте эти схемы, как в веб-браузере.

- `http:`
- `https:`
- `mailto:` — AIR направляет эти запросы зарегистрированному системному почтовому приложению

Использование других схем URI запрещено.

**Локальная изолированная программная среда файла**

Допускается использование следующих схем. Используйте эти схемы, как в веб-браузере.

- `file:`
- `mailto:` — AIR направляет эти запросы зарегистрированному системному почтовому приложению

Использование других схем URI запрещено.

**Локальная изолированная программная среда с сетевым подключением**

Допускается использование следующих схем. Используйте эти схемы, как в веб-браузере.

- `http:`
- `https:`
- `mailto:` — AIR направляет эти запросы зарегистрированному системному почтовому приложению

Использование других схем URI запрещено.

**Локальная доверенная изолированная программная среда**

Допускается использование следующих схем. Используйте эти схемы, как в веб-браузере.

- `file:`
- `http:`
- https:
- `mailto:` — AIR направляет эти запросы зарегистрированному системному почтовому приложению

Использование других схем URI запрещено.

# Глава 45. Связь с другими экземплярами Flash Player и AIR

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс LocalConnection позволяет устанавливать связь между приложениями Adobe® AIR®, а также между содержимым SWF, выполняющимся в браузере. Можно также использовать класс LocalConnection для установления связи между приложением AIR и содержимым SWF, выполняющимся в браузере. Класс LocalConnection позволяет создавать легко приспосабливаемые приложения, поддерживающие обмен данными между экземплярами Flash Player и AIR.

## О классе LocalConnection

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс LocalConnection позволяет создавать SWF-файлы, которые могут отправлять инструкции другим SWF-файлам, не используя метод fscommand() или JavaScript. Объекты LocalConnection могут поддерживать связь только с SWF-файлами, выполняемыми на одном клиентском компьютере, но, возможно, в разных приложениях. Например, SWF-файл, выполняемый в браузере, и SWF-файл, выполняемый в проекторе, могут обмениваться информацией, если проектор содержит локальные сведения, а SWF-файл в обозревателе подключается удаленно. (Проектор — это SWF-файл, сохраненный в формате, который позволяет ему работать в качестве автономного приложения, то есть для проектора не требуется установка проигрывателя Flash Player, так как он встроен в исполняемый файл.)

Объекты LocalConnection можно использовать для поддержания связи между SWF-файлами, созданными с помощью разных версий ActionScript:

- Объекты ActionScript 3.0 LocalConnection могут поддерживать связь с объектами LocalConnection, созданными в версиях ActionScript 1.0 и 2.0.
- Объекты LocalConnection версии ActionScript 1.0 или 2.0 могут поддерживать связь с объектами LocalConnection, созданными в версии ActionScript 3.0.

Проигрыватель Flash Player автоматически обрабатывает такую связь между объектами LocalConnection разных версий.

Самым простым способом использования объекта LocalConnection является разрешение установления связи только между объектами LocalConnection, расположенными в том же домене или в том же приложении AIR. В этом случае не нужно беспокоиться о безопасности. Однако если нужно разрешить установление связи между доменами, следует принять меры безопасности. Дополнительные сведения см. в обсуждении параметра `connectionName` метода `send()` и в описаниях `allowDomain()` и `domain` в классе LocalConnection в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

*Объекты LocalConnection можно использовать для отправки и получения данных в одном SWF-файле, но компания Adobe не рекомендует это делать. Вместо этого используйте общие объекты.*

Существует три способа добавлять методы обратного вызова в объекты LocalConnection:

- создать подкласс для LocalConnection и добавить методы;

- задать в качестве значения свойства `LocalConnection.client` объект, реализующий методы;
- создать динамический класс, расширяющий LocalConnection, и присоединить методы в динамическом режиме.

Первый способ добавления методов обратного вызова подразумевает создание подкласса для LocalConnection. Необходимо определить методы в пользовательском классе, а не добавлять их динамически в экземпляр LocalConnection. Этот подход демонстрируется на примере следующего кода.

```
package
{
    import flash.net.LocalConnection;
    public class CustomLocalConnection extends LocalConnection
    {
        public function CustomLocalConnection(connectionName:String)
        {
            try
            {
                connect(connectionName);
            }
            catch (error:ArgumentError)
            {
                // server already created/connected
            }
        }
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

Чтобы создать новый экземпляр класса CustomLocalConnection, необходимо использовать следующий код:

```
var serverLC:CustomLocalConnection;
serverLC = new CustomLocalConnection("serverName");
```

Второй способ добавления методов обратного вызова подразумевает использование свойства `LocalConnection.client`. Для этого нужно создать пользовательский класс и назначить новый экземпляр свойству `client`, как показано в следующем коде.

```
var lc:LocalConnection = new LocalConnection();
lc.client = new CustomClient();
```

Свойство `LocalConnection.client` указывает требуемые методы обратного вызова объекта. В предыдущем коде в качестве значения свойства `client` был задан новый экземпляр пользовательского класса, CustomClient. Значением по умолчанию для свойства `client` является текущий экземпляр LocalConnection. Свойство `client` можно использовать, если имеется два обработчика данных с одинаковыми методами, но разными видами поведения, например в приложении, где кнопка в одном окне переключает вид во втором окне.

Чтобы создать класс CustomClient, можно использовать следующий код:

```
package
{
    public class CustomClient extends Object
    {
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

Третий способ добавления методов обратного вызова подразумевает создание динамического класса и присоединение методов во время выполнения. это очень похоже на использование класса LocalConnection в предыдущих версиях ActionScript, как показано в следующем коде.

```
import flash.net.LocalConnection;
dynamic class DynamicLocalConnection extends LocalConnection {}
```

Методы обратного вызова можно динамически добавлять к этому классу с помощью следующего кода.

```
var connection:DynamicLocalConnection = new DynamicLocalConnection();
connection.onMethod = this.onMethod;
// Add your code here.
public function onMethod(timeString:String):void
{
    trace("onMethod called at: " + timeString);
}
```

Предыдущий способ добавления методов обратного вызова не рекомендуется, так как полученный с его помощью код не отличается особой портативностью. Кроме того, такой способ создания локальных подключений может вызвать проблемы с быстродействием, так как обращение к динамическим свойствам выполняется намного медленнее, чем обращение к фиксированным свойствам.

### Свойство isPerUser

Свойство `isPerUser` было добавлено в проигрыватель Flash Player (10.0.32) и AIR (1.5.2) для устранения конфликтов, которые возникали, если в компьютер Mac выполнили вход несколько пользователей. В других операционных системах это свойство игнорируется, так как локальное подключение всегда связано с отдельным пользователем. В новом коде свойству `isPerUser` необходимо присваивать значение `true`. Однако в целях обеспечения обратной совместимости по умолчанию используется значение `false`. Это значение по умолчанию может быть изменено в будущих версиях среды выполнения.

# Обмен сообщениями между двумя приложениями

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс LocalConnection используется для связи между различными приложениями AIR, а также между приложениями Adobe® Flash® Player (SWF), выполняющимися в браузере. Можно также использовать класс LocalConnection для установления связи между приложением AIR и приложением SWF, выполняющимся в браузере.

Например, на веб-странице может быть несколько экземпляров Flash Player, или же один экземпляр Flash Player может получать данные из экземпляра Flash Player во всплывающем окне.

Следующий код определяет объект LocalConnection, который выступает в качестве сервера и принимает входящие вызовы LocalConnection от других приложений:

```
package
{
    import flash.net.LocalConnection;
    import flash.display.Sprite;
    public class ServerLC extends Sprite
    {
        public function ServerLC()
        {
            var lc:LocalConnection = new LocalConnection();
            lc.client = new CustomClient1();
            try
            {
                lc.connect("conn1");
            }
            catch (error:Error)
            {
                trace("error:: already connected");
            }
        }
    }
}
```

Этот код сначала создает объект LocalConnection с именем `lc` и задает для свойства `client` объект `clientObject`. Когда другое приложение вызывает метод в этом экземпляре LocalConnection, среда выполнения производит поиск этого метода в объекте `clientObject`.

Если соединение с таким именем уже существует, возникнет исключение ArgumentError, свидетельствующее о том, что произошла ошибка подключения, так как объект уже подключен.

Каждый раз, когда экземпляр Flash Player подключается к SWF-файлу и пытается вызвать любой метод для указанного локального подключения, запрос отправляется классу, заданному в свойстве`client` класса CustomClient1.

```
package
{
    import flash.events.*;
    import flash.system.fscommand;
    import flash.utils.Timer;
    public class CustomClient1 extends Object
    {
        public function doMessage(value:String = ""):void
        {
            trace(value);
        }
        public function doQuit():void
        {
            trace("quitting in 5 seconds");
            this.close();
            var quitTimer:Timer = new Timer(5000, 1);
            quitTimer.addEventListener(TimerEvent.TIMER, closeHandler);
        }
        public function closeHandler(event:TimerEvent):void
        {
            fscommand("quit");
        }
    }
}
```

Чтобы создать сервер LocalConnection, вызовите метод `LocalConnection.connect()` и выберите уникальное имя подключения. Если уже имеется подключение с указанным имеем, выводится ошибка ArgumentError, указывающая на сбой попытки подключения, так как объект уже подключен.

Следующий сниппет иллюстрирует создание LocalConnection с именем `conn1`:

```
try
{
    connection.connect("conn1");
}
catch (error:ArgumentError)
{
    trace("Error! Server already exists\n");
}
```

При подключении к основному приложению из второстепенного необходимо сначала создать объект LocalConnection в отправляющем объекте LocalConnection, затем вызвать метод `LocalConnection.send()` с именем соединения и именем метода, который нужно выполнить. Например, для отправки метода `doQuit` в объект LocalConnection, созданный ранее, используйте следующий код:

```
sendingConnection.send("conn1", "doQuit");
```

Этот код подключается к существующему объекту LocalConnection с именем соединения `conn1` и вызывает метод `doMessage()` в удаленном приложении. Если необходимо отправить параметры удаленному приложению, укажите дополнительные аргументы после имени метода в методе `send()`, как показано в следующем сниппете:

```
sendingConnection.send("conn1", "doMessage", "Hello world");
```

# Подключение к содержимому в разных доменах и к другим приложениям AIR

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы разрешить установление связи только с конкретными доменами, необходимо вызвать метод `allowDomain()` или `allowInsecureDomain()` класса LocalConnection и передать список имен одного или нескольких доменов, которым предоставлен доступ к данному объекту LocalConnection.

В предыдущих версиях ActionScript методы обратного вызова `LocalConnection.allowDomain()` и `LocalConnection.allowInsecureDomain()` должны были реализовываться разработчиками и возвращать логическое значение. В ActionScript 3.0 методы `LocalConnection.allowDomain()` и `LocalConnection.allowInsecureDomain()` являются встроенными. Разработчики могут вызывать их так же, как методы `Security.allowDomain()` и `Security.allowInsecureDomain()`, передавая одно или несколько имен разрешенных доменов.

В проигрыватель Flash Player 8 были внесены ограничения в сфере безопасности локальных SWF-файлов. SWF-файл, которому разрешен доступ к Интернету, не может иметь доступ к локальной файловой системе. Если задать значение `localhost`, SWF-файл будет доступен любому локальному SWF-файлу. Если метод `LocalConnection.send()` попытается установить связь с SWF-файлом из изолированной программной среды, не доступной для вызывающего кода, отправляется событие `securityError` (`SecurityErrorEvent.SECURITY_ERROR`). Во избежание этой ошибки можно указать домен вызывающей стороны в методе `LocalConnection.allowDomain()` получателя.

Методам `LocalConnection.allowDomain()` и `LocalConnection.allowInsecureDomain()` можно передать два специальных значения: `*` и `localhost`. Значение «звездочка» `(*)` предоставляет доступ из всех доменов. Строка `localhost` разрешает вызов приложения содержимым, установленным на локальном компьютере, но за пределами каталога ресурса приложения.

При попытке метода `LocalConnection.send()` установить связь с приложением, находящимся в изолированной программной среде безопасности, к которой вызывающий код не имеет доступа, отправляется событие `securityError` (`SecurityErrorEvent.SECURITY_ERROR`). Во избежание этой ошибки можно указать домен вызывающей стороны в методе `LocalConnection.allowDomain()` получателя.

При установлении связи только между содержимым в одном домене можно указать параметр `connectionName`, который не начинается с символа нижнего подчеркивания (_) и не указывает доменное имя (например, `myDomain:connectionName`). Используйте ту же строку в команде `LocalConnection.connect(connectionName)`.

При установлении связи между содержимым в разных доменах указывается параметр `connectionName`, который начинается с нижнего подчеркивания. Использование нижнего подчеркивания улучшает перемещаемость содержимого с принимающим объектом LocalConnection между доменами. Вот два возможных случая:

- Если строка для `connectionName` не начинается с нижнего подчеркивания, среда выполнения добавляет префикс с именем супердомена и двоеточие (например, `myDomain:connectionName`). Хотя это обеспечивает отсутствие конфликтов соединения с одноименными соединениями из других доменов, каждый отправляющий объект LocalConnection должен указывать это имя супердомена (например, `myDomain:connectionName`). При перемещении файла HTML или SWF с принимающим объектом LocalConnection в другой домен среда выполнения изменяет префикс в соответствии с новым супердоменом (например, `anotherDomain:connectionName`). Все отправляющие объекты LocalConnection необходимо изменить вручную в соответствии с новым супердоменом.

- Если строка для `connectionName` начинается с нижнего подчеркивания (например, `_connectionName`), среда выполнения не прибавляет к ней префикс. Это значит, что принимающие и отправляющие объекты LocalConnection используют одинаковые строки для `connectionName`. Если принимающий объект использует `LocalConnection.allowDomain()` для указания на то, что будут приняты соединения из любого домена, можно переместить файл HTML или SWF с принимающим объектом LocalConnection в другой домен без изменения отправляющих объектов LocalConnection.

  Недостатком использования имен с нижним подчеркиванием в параметре `connectionName` является возможность конфликтов, например, когда два приложения пытаются подключиться с использованием одного и того же имени `connectionName`. Второй недостаток, связанный с первым, относится к безопасности. Имена соединений с нижним подчеркиванием не идентифицируют домен прослушивающего приложения. По этой причине предпочтительными именами являются имена с указанием домена.

**Adobe AIR**

Для обращения к содержимому в изолированной среде приложения AIR (содержимому, установленному вместе с приложением AIR) необходимо использовать префикс для имени подключения, содержащий супердомен, идентифицирующий приложение AIR. Строка супердомена начинается с `app#`, далее следует идентификатор приложения, а после точки (.) — идентификатор издателя (если он определен). Например, супердомен для параметра `connectionName` для приложения с идентификатором приложения `com.example.air.MyApp` и отсутствующим идентификатором издателя выглядит следующим образом: `"app#com.example.air.MyApp"`. Таким образом, если основное подключение имеет имя appConnection, то в параметре `connectionName` необходимо использовать строку `"app#com.example.air.MyApp:appConnection"`. Если для приложения определен идентификатор издателя, то его также необходимо включить в строку супердомена: `"app#com.example.air.MyApp.B146A943FBD637B68C334022D304CEA226D129B4.1"`.

Если одному приложению AIR разрешается установить связь с другим приложением через локальное соединение, необходимо вызвать метод `allowDomain()` объекта LocalConnection, передав доменное имя локального соединения. Для приложения AIR доменное имя образуется из идентификаторов приложения и издателя тем же образом, что и строка соединения. Например, если отправляющее приложение AIR имеет идентификатор приложения `com.example.air.FriendlyApp` и идентификатор издателя `214649436BD677B62C33D02233043EA236D13934.1`, то доменная строка, используемая для разрешения подключения приложения, будет выглядеть так: `app#com.example.air.FriendlyApp.214649436BD677B62C33D02233043EA236D13934.1`. (Начиная с AIR 1.5.3 идентификатор издателя присутствует не во всех приложениях AIR.)

---

# Глава 46. Связь с собственными процессами в AIR

**Adobe AIR 2 и более поздних версий**

Начиная с версии Adobe AIR 2, приложения AIR могут выполняться и связываться с другими собственными процессами с помощью командной строки. Например, приложение AIR может выполнить процесс и связаться с ним с помощью стандартных потоков ввода и вывода.

Для обмена данными с встроенными процессами упакуйте подлежащее установке приложение AIR с помощью собственной программы установки. Тип файла собственной программы установки зависит от операционной системы, для которой она создается:

- DMG-файл в ОС Mac OS.
- EXE-файл в ОС Windows.
- Пакет RPM или DEB в ОС Linux.

Такие приложения называются приложениями расширенного профиля рабочего стола. Собственную программу по установке можно создать, указав параметр `-target native` при вызове команды `-package` с помощью ADT.

### Дополнительные разделы справки

flash.filesystem.File.openWithDefaultApplication()

flash.desktop.NativeProcess

## Обзор связей с собственным процессом

**Adobe AIR 2 и более поздних версий**

Приложение AIR в расширенном профиле рабочего стола может выполнять файл, как если бы он был вызван из командной строки. Оно может обмениваться данными со стандартными потоками собственного процесса. Стандартные потоки включают стандартный поток ввода (stdin), поток вывода (stdout) и стандартный поток ошибок (stderr).

***Примечание.*** *Приложения в расширенном профиле рабочего стола могут также запускать файлы и приложения с помощью метода `File.openWithDefaultApplication()`. Однако использование этого метода не дает приложению AIR доступ к стандартным потокам. Дополнительные сведения см. в разделе ««Открытие файлов с помощью выбранной по умолчанию системной программы» на странице 732»*

В следующем примере кода показано, как запускается программа test.exe в каталоге приложений. Эта программа передает аргумент `"hello"` как аргумент командной строки и добавляет прослушиватель событий к стандартному потоку вывода процесса:

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
var processArgs:Vector.<String> = new Vector.<String>();
processArgs.push("hello");
nativeProcessStartupInfo.arguments = processArgs;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_OUTPUT_DATA, onOutputData);
process.start(nativeProcessStartupInfo);
public function onOutputData(event:ProgressEvent):void
{
    var stdOut:ByteArray = process.standardOutput;
    var data:String = stdOut.readUTFBytes(process.standardOutput.bytesAvailable);
    trace("Got: ", data);
}
```

# Запуск и закрытие собственного процесса

**Adobe AIR 2 и более поздних версий**

Чтобы запустить собственный процесс, настройте объект NativeProcessInfo, чтобы он выполнял следующее:

- указывал на файл, который требуется запустить;
- сохранял аргументы командной строки для передачи процессу при запуске (необязательно)
- задавал рабочий каталог процесса (необязательно).

Чтобы запустить собственный процесс, передайте объект NativeProcessInfo в качестве параметра метода `start()` для объекта NativeProcess.

Например, в следующем коде показано, как запускается программа test.exe в каталоге приложений. Приложение передает аргумент `"hello"` и выбирает каталог документов пользователя в качестве рабочего:

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
var processArgs:Vector.<String> = new Vector.<String>();
processArgs[0] = "hello";
nativeProcessStartupInfo.arguments = processArgs;
nativeProcessStartupInfo.workingDirectory = File.documentsDirectory;
process = new NativeProcess();
process.start(nativeProcessStartupInfo);
```

Чтобы завершить этот процесс, вызовите метод `exit()` объекта NativeProcess.

Если необходимо, чтобы файл в установленном приложении являлся исполняемым, то при создании пакета приложения убедитесь, что он является исполняемым в данной файловой системе. (В ОС Mac и Linux для установки флага executable можно использовать команду chmod.)

# Связь с собственным процессом

**Adobe AIR 2 и более поздних версий**

После того как приложение AIR запустило собственный процесс, оно может связаться со стандартными потоками ввода, вывода и ошибок процесса.

Чтение и запись данных в потоки выполняется с помощью следующих свойств объекта NativeProcess:

- `standardInput` — содержит доступ к данным стандартного потока ввода.
- `standardOutput` — содержит доступ к данным стандартного потока вывода.
- `standardError` — содержит доступ к данным стандартного потока ошибок.

### Запись в стандартный поток ввода

Записать данные в стандартный поток ввода можно с помощью методов записи для свойства `standardInput` объекта NativeProcess. По мере того как приложение AIR записывает данные в процесс, объект NativeProcess отправляет события `standardInputProgress`.

Если происходит ошибка при записи в стандартный поток ввода, объект NativeProcess отправляет событие `ioErrorStandardInput`.

Поток ввода можно закрыть, вызвав метод `closeInput()` объекта NativeProcess. Когда поток ввода закрывается, объект NativeProcess отправляет событие `standardInputClose`.

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.start(nativeProcessStartupInfo);
process.standardInput.writeUTF("foo");
if(process.running)
{
    process.closeInput();
}
```

### Чтение из стандартного потока вывода

Данные можно считать из стандартного потока вывода с помощью методов чтения этого свойства. По мере того как приложение AIR получает данные потока вывода из процесса, объект NativeProcess отправляет события `standardOutputData`.

Если происходит ошибка при записи в стандартный поток вывода, объект NativeProcess отправляет событие `standardOutputError`.

Когда процесс закрывает поток вывода, объект NativeProcess отправляет событие `standardOutputClose`.

При чтении данных из стандартного потока ввода убедитесь, что данные читаются в порядке их создания. Другими словами, добавьте прослушиватель событий для события `standardOutputData`. В прослушивателе событий `standardOutputData` прочитайте данные из свойства `standardOutput` объекта NativeProcess. Для чтения всех данных не дожидайтесь появления события `standardOutputClose` или `exit`. Если данные не читаются по мере их создания собственным процессом, буфер может заполниться или данные могут быть потеряны. Заполненный буфер может стать причиной остановки собственного процесса при попытке записи дополнительных данных. Однако, если прослушиватель событий не зарегистрирован для события `standardOutputData`, буфер не будет заполняться и процесс не будет останавливаться. В этом случае доступ к данным будет отсутствовать.

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_OUTPUT_DATA, dataHandler);
process.start(nativeProcessStartupInfo);
var bytes:ByteArray = new ByteArray();
function dataHandler(event:ProgressEvent):void
{
    bytes.writeBytes(process.standardOutput.readBytes(process.standardOutput.bytesAvailable);
}
```

**Чтение из стандартного потока ошибок**

Данные можно считать из стандартного потока ошибок с помощью методов чтения этого свойства. Поскольку приложение AIR считывает данные потока вывода из процесса, объект NativeProcess отправляет события `standardErrorData`.

Если происходит ошибка при записи в стандартный поток ошибок, объект NativeProcess отправляет событие `standardErrorIoError`.

Когда процесс закрывает поток ошибок, объект NativeProcess отправляет событие `standardErrorClose`.

При чтении данных из стандартного потока ошибок убедитесь, что данные читаются в порядке их создания. Другими словами, добавьте прослушиватель событий для события `standardErrorData`. В прослушивателе событий `standardErrorData` прочитайте данные из свойства `standardError` объекта NativeProcess. Для чтения всех данных не дожидайтесь появления события `standardErrorClose` или `exit`. Если данные не читаются по мере их создания собственным процессом, буфер может заполниться или данные могут быть потеряны. Заполненный буфер может стать причиной остановки собственного процесса при попытке записи дополнительных данных. Однако, если прослушиватель событий не зарегистрирован для события `standardErrorData`, буфер не будет заполняться и процесс не будет останавливаться. В этом случае доступ к данным будет отсутствовать.

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_ERROR_DATA, errorDataHandler);
process.start(nativeProcessStartupInfo);
var errorBytes:ByteArray = new ByteArray();
function errorDataHandler(event:ProgressEvent):void
{
    bytes.writeBytes(process.standardError.readBytes(process.standardError.bytesAvailable);
}
```

# Вопросы безопасности при связи с собственным процессом

**Adobe AIR 2 и более поздних версий**

В API-интерфейсе исходного процесса можно запускать любые исполняемые файлы в системе пользователя. Соблюдайте крайнюю осторожность при создании и выполнении команд. Если любая часть используемой команды исходит от внешнего источника, тщательно проанализируйте безопасность выполнения такой команды. Подобным образом, приложение AIR должно проверять все данные, передаваемые запущенному процессу.

Однако проверка ввода может представлять определенную трудность. Чтобы избежать таких трудностей, лучше написать собственное приложение (например, EXE-файл в ОС Windows), со специальными API-интерфейсами. Эти API-интерфейсы должны обрабатывать только те команды, которые заданы приложением. К примеру, приложения могут принимать только ограниченный набор инструкций посредствам стандартного потока ввода.

Среда выполнения AIR в ОС Windows не разрешает непосредственное выполнение файлов .bat. Приложение интерпретатора команд (cmd.exe) выполняет файлы .bat ОС Windows. При вызове файла .bat это приложение интерпретирует аргументы, передаваемые команде, как дополнительные приложения, которые требуется запустить. Вредоносная вставка лишних символов в строку аргумента может привести к выполнению приложением cmd.exe вредоносного или небезопасного приложения. Например, без надлежащей проверки данных приложение AIR может вызвать `myBat.bat myArguments c:/evil.exe`. В результате этого приложение cmd.exe вместе с файлом .bat запустит приложение evil.exe.

# Глава 47. Использование внешнего API

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Внешний API-интерфейс ActionScript 3.0 (flash.external.ExternalInterface) позволяет ActionScript и приложению контейнера, в котором запущен проигрыватель Adobe Flash Player, просто обмениваться данными. Используйте API-интерфейс ExternalInterface, чтобы создать взаимодействие между SWF-документом и JavaScript на HTML-странице.

Можно использовать внешний API-интерфейс для взаимодействия с приложением контейнера, передавать данные между ActionScript и JavaScript на HTML-странице.

Некоторые общие задачи внешнего API-интерфейса включают:

- получение информации о приложении контейнера;
- Использование ActionScript для вызова кода на веб-странице, отображаемой в браузере или в программе AIR на компьютере
- Вызов кода ActionScript с веб-страницы
- Создание прокси-сервера, упрощающего вызов кода ActionScript с веб-страницы

***Примечание.*** *В этом описании внешнего интерфейса охватывается только связь между кодом ActionScript в SWF-файле и приложением контейнера, которое содержит ссылку на проигрыватель Flash Player или экземпляр, в который загружен SWF-файл. Любое другое использование Flash Player в приложении находится вне области действия этой документации. Flash Player разработан для использования в качестве подключаемого модуля веб-обозревателя или проектора (автономное приложение). Другие сценарии применения могут иметь ограниченную поддержку.*

**Использование внешнего API в среде AIR**

Поскольку в приложении AIR отсутствует внешний контейнер, как правило, этот внешний интерфейс не применяется и не требуется. Когда приложение AIR загружает SWF-файл напрямую, код приложения может напрямую обращаться к коду ActionScript в SWF-файле (в соответствии с ограничениями изолированной программной среды).

Однако, когда приложение AIR загружает SWF-файл с использованием страницы HTML в объект HTMLLoader (или компонент HTML в среде Flex), объект HTMLLoader служит внешним контейнером. Поэтому можно использовать внешний интерфейс для связи кода ActionScript в загруженном SWF-файле с кодом JavaScript на странице HTML, загруженной в объект HTMLLoader.

# Основы использования внешнего API

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Хотя в некоторых случаях SWF-файл может выполняться самостоятельно (например, если вы используете Adobe® Flash® Professional при создании проектора SWF-файлов), в большинстве ситуаций приложение SWF выполняется как элемент внутри другого приложения. Как правило, контейнер, который включает SWF-файл, является файлом HTML. Несколько менее часто SWF-файл используется для всего пользовательского интерфейса приложения для настольной системы или для ее части.

Так как вы работаете над более сложными приложениями, возможно, потребуется установить связь между SWF-файлом и приложением-контейнером. Например, зачастую на веб-странице текст и другая информация показаны в виде HTML, а SWF-файл включен для показа динамического визуального содержимого (например, диаграммы или видеофайла). В данном случае может быть необходимо сделать так, что если пользователи нажимают кнопку на веб-странице, что-то изменяется в SWF-файле. ActionScript содержит механизм, который называется внешним API и который упрощает этот тип связи между ActionScript в SWF-файле и другим кодом в приложении контейнера.

### Важные понятия и термины

Ниже приводится список важных терминов, необходимых при работе.

**Приложение контейнера** Приложение, в котором Flash Player выполняет SWF-файл, такое как веб-браузер или HTML-страница, которая включает содержимое Flash Player или приложение AIR, загружающее SWF на страницу.

**Проектор** Исполняемый файл, включающий в себя SWF-содержимое и внедренную версию Flash Player. Вы можете создать файл проектора, используя для этого Flash Professional или автономный Flash Player. Проекторы обычно применяются для распределения SWF-файлов по компакт-дискам или в иных ситуациях, когда размер загрузки неважен, а автор SWF-файла хочет быть уверен, что пользователь сможет выполнить SWF-файл независимо от того, установлен ли Flash Player на компьютере пользователя.

**Прокси** Приложение-посредник или код, который вызывает код в одном приложении («внешнем приложении») со стороны другого приложения («вызывающего приложения») и возвращает значения вызывающему приложению. Прокси можно использовать в различных целях.

- Чтобы упростить процесс выполнения вызовов внешней функции путем преобразования исходных вызовов функции в формат, который понимается внешним приложением.
- Чтобы обойти ограничения безопасности или другие ограничения, которые не позволяют вызывающему напрямую связываться с внешним приложением.

**Сериализация** Преобразование объектов или значений данных в формат, который может использоваться для передачи значений в сообщениях между двумя системами программирования (например, по Интернету или между двумя приложениями, запущенными на одном и том же компьютере).

### Работа с примерами

Многие приведенные примеры кода являются небольшими примерами, созданными в демонстрационных целях, а не полными рабочими примерами кода, проверяющего значения. Поскольку использование внешнего API требует (по определению) составления кода ActionScript, а также кода в приложении контейнера, проверка примеров включает создание контейнера (например, веб-страницы, содержащей SWF-файл) и использование распечаток кода для взаимодействия с контейнером.

**Чтобы проверить пример связи между ActionScript и JavaScript, выполните следующие действия.**

1 Создайте новый документ в среде Flash Professional и сохраните его на компьютере.

2 В основном меню выберите «Файл» > «Параметры публикации».

3 В диалоговом окне «Параметры публикации» на вкладке «Форматы» убедитесь в том, что установлены флажки Flash и HTML.

4 Нажмите кнопку «Опубликовать». Это приводит к созданию SWF-файла и HTML-файла в той же папке и с тем же именем, которое использовалось для сохранения документа Чтобы закрыть диалоговое окно «Параметры публикации», нажмите кнопку «OK».

5 Снимите флажок «HTML». После того, как HTML-страница создана, необходимо изменить ее, чтобы добавить соответствующий код JavaScript. Снятие флажка «HTML» приводит к тому, что после изменения HTML-страницы Flash не станет перезаписывать ваши изменения, показывая новую HTML-страницу при публикации SWF-файла.

6 Чтобы закрыть диалоговое окно «Параметры публикации», нажмите кнопку «OK».

7 Откройте в текстовом редакторе или редакторе HTML файл HTML, созданный Flash при публикации SWF-файла. В исходном коде HTML добавьте начальный и завершающий тег `script`, а затем скопируйте в них код JavaScript из примера кода:

```
<script>
// add the sample JavaScript code here
</script>
```

8 Сохраните файл HTML и вернитесь в программу Flash.

9 Выберите ключевой кадр в Кадре 1 временной шкалы и откройте панель «Действия».

10 Скопируйте код ActionScript на панель «Сценарий».

11 Выберите в основном меню «Файл» > «Опубликовать», чтобы обновить SWF-файл, внеся в него изменения.

12 Откройте с помощью веб-обозревателя HTML-страницу, которую вы отредактировали, для просмотра страницы и проверьте связь между ActionScript и страницей HTML.

**Чтобы проверить пример связи между контейнерами ActionScript, выполните следующие действия.**

1 Создайте новый документ в среде Flash Professional и сохраните его на компьютере. Его можно сохранить в той папке, где ваше приложение контейнера обычно ищет SWF-файл.

2 В основном меню выберите «Файл» > «Параметры публикации».

3 В диалоговом окне «Параметры публикации» на вкладке «Форматы» убедитесь в том, что установлен только флажок Flash.

4 В поле «Файл», расположенном возле флажка «Flash», щелкните значок папки, чтобы выбрать папку, в которой следует опубликовать SWF-файл. Выбрав местоположение своего SWF-файла; можно, например, сохранить документ Flash в одной папке, а опубликованный SWF-файл — в другой папке (допустим, в папке, содержащей исходный код приложения контейнера).

5 Выберите ключевой кадр в Кадре 1 временной шкалы и откройте панель «Действия».

6 Скопируйте код ActionScript из примера на панель «Сценарий».

7 Выберите в основном меню «Файл» > «Опубликовать», чтобы заново опубликовать SWF-файл.

8 Создайте и выполните свое приложение контейнера, чтобы проверить связь между ActionScript и приложением контейнера.

Полный список примеров использования внешнего API-интерфейса для взаимодействия с HTML-страницей см. в следующей теме:

- «Пример внешнего API: связь между ActionScript и JavaScript в веб-обозревателе» на странице 911

Эти примеры содержат полный код, включая ActionScript и код проверки контейнера, который необходимо применять при создании кода с помощью внешнего API. Еще один полноценный случай с применением внешнего API приведен в примере класса ExternalInterface из справочника ActionScript® 3.0 для Adobe® Flash® Professional CS5.

# Требования к внешнему API и преимущества

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Внешний API является частью ActionScript, которая обеспечивает механизм для связи между ActionScript и кодом, запущенным во «внешнем приложении», который выступает в роли контейнера для Flash Player (обычно это веб-обозреватель или автономное приложение проектора). В ActionScript 3.0 функциональные возможности внешнего API обеспечиваются классом ExternalInterface. В версиях, появившихся перед версией Flash Player 8, действие `fscommand()` использовалось для осуществления связи с приложением контейнера. Класс ExternalInterface является заменой функции `fscommand()`.

***Примечание.*** *Если потребуется воспользоваться старой функцией `fscommand()` (например, для сохранения совместимости со старыми приложениями или взаимодействия со сторонним приложением контейнера SWF или автономным проигрывателем Flash Player), она остается доступна на уровне пакета в пакете flash.system.*

Класс ExternalInterface представляет собой подсистему, которая обеспечивает простое взаимодействие ActionScript и Flash Player с JavaScript на HTML-странице.

Класс ExternalInterface доступен только при следующих условиях:

- во всех поддерживаемых версиях Internet Explorer для Windows (5.0 и выше);
- в любом обозревателе, который поддерживает интерфейс NPRuntime, включая на данный момент версии, начиная с Firefox 1.0, Mozilla 1.7.5, Netscape 8.0 и Safari 1.3.
- в приложении AIR, когда SWF-файл встроен в HTML-страницу, отображаемую элементом управления HTMLLoader.

Во всех остальных ситуациях (например, при запуске в автономном проигрывателе) свойство `ExternalInterface.available` возвращает значение `false`.

Из ActionScript можно вызвать функцию JavaScript на странице HTML. Внешний API предлагает следующие усовершенствованные функции по сравнению с `fscommand()`:

- Можно воспользоваться любой функцией JavaScript, а не только функциями, которые можно применять вместе с функцией `fscommand()`.
- Можно передать любое количество аргументов с любыми именами, не ограничиваясь передачей команды и однострочного аргумента. Это придает внешнему API гораздо больше гибкости, чем методу `fscommand()`.
- Передавать можно различные типы данных (например, Boolean, Number и String), а не только параметры String.
- Можно получить значение вызова, и это значение немедленно вернется в ActionScript (как возвращаемое значение выполняемого вызова).

***Важная информация.*** *Если имя, присвоенное экземпляру Flash Player на странице HTML (атрибут `id` для тега `object`), включает дефис (`-`) или другие символы, определенные в качестве операторов в JavaScript (например, `+`, `*`, `/`, `\`, `.` и так далее), вызовы ExternalInterface из ActionScript при просмотре веб-страницы контейнера обозревателем Internet Explorer работать не будут. Кроме того, если теги HTML, определяющие экземпляр Flash Player (теги `object` и `embed`), вложены в тег HTML `form`, вызовы ExternalInterface из ActionScript работать не будут.*

# Использование класса ExternalInterface

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Обмен данными между ActionScript и приложением контейнера может принимать две формы: ActionScript может вызвать код (как в функции JavaScript), определенный в контейнере, либо код в контейнере может вызвать функцию ActionScript, которая была разработана с учетом возможности вызова. В любом случае информацию можно отправить к вызываемому коду, а результаты вернуть коду, который совершил вызов.

Чтобы упростить обмен данными, класс ExternalInterface включает два статических свойства и два статических метода. Эти свойства и методы используются для получения информации о подключении внешнего интерфейса, для выполнения кода в контейнере из ActionScript и для того, чтобы функции ActionScript можно было бы вызвать с помощью контейнера.

## Получение сведений о внешнем контейнере

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Свойство `ExternalInterface.available` указывает, находится ли текущий Flash Player в контейнере, имеющем внешний интерфейс. Если внешний интерфейс имеется, свойство принимает значение `true`; в противном случае — `false`. Перед использованием любых других функций в классе ExternalInterface необходимо всегда проверять, что текущий контейнер поддерживает обмен данными с внешним интерфейсом. Это делается следующим образом:

```
if (ExternalInterface.available)
{
    // Perform ExternalInterface method calls here.
}
```

***Примечание.*** *Свойство `ExternalInterface.available` говорит о том, принадлежит ли текущий контейнер типу, поддерживающему возможность подключения ExternalInterface. Оно не информирует о том, включен ли JavaScript в текущем обозревателе.*

Свойство `ExternalInterface.objectID` позволяет определить уникальный идентификатор экземпляра Flash Player (особенно атрибут `id` тега `object` в Internet Explorer или атрибут `name` тега `embed` в обозревателях с помощью интерфейса NPRuntime). Этот уникальный идентификатор представляет текущий SWF-документ в обозревателе. Его можно использовать для ссылок на SWF-документ (например, при вызове функции JavaScript на странице HTML контейнера). Когда контейнер Flash Player не является веб-обозревателем, этому свойству присваивается значение `null`.

## Вызов внешнего кода из ActionScript

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `ExternalInterface.call()` выполняет код в приложении контейнера. Он требует наличия по крайней мере одного параметра (строки, содержащей имя функции, которую необходимо вызвать в приложении контейнера). Любые дополнительные параметры, переданные методу `ExternalInterface.call()`, передаются в контейнер как параметры вызова функции.

```
// calls the external function "addNumbers"
// passing two parameters, and assigning that function's result
// to the variable "result"
var param1:uint = 3;
var param2:uint = 7;
var result:uint = ExternalInterface.call("addNumbers", param1, param2);
```

Если контейнер является страницей HTML, этот метод запускает функцию JavaScript с указанным именем, которая должна быть определена в элементе `script` на той странице HTML, где он содержится. Возвращаемое значение функции JavaScript передается обратно в ActionScript.

```
<script language="JavaScript">
    // adds two numbers, and sends the result back to ActionScript
    function addNumbers(num1, num2)
    {
        return (num1 + num2);
    }
</script>
```

Если контейнер находится в каком-либо другом контейнере ActiveX, этот метод заставляет элемент управления ActiveX для Flash Player отправить событие `FlashCall`. Flash Player преобразует имя функции и имеющиеся параметры в строку XML. Контейнер может выполнять доступ к этой информации в свойстве `request` объекта события и использовать его для определения метода выполнения собственного кода. Чтобы вернуть значение в ActionScript, код контейнера вызывает метод `SetReturnValue()` объекта ActiveX, передавая результат (последовательно представленный в виде строки XML) в качестве параметра этого метода. Дополнительные сведения о формате XML, использованном для этой связи, см. в разделе ««Формат XML внешнего API» на странице 910».

Независимо от того, является ли контейнер веб-обозревателем или другим контейнером ActiveX, если вызов неудачен или метод контейнера не указывает возвращаемое значение, возвращается `null`. Метод `ExternalInterface.call()` выдает исключение SecurityError, если содержащая его среда принадлежит к изолированной программной среде, к которой не имеет доступа вызывающий код. Эту проблему можно обойти, установив соответствующее значение `allowScriptAccess` в среде, содержащей метод. Например, чтобы изменить значение `allowScriptAccess` на странице HTML, вам необходимо изменить соответствующий атрибут в тегах `object` и `embed`.

## Вызов кода ActionScript из контейнера

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Контейнер может вызвать только тот код ActionScript, который находится в функции — он не может вызвать никакой другой код ActionScript. Чтобы вызвать функцию ActionScript из приложения контейнера, необходимо сделать две вещи: зарегистрировать функцию в классе ExternalInterface, а затем вызвать ее из кода контейнера.

Во-первых, необходимо зарегистрировать вашу функцию ActionScript, чтобы указать, что она должна быть доступна контейнеру. Воспользуйтесь методом `ExternalInterface.addCallback()` следующим образом:

```
function callMe(name:String):String
{
    return "busy signal";
}
ExternalInterface.addCallback("myFunction", callMe);
```

Метод `addCallback()` принимает два параметра. Первый из них — это имя функции в виде объекта String. Это имя, под которым функция будет известна в контейнере. Второй параметр — это фактическая функция ActionScript, которая будет выполняться при вызове контейнером определенного имени функции. Поскольку это разные имена, можно указать имя функции, которое будет использоваться в качестве контейнера, даже если фактическая функция ActionScript имеет другое имя. Это особенно полезно, если имя функции неизвестно (например, если указывается анонимная функция или если подлежащая вызову функция определяется во время выполнения).

После регистрации функции ActionScript в классе ExternalInterface контейнер может вызвать эту функцию. Метод вызова зависит от типа контейнера. Например, код JavaScript в веб-обозревателе вызывает функцию ActionScript с помощью зарегистрированного имени функции, как если бы это был метод объекта обозревателя Flash Player (то есть, метод объекта JavaScript, отражающий тег `object` или `embed`). Иными словами, передаются параметры, а результат возвращается так, как если бы вызывалась локальная функция.

```
<script language="JavaScript">
    // callResult gets the value "busy signal"
    var callResult = flashObject.myFunction("my name");
</script>
...
<object id="flashObject"...>
    ...
    <embed name="flashObject".../>
</object>
```

Кроме того, при вызове функции ActionScript в SWF-файле, запущенном в приложении для настольных систем, зарегистрированное имя функции и любые параметры должны быть представлены в виде строки формата XML. Затем вызов выполняется путем обращения к метолу `CallFunction()` элемента управления ActiveX, где в качестве параметра выступает строка XML. Дополнительные сведения о формате XML, использованном для этой связи, см. в разделе ««Формат XML внешнего API» на странице 910».

В любом случае значение, возвращаемое функцией ActionScript, передается обратно в код контейнера. Передача осуществляется напрямую в виде значения, когда вызывающей стороной является код JavaScript в обозревателе, либо в виде значения в формате XML, когда вызов выполняется контейнером ActiveX.

## Формат XML внешнего API

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Связь между ActionScript и приложением, в котором находится элемент управления ActiveX для Shockwave Flash, реализуется с помощью формата XML, используемого для кодирования вызовов функции и значений. Существует две части формата XML, использованного внешним API. Один формат применяется для представления вызовов функции. Другой формат применяется для представления отдельных значений. Этот параметр используется для обозначения параметров в функциях, а также значений, возвращаемых функциями. Формат XML для вызовов функций применяется для вызовов кода ActionScript и из самого кода ActionScript. При вызове функции из ActionScript Flash Player передает XML в контейнер. Если вызов осуществляется из контейнера, Flash Player ожидает, что приложение контейнера передаст строку XML в этом формате. В следующем фрагменте XML показан пример вызова функции в формате XML:

```
<invoke name="functionName" returntype="xml">
    <arguments>
        ... (individual argument values)
    </arguments>
</invoke>
```

Корневой узел носит имя `invoke`. У него есть два атрибута: `name` указывает на имя вызываемой функции, а `returntype` всегда присваивается значение `xml`. Если вызов функции включает параметры, узел `invoke` имеет дочерний узел `arguments`, чьи дочерние узлы являются значениями параметров, имеющими формат отдельных значений, как указано ниже.

Отдельные значения, включая параметры функций и возвращаемые ими значения, используют схему форматирования, которая включает сведения о типе данных помимо фактических значений. В следующей таблице перечислены классы ActionScript. Для кодирования значений с этим типом данных применяется формат XML.

| **Класс/значение ActionScript** | **Класс/значение C#** | **Формат** | **Комментарии** |
|---|---|---|---|
| null | null | <null/> | |
| Boolean `true` | bool true | <true/> | |
| Boolean `false` | bool false | <false/> | |
| String | string | <string>строковое значение</string> | |
| Number, int, uint | single, double, int, uint | `<number>27.5</number>`<br>`<number>-12</number>` | |

| Класс/значение ActionScript | Класс/значение C# | Формат | Комментарии |
|---|---|---|---|
| Массив (элементы могут быть смешанных типов) | Коллекция, способная включать элементы смешанных типов (например, ArrayList или object[]) | `<array>`<br>`<property id="0">`<br>`<number>27.5</number>`<br>`</property>`<br>`<property id="1">`<br>`<string>Hello there!</string>`<br>`</property>`<br>`...`<br>`</array>` | Узел `property` определяет отдельные элементы, а атрибут `id` является числовым указателем, начинающимся с нуля. |
| Object | Словарь со строковыми ключами и значениями объектов (например, HashTable со строковыми ключами) | `<object>`<br>`<property id="name">`<br>`<string>John Doe</string>`<br>`</property>`<br>`<property id="age">`<br>`<string>33</string>`<br>`</property>`<br>`...`<br>`</object>` | Узел `property` определяет отдельные свойства, а атрибут `id` является именем свойства (строкой). |
| Другие встроенные или заказные классы | | `<null/> or`<br>`<object></object>` | ActionScript кодирует другие объекты как нулевые или пустые. В любом случае все значения свойств теряются. |

***Примечание.*** *В качестве примера в этой таблице указаны эквивалентные классы C# помимо классов ActionScript. Однако внешний API можно использовать для связи с любым языком программирования или средой выполнения, которая поддерживает элементы управления ActiveX и не ограничивается приложениями на C#.*

При составлении собственных приложений с применением внешнего API и приложения контейнера ActiveX, вы, возможно, сочтете удобным создание прокси-сервера, который осуществит задачу преобразования собственных вызовов функции в последовательный формат XML. Пример класса прокси-сервера, написанного на C#, представлен в разделе Подробно о классе ExternalInterfaceProxy.

# Пример внешнего API: связь между ActionScript и JavaScript в веб-обозревателе

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Этот образец приложения демонстрирует применение подходящих методов обмена данными между ActionScript и JavaScript в веб-обозревателе в контексте приложения для мгновенного обмена сообщениями, которое позволяет человеку участвовать в чате с самим собой (отсюда и имя приложения: Introvert IM). Пересылка сообщений между формой HTML на веб-странице и интерфейсом SWF выполняется с помощью внешнего API. В этом примере показаны, например, следующие методы:

- Правильная инициация обмена данными путем проверки готовности обозревателя к связи перед установкой подключения
- Проверка наличия у контейнера поддержки внешнего API
- Вызов функций JavaScript из ActionScript, передача параметров и получение в ответ значений

- Предоставление возможности вызова с помощью JavaScript методов ActionScript и выполнение этих вызовов

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения Introvert IM находятся в папке Samples/IntrovertIM_HTML. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| IntrovertIMApp.fla<br>или<br>IntrovertIMApp.mxml | Основной файл приложения в формате Flash (FLA) или Flex (MXML). |
| com/example/programmingas3/introvertIM/IMManager.as | Класс, который устанавливает связь между ActionScript и контейнером и управляет ею. |
| com/example/programmingas3/introvertIM/IMMessageEvent.as | Заказной тип события, отправляемого классом IMManager при получении сообщения из контейнера. |
| com/example/programmingas3/introvertIM/IMStatus.as | Перечислимый тип, значения которого отражают разные значения состояния «доступности», которые можно выбрать в приложении. |
| html-flash/IntrovertIMApp.html<br>или<br>html-template/index.template.html | HTML-страница приложения для Flash (html-flash/IntrovertIMApp.html) или шаблон, который использовался для создания HTML-страницы приложения для Adobe Flex (html-template/index.template.html). Этот файл содержит все функции JavaScript, которые делают контейнер частью приложения. |

## Подготовка связи между ActionScript и обозревателем

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Одно из наиболее частых применений внешнего API состоит в том, чтобы позволять приложениям ActionScript связываться с обозревателем. С помощью внешнего API методы ActionScript могут вызывать код, написанный на языке JavaScript, и наоборот. Из-за сложности обозревателей и их метода внутренней визуализации страниц не существует метода, гарантирующего, что документ SWF зарегистрирует обратные вызовы перед тем, как будет запущен первый код JavaScript на HTML-странице. По этой причине перед вызовом функций в SWF-документе из JavaScript ваш SWF-документ должен всегда вызывать HTML-страницу, чтобы уведомить ее о том, что SWF-документ готов к приему подключений.

Например, выполнив ряд действий с применением класса IMManager, Introvert IM определяет, готов ли обозреватель для связи, и подготовить SWF-файл для связи. Первый шаг состоит в определении времени, когда обозреватель готов для связи. Он выполняется в конструкторе IMManager следующим образом:

```
public function IMManager(initialStatus:IMStatus)
{
    _status = initialStatus;

    // Check if the container is able to use the external API.
    if (ExternalInterface.available)
    {
        try
        {
            // This calls the isContainerReady() method, which in turn calls
            // the container to see if Flash Player has loaded and the container
            // is ready to receive calls from the SWF.
            var containerReady:Boolean = isContainerReady();
            if (containerReady)
            {
                // If the container is ready, register the SWF's functions.
                setupCallbacks();
            }
            else
            {
                // If the container is not ready, set up a Timer to call the
                // container at 100ms intervals. Once the container responds that
                // it's ready, the timer will be stopped.
                var readyTimer:Timer = new Timer(100);
                readyTimer.addEventListener(TimerEvent.TIMER, timerHandler);
                readyTimer.start();
            }
        }
        ...
    }
    else
    {
        trace("External interface is not available for this container.");
    }
}
```

Первым делом код проверяет, доступен ли внешний API в текущем контейнере с помощью свойства `ExternalInterface.available`. При этом он приступает к установке подключения. Поскольку исключения, связанные с безопасностью, и другие ошибки могут происходить при попытке связи с внешним приложением, код упакован в блок `try` (соответствующие блоки `catch` были пропущены при перечислении для краткости).

Затем код вызывает метод `isContainerReady()`:

```
private function isContainerReady():Boolean
{
    var result:Boolean = ExternalInterface.call("isReady");
    return result;
}
```

Метод `isContainerReady()` в свою очередь обращается к методу `ExternalInterface.call()`, чтобы вызвать функцию JavaScript `isReady()` следующим образом:

```
<script language="JavaScript">
<!--
// ------- Private vars -------
var jsReady = false;
...
// ------- functions called by ActionScript -------
// called to check if the page has initialized and JavaScript is available
function isReady()
{
    return jsReady;
}
...
// called by the onload event of the <body> tag
function pageInit()
{
    // Record that JavaScript is ready to go.
    jsReady = true;
}
...
//-->
</script>
```

Функция `isReady()` просто возвращает значение переменной `jsReady`. Этой переменной исходно присвоено значение `false`; после запуска события `onload` на веб-странице переменная принимает значение `true`. Другим словами, если ActionScript вызывает функцию `isReady()` перед загрузкой страницы, JavaScript возвращает `false` для `ExternalInterface.call("isReady")`, а метод ActionScript `isContainerReady()` возвращает соответственно значение `false`. После завершения загрузки страницы функция JavaScript `isReady()` возвращает значение `true`. Поэтому метод `isContainerReady()` в ActionScript также возвращает `true`.

В конструкторе IMManager происходит одно из двух событий в зависимости от готовности контейнера. Если `isContainerReady()` возвращает `true`, код просто вызывает метод `setupCallbacks()`, который завершает процесс установки связи с JavaScript. С другой стороны, если `isContainerReady()` возвращает `false`, процесс по существу откладывается. Создается объект Timer, которому дается инструкция вызывать метод `timerHandler()` раз в 100 миллисекунд:

```
private function timerHandler(event:TimerEvent):void
{
    // Check if the container is now ready.
    var isReady:Boolean = isContainerReady();
    if (isReady)
    {
        // If the container has become ready, we don't need to check anymore,
        // so stop the timer.
        Timer(event.target).stop();
        // Set up the ActionScript methods that will be available to be
        // called by the container.
        setupCallbacks();
    }
}
```

Каждый раз, когда вызывается метод `timerHandler()`, он снова проверяет результат метода `isContainerReady()`. После инициализации контейнера этот метод возвращает значение `true`. Затем код останавливает Timer и вызывает метод `setupCallbacks()`, завершая процесс установки подключения к обозревателю.

## Предоставление методов ActionScript JavaScript

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Как показано в предыдущем примере, после того, как код определяет, что обозреватель готов, вызывается метод `setupCallbacks()`. Этот метод готовит ActionScript для принятия вызовов со стороны JavaScript, как указано ниже:

```
private function setupCallbacks():void
{
    // Register the SWF client functions with the container
    ExternalInterface.addCallback("newMessage", newMessage);
    ExternalInterface.addCallback("getStatus", getStatus);
    // Notify the container that the SWF is ready to be called.
    ExternalInterface.call("setSWFIsReady");
}
```

Метод `setCallBacks()` завершает задачу подготовки связи с контейнером путем вызова метода `ExternalInterface.addCallback()` для регистрации двух методов, которые можно будет вызвать из JavaScript. В этом коде первый параметр (имя, под которым этот метод известен JavaScript (`"newMessage"` и `"getStatus"`) идентичен имени метода в ActionScript. В данном случае использование других имен не давало преимущества, поэтому для простоты повторно применялось одно и то же имя. В итоге метод `ExternalInterface.call()` использовался для вызова функции JavaScript`setSWFIsReady()`, которая уведомляет контейнер о том, что функции ActionScript были зарегистрированы.

## Связь между ActionScript и обозревателем

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Приложение Introvert IM демонстрирует целый ряд примеров вызова функций JavaScript на странице контейнера. В простейшем случае (пример из метода `setupCallbacks()`) функция `setSWFIsReady()` в JavaScript вызывается без передачи каких-либо параметров или получения в ответ значения:

```
ExternalInterface.call("setSWFIsReady");
```

В другом примере из метода `isContainerReady()` ActionScript вызывает функцию `isReady()` и получает в ответ логическое значение:

```
var result:Boolean = ExternalInterface.call("isReady");
```

Передать параметры функциям JavaScript можно также с помощью внешнего API. Рассмотрим, к примеру, метод `sendMessage()` класса IMManager, который вызывается, когда пользователь отправляет новое сообщение своему «собеседнику»:

```
public function sendMessage(message:String):void
{
    ExternalInterface.call("newMessage", message);
}
```

И снова здесь применяется метод `ExternalInterface.call()` для вызова назначенной функции JavaScript, уведомляя обозревателя о новом сообщении. Кроме того, само сообщение передается в качестве дополнительного параметра методу `ExternalInterface.call()` и соответственно передается в качестве параметра функции JavaScript`newMessage()`.

## Вызов кода ActionScript из JavaScript

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Связь предполагает двусторонний обмен, и приложение Introvert IM не является исключением. Не только клиент системы мгновенного обмена сообщениями в проигрывателе Flash Player вызывает JavaScript для отправки сообщений. HTML-форма также вызывает код JavaScript для отправки сообщений и запроса информации из SWF-файла. Например, когда SWF-файл уведомляет контейнер о том, что он закончил устанавливать контакт и готов к связи, обозреватель первым делом вызывает метод `getStatus()` класса IMManager, чтобы узнать исходное состояние доступности пользователя из клиента системы мгновенного обмена сообщениями SWF. Это выполняется на веб-странице функции `updateStatus()` следующим образом:

```
<script language="JavaScript">
...
function updateStatus()
{
    if (swfReady)
    {
        var currentStatus = getSWF("IntrovertIMApp").getStatus();
        document.forms["imForm"].status.value = currentStatus;
    }
}
...
</script>
```

Код проверяет значение переменной `swfReady`, которая следит за тем, уведомил ли SWF-файл обозреватель о том, что он зарегистрировал свои методы в классе ExternalInterface. Если SWF-файл готов к приему связи, следующая строка (`var currentStatus =` ...) фактически вызывает метод `getStatus()` в классе IMManager. В этой строке происходят три события.

- Вызывается функция `getSWF()` для JavaScript, которая возвращает ссылку на объект JavaScript, представляющий SWF-файл. Параметр, переданный `getSWF()`, определяет, какой объект обозревателя возвращается в том случае, если на HTML-странице имеется более одного SWF-файла. Значение, переданное этому параметру, должно соответствовать атрибуту `id` тега `object` и атрибуту `name` тега `embed`, используемым для включения SWF-файла.
- С помощью ссылки на SWF-файл метод `getStatus()` вызывается так, как если бы он был методом объекта SWF. В данном случае применяется имя «`getStatus`», потому что это то имя, под которым зарегистрирована функция ActionScript с помощью метода `ExternalInterface.addCallback()`.
- Метод `getStatus()` для ActionScript возвращает значение, и это значение присваивается переменной `currentStatus`, которая затем назначается в качестве содержимого (свойство `value`) текстового поля `status`.

***Примечание.*** *Если вы внимательно отслеживаете код, то, вероятно, заметили, что в исходном коде для функции `updateStatus()` строка кода, вызывающая функцию `getSWF()`, на самом деле записана следующим образом: var currentStatus = getSWF("${application}").getStatus(). Текст `${application}` является местозаполнителем в шаблоне страницы HTML. Когда Adobe Flash Builder создает конкретную страницу HTML для приложения, этот местозаполнитель заменяется тем же текстом, который используется в качестве атрибута `id` тега `object` и атрибута `name` тега `embed` (в данном примере это текст `IntrovertIMApp`). Именно это значение и ожидается функцией `getSWF()`.*

Функция `sendMessage()` для JavaScript демонстрирует передачу параметра функции ActionScript. (`sendMessage()` — это функция, которая вызывается, когда пользователь нажимает кнопку «Отправить» на HTML-странице.)

```
<script language="JavaScript">
...
function sendMessage(message)
{
    if (swfReady)
    {
        ...
        getSWF("IntrovertIMApp").newMessage(message);
    }
}
...
</script>
```

Метод `newMessage()` для ActionScript ожидает один параметр. Поэтому переменная `message` для JavaScript передается в ActionScript. Она служит в качестве параметра при вызове метода `newMessage()` в коде JavaScript.

## Определение типа обозревателя

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Из-за различий в методе доступа обозревателей к содержимому важно всегда применять JavaScript, чтобы определить выбранный пользователем обозреватель и вызвать ролик в соответствии с синтаксисом для конкретного обозревателя с помощью объекта окна или документа, как показано в функции `getSWF()` для JavaScript в этом примере:

```
<script language="JavaScript">
...
function getSWF(movieName)
{
    if (navigator.appName.indexOf("Microsoft") != -1)
    {
        return window[movieName];
    }
    else
    {
        return document[movieName];
    }
}
...
</script>
```

Если в вашем сценарии не определен тип обозревателя пользователя, пользователь может столкнуться с неожиданным поведением при воспроизведении SWF-файлов в контейнере HTML.

# Глава 48. Проверка подписи XML в среде AIR

**Adobe AIR 1.5 и более поздних версий**

Используйте классы в API-интерфейсе Adobe® AIR® XMLSignatureValidator для проверки соответствия цифровых подписей поднабору рекомендаций W3C для синтаксиса и обработки XML-подписей (http://http://www.w3.org/TR/xmldsig-core/). XML-подписи могу использоваться для проверки целостности и идентификации подписчика для данных или информации.

XML-подписи могу использоваться для подтверждения сообщений или ресурсов, загруженных приложением. Например, если приложение предоставляет услуги по подписке, может потребоваться включить условия подписки в XML-документ с подписью. Если кто-либо попытается изменить документ с условиями подписки, проверка подписи завершится с ошибкой.

Также XML-подпись можно использовать для проверки загруженных ресурсов, используемых приложением, включая подписанный манифест, содержащий дайджесты этих ресурсов. Приложение может проверить, что ресурсы не изменялись, сравнив дайджест в подписанном файле с дайджестом, рассчитанным с учетом загруженных байтов. Это особенно полезно, если загруженный ресурс является SWF-файлом или другим активным содержимым, которое требуется воспроизвести в защищенной изолированной программной среде приложения.

## Основные принципы проверки XML-подписей

**Adobe AIR 1.5 и более поздних версий**

Краткое разъяснение и примеры кодов с проверкой подписей XML см. в следующих статьях по быстрому началу работы в центре Adobe Developer Connection:

- Создание и проверка XML-подписей (Flex)
- Создание и проверка XML-подписей (Flash)

Adobe® AIR® предлагает класс XMLSignatureValidator и интерфейс IURIDereferencer для проверки XML-подписей. XML-синтаксис, приемлемый для класса XMLSignatureValidator, определяется частью набора рекомендаций W3C для синтаксиса и порядка обработки XML-подписей. (Поскольку поддерживается только часть рекомендаций, не все допустимые подписи могут быть проверены.) Adobe AIR не предлагает прикладного интерфейса программирования для создания XML-подписей.

### Классы проверки подписей XML

**Adobe AIR 1.5 и более поздних версий**

Прикладной интерфейс программирования для проверки XML-подписей содержит следующие классы:

| Пакет | Классы |
|---|---|
| flash.security | • XMLSignatureValidator<br>• IURIDereferencer (интерфейс)<br>Строковые константы XMLSignatureValidator определяются в следующих классах:<br>• ReferencesValidationSetting<br>• RevocationCheckSettings<br>• SignatureStatus<br>• SignerTrustSettings |
| flash.events | • Event<br>• ErrorEvent |

## Использование классов проверки XML-подписей

**Adobe AIR 1.5 и более поздних версий**

Чтобы использовать класс XMLSignatureValidator для проверки XML-подписи, необходимо выполнить следующие действия:

- Создать объект XMLSignatureValidator
- Обеспечить реализацию интерфейса IURIDereferencer. Объект XMLSignatureValidator вызывает метод `dereference()` интерфейса IURIDereferencer, передавая ему идентификатор ресурса URI для каждой ссылки в подписи. Метод `dereference()` должен выполнять разрешение идентификатора URI и возвращать упоминаемые в ссылках данные (которые могут находиться в том же документе, что и подпись, а также во внешнем источнике).
- Установка параметров доверенности сертификата, проверки аннулирования и ссылочной проверки для объекта XMLSignatureValidator, как требуется для приложения.
- Добавьте прослушиватели для событий `complete` и `error`.
- Вызовите метод `verify()`, передав ему подпись для проверки.
- Обработайте события `complete` и `error`, интерпретируйте результаты.

В следующем примере реализована функция `validate()`, которая проверяет достоверность XML-подписи. Свойства XMLSignatureValidator устанавливаются таким образом, что подписывающий сертификат должен находиться в хранилище доверенных сертификатов или быть связанным с сертификатом в хранилище доверенных сертификатов. В этом примере также предполагается, что в интерфейсе IURIDereferencer существует подходящий класс с именем *XMLDereferencer*.

```
private function validate( xmlSignature:XML ):void
{
    var verifier:XMLSignatureValidator = new XMLSignatureValidator();
    verifier.addEventListener(Event.COMPLETE, verificationComplete);
    verifier.addEventListener(ErrorEvent.ERROR, verificationError);
    try
    {
        verifier.uriDereferencer = new XMLDereferencer();

        verifier.referencesValidationSetting =
            ReferencesValidationSetting.VALID_IDENTITY;
        verifier.revocationCheckSetting = RevocationCheckSettings.BEST_EFFORT;
        verifier.useSystemTrustStore = true;

        //Verify the signature
        verifier.verify( xmlSignature );
    }
    catch (e:Error)
        {
            trace("Verification error.\n" + e);
        }
}

//Trace verification results
private function verificationComplete(event:Event):void

    var signature:XMLSignatureValidator = event.target as XMLSignatureValidator;
    trace("Signature status: " + signature.validityStatus + "\n");
    trace("  Digest status: " + signature.digestStatus + "\n");
    trace("  Identity status: " + signature.identityStatus + "\n");
    trace("  Reference status: " + signature.referencesStatus + "\n");
}

private function verificationError(event:ErrorEvent):void
{
    trace("Verification error.\n" + event.text);
}
```

## Процесс проверки XML-подписи

**Adobe AIR 1.5 и более поздних версий**

При вызове метода `verify()` класса XMLSignatureValidator в AIR выполняются следующие шаги:

- Среда выполнения проверяет криптографическую целостность подписи, используя открытый ключ подписывающего сертификата.
- Среда выполнения устанавливает криптографическую целостность, достоверность и надежность сертификата на основе текущих параметров объекта XMLSignatureValidator.

  Доверенность подписывающего сертификата является основой целостности процесса проверки. Проверка подписи осуществляется с помощью хорошо определенного криптографического процесса, но достоверность подписывающего сертификата не может быть оценена алгоритмически.

  В целом существует три способа принятия решения, является ли сертификат надежным:

  - Можно положиться на полномочные органы сертификации и хранилище доверенных сертификатов операционной системы.

- Можно получить напрямую от подписывающей стороны копию сертификата, другой сертификат, который служит в качестве доверенного якоря для этого сертификата, а также достаточную информацию для надежной идентификации сертификата, например открытый ключ.
- Можно запросить конечного пользователя приложения, доверяет ли он данному сертификату. Такой запрос недопустим для самозаверяющих сертификатов, поскольку идентификационная информация в таком сертификате не может быть подтверждена его происхождением.

- Среда выполнения проверяет криптографическую целостность подписанных данных.

  Подписанные данные проверяются с помощью реализованного интерфейса IURIDereferencer. Для каждой ссылки в документе подписи вызывается `dereference()` интерфейса IURIDereferencer. Данные, возвращаемые методом `dereference()`, используются для расчета дайджеста ссылки. Значение этого дайджеста сравнивается с дайджестом, записанным в документе подписи. Если дайджесты совпадают, то, значит, данные не изменялись с момента своего подписания.

  Важным условием обеспечения безопасности на основе проверки XML-подписи является принцип, что безопасно только подписанное содержимое. Для примера рассмотрим подписанный манифест, в котором перечислены файлы, входящие в пакет. Если объект XMLSignatureValidator проверяет подпись, он убеждается только в том, что не изменялся сам манифест. Данные в этих файлах не подписаны, поэтому подпись пройдет проверку даже в том случае, если упомянутые в манифесте файлы изменяются или удаляются.

  ***Примечание.*** *Для проверки файлов в таком манифесте можно рассчитать дайджест файлов данных (используя тот же алгоритм хэширования, который используется в манифесте) и сравнить результаты с дайджестом, хранимым в подписанном манифесте. В некоторых случаях необходимо также проверять наличие дополнительных файлов.*

## Интерпретация результатов проверки

**Adobe AIR 1.5 и более поздних версий**

Результаты проверки передаются в свойствах состояния объекта XMLSignatureValidator. Эти свойства могут быть считаны после того, как объект validator отправит событие *complete*. Возможны четыре следующих состояния свойств: `validityStatus`, `digestStatus`, `identityStatus` и `referencesStatus`.

### Свойство validityStatus

**Adobe AIR 1.5 и более поздних версий**

Свойство `validityStatus` сообщает об общей достоверности подписи. Свойство `validityStatus` зависит от состояния других трех свойств состояния и может принимать одно из следующих значений:

- `valid` — Если свойства `digestStatus`, `identityStatus` и `referencesStatus` все имеют значение `valid`.
- `invalid` — Если хотя бы одно из отдельных свойств состояния имеет значение `invalid`.
- `unknown` — Если хотя бы одно из отдельных свойств состояния имеет значение `unknown` и нет отдельных свойств состояния со значением `invalid`.

### Свойство digestStatus

**Adobe AIR 1.5 и более поздних версий**

Свойство `digestStatus` сообщает о результате криптографической проверки дайджеста сообщения. Свойство `digestStatus` может иметь одно из следующих значений:

- `valid` — Если сам документ подписи не изменялся с момента подписания.
- `invalid` — Если документ подписи был изменен с момента подписания.
- `unknown` — Если выполнение метода `verify()` не было завершено без ошибок.

### Свойство identityStatus

**Adobe AIR 1.5 и более поздних версий**

Свойство `identityStatus` сообщает о состоянии подписывающего сертификата. Значение этого свойства зависит от следующих факторов:

- криптографической целостности сертификата;
- не истек ли срок действия сертификата и не был ли он отозван;
- доверен ли сертификат на этом компьютере;
- состояние объекта XMLSignatureValidator (были ли добавлены дополнительные сертификаты для создания цепи подтверждения доверенности, были ли эти сертификаты доверенными, каковы значения свойств `useSystemTrustStore` и `revocationCheckSettings`)

Свойство `identityStatus` может иметь следующие значения:

- `valid` — Чтобы расцениваться как действительный, подписывающий сертификат должен соответствовать следующим условиям:
  - Подписывающий сертификат не должен изменяться.
  - Подписывающий сертификат не должен быть сертификатом с истекшим сроком действия или отозванным сертификатом, за исключением случаев, когда в подписи указана допустимая метка времени. Если в подписи указана метка времени, сертификат будет расцениваться как действительный с учетом срока действия на момент подписания документа. (Сертификат, используемый службой создания меток времени для подписания меток времени должен быть связан с доверенным корневым сертификатом на компьютере пользователя.)
  - Подписывающий сертификат доверен. Сертификат считается доверенным, если он находится в хранилище доверенных сертификатов или связан с другим сертификатом в хранилище доверенных сертификатов и для его свойства `useSystemTrustStore` установлено значение true. Можно также определить доверен ли сертификат, используя метод `addCertificate()` объекта XMLSignatureValidator.
  - Этот сертификат, действительно, является подписывающим сертификатом.
- `invalid` — Срок действия сертификата истек или он отозван; не предоставлено меток времени, удостоверяющих действительность сертификата во время подписания; возможно, сертификат был изменен.
- `unknown` — В случае, если сертификат не является недействительным, но и не является доверенным. Самозаверяющие сертификаты, например, будут всегда получать состояние `unknown` (если они явным образом не отмечены как доверенные)3. Свойство `identityStatus` также получает значение `unknown`, если выполнение метода `verify()` не было завершено без ошибки, а также если достоверность не была проверена из-за недопустимости дайджеста подписи.

### Свойство referencesStatus

**Adobe AIR 1.5 и более поздних версий**

Свойство `referencesStatus` сообщает о криптографической целостности ссылок в элементе SignedData подписи.

- `valid` — Если вычисленный дайджест для каждой ссылки в подписи совпадает с соответствующим дайджестом, записанным в XML-подписи. Состояние `valid` означает, что подписанные данные не изменялись.
- `invalid` — Если любой вычисленный дайджест не совпадает с соответствующим дайджестом в подписи.
- `unknown` — Если дайджест ссылки не был проверен. Ссылки не проверяются, если общий дайджест подписи имеет состояние `invalid` или не действителен подписывающий сертификат. Если в свойстве `identityStatus` устанавливается значение`unknown`, то ссылки проверяются только в том случае, когда для свойства `referencesValidationSetting` задано `validOrUnknown`.

# Об XML-подписях

**Adobe AIR 1.5 и более поздних версий**

XML-подпись является цифровой подписью, создаваемой с использованием XML-синтаксиса. Данные из XML-подписи могут использоваться для проверки того, что подписанная информация не изменялась с момента подписания. Кроме того, при выпуске подписывающего сертификата доверенным сертификационным органом достоверность подписчика может быть проверена с помощью инфраструктуры открытого ключа.

XML-подпись может применяться к любому типу цифровых данных (в двоичном или XML-формате). XML-подписи обычно используются в следующих целях:

- для проверки, не изменялись ли внешние или загруженные ресурсы;
- для проверки сообщений, полученных из внешних источников;
- для проверки лицензий на приложение или связанных с подпиской прав.

## Поддерживаемый синтаксис XML-подписи

**Adobe AIR 1.5 и более поздних версий**

Adobe AIR поддерживает следующие элементы из рекомендаций W3C для синтаксиса и порядка обработки XML-подписей:

- Все базовые элементы синтаксиса подписей (раздел 4 в документе рекомендаций W3C), за исключением элемента KeyInfo, поддерживаемого не полностью
- Элемент KeyInfo должен содержать только элемент X509Data
- Элемент X509Data должен содержать только элемент X509Certificate
- Метод создания дайджеста SHA256
- Алгоритм подписания RSA-SHA1 (PKCS1)
- Преобразование и метод канонизации «Канонический XML без комментариев»

- Преобразование вложенной подписи
- метки времени

В следующем документе проиллюстрирован пример стандартной XML-подписи (большая часть криптографических данных была удалена для упрощения этого примера).

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <Reference URI="URI_to_signed_data">
            <Transforms>
                <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/></Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <DigestValue>uoo...vY=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue>Ked...w==</SignatureValue>
    <KeyInfo>
        <X509Data>
            <X509Certificate>i7d...w==</X509Certificate>
        </X509Data>
    </KeyInfo>
</Signature>
```

Ключевыми для подписи являются следующие элементы:

- SignedInfo — Содержит ссылки на подписанные данные и расчетные значения дайджестов на момент подписания. Подписанные данные сами могут быть включены в тот же документ, что и XML-подпись, а также могут находиться вне его.
- SignatureValue — Содержит дайджест элемента SignedInfo, зашифрованный закрытым ключом подписывающей стороны.
- KeyInfo — Содержит подписывающий сертификат, а также любые дополнительные сертификаты, необходимые для установления цепи доверия. Обратите внимание, что, хотя элемент KeyInfo технически не обязателен, AIR не сможет проверить достоверность подписи при его отсутствии.

Существует три основных типа XML-подписей:

- Вложенные — подпись вставляется в XML-данные, которые она подписывает.
- Обертывающие — подписанные XML-данные содержатся в элементе-объекте внутри элемента подписи.
- Отдельные — подписанные данные являются внешними по отношению к XML-подписи. Подписанные данные могут содержаться во внешнем файле. В качестве альтернативы, они также могут находиться в том же XML-документе, что и подпись, но не являются дочерними или родительскими элементами относительно элемента подписи.

XML-подписи используют идентификаторы URI для ссылки на подписанные данные. Подписывающее и проверяющее подпись приложения должны использовать одинаковые принципы разрешения этих идентификаторов URI. При использовании класса XMLSignatureValidator необходимо реализовать интерфейс IURIDereferencer. Этот интерфейс отвечает за разрешение идентификатора URI и возвращение подписанных данных в виде объекта ByteArray. Для возвращаемого объекта ByteArray вычисляется дайджест с помощью того же алгоритма. которым создавался дайджест в подписи.

## Сертификаты и доверенное отношение

**Adobe AIR 1.5 и более поздних версий**

Сертификат состоит из открытого ключа, идентифицирующего информацию, а также, возможно, одного или нескольких сертификатов, принадлежащих выпускающему сертификаты органу.

Существует два способа установления доверенного отношения для сертификата. Можно установить доверенное отношение, получив копию сертификата непосредственно от подписывающей стороны, например, на физическом носителе или по защищенному цифровому каналу связи, например по SSL. Также в определении достоверности подписывающего сертификата можно довериться уполномоченному органу сертификации.

Для этого подписывающий сертификат должен быть выпущен органом, доверенным для компьютера, на котором проверяется подпись. Большинство производителей операционных систем помещают корневые сертификаты ряда сертификационных органов в хранилище доверенных сертификатов операционной системы. Пользователи также могут добавлять сертификаты в это хранилище или удалять их оттуда.

Даже если сертификат выпущен доверенным сертификационным органом, необходимо принять решение о доверии к владельцу этого сертификата. В большинстве случаев на практике это решение принимает конечный пользователь. Например, при установке приложения AIR программа установки отображает идентификационную информацию из сертификата издателя и запрашивает пользователя, хочет ли он установить это приложение. В других случаях может потребоваться сравнить открытый ключ или другую сертификационную информацию со списком допустимых ключей. (Этот список должен быть защищен, возможно, своей собственной подписью, а также может храниться в зашифрованном локальном хранилище AIR, так что сам такой список не может быть подделан.)

***Примечание.*** *Хотя можно принять решение о доверии подписывающему сертификату без независимой проверки, например если подпись самозаверяющая, в действительности установление подлинности такой подписи не дает уверенности ни в чем. Без информации о том, кто создал подпись, подтверждение ее подлинности имеет не очень большое значение. Такая подпись может быть правильно выполненной подделкой.*

### Истечение срока действия сертификатов и их отзыв

**Adobe AIR 1.5 и более поздних версий**

Все сертификаты имеют конечный срок действия. Сертификаты также могут быть отозваны выпускающим органом если, например, связанный с сертификатом закрытый ключ был разглашен или утрачен. Если при создании подписи использовался просроченный или отозванный сертификат, то такая подпись будет считаться недействительной, если в нее не включена метка времени. При наличии метки времени класс XMLSignatureValidator будет проверять подпись, выясняя, был ли действителен сертификат на момент подписания.

Метка времени — это подписанное цифровое сообщение службы меток времени, удостоверяющее, что данные были подписаны в определенную дату и определенный момент времени. Метки времени создаются соответствующими полномочными службами и подписываются их собственными сертификатами. Сертификат службы меток времени, включенный в метку времени, должен быть доверенным на данном компьютере, чтобы данная метка времени считалась действительной. Класс XMLSignatureValidator не предоставляет прикладной интерфейс программирования для обозначения другого сертификата, используемого для проверки метки времени.

# Реализация интерфейса IURIDereferencer

**Adobe AIR 1.5 и более поздних версий**

Для проверки XML-подписи необходимо реализовать интерфейс IURIDereferencer. Этот интерфейс отвечает за разрешение идентификаторов URI в элементах Reference документа XML-подписи и возвращает данные, по которым можно рассчитать дайджест. Расчетный дайджест сравнивается с дайджестом в подписи для определения того, не изменялись ли соответствующие данные с момента создания подписи.

***Примечание.*** *HTML-приложения AIR должны импортировать SWF-библиотеку, содержащую реализацию ActionScript, чтобы можно было проверить XML-подписи. Интерфейс IURIDereferencer не может быть реализован в JavaScript.*

Интерфейс IURIDerefencer содержит один единственный метод, `dereference(uri:String)`, который должен быть реализован. Объект XMLSignatureValidator вызывает этот метод для каждой ссылки в подписи. Этот метод должен возвращать данные в виде объекта ByteArray.

В большинстве случаев также потребуется добавить свойства или методы, которые позволяют объекту dereferencer находить упомянутые данные. Например, если подписанные данные находятся в том же документе, что и подпись, можно добавить переменную member, обеспечивающую ссылку на документ XML. Метод `dereference()` затем может использовать эту переменную вместе с идентификатором URI для обнаружения упомянутых данных. Сходным образом, если подписанные данные размещаются в каталоге локальной файловой системы, методу `dereference()` может потребоваться свойство, предоставляющее путь к этому каталогу, для того чтобы найти упомянутые файлы.

Класс XMLSignatureValidator полностью полагается на объект dereferencer при интерпретации строк с идентификаторами URI. Стандартные правила разыменования идентификаторов URI приведены в разделе 4.3.3 рекомендаций W3C для синтаксиса и порядка обработки XML-подписей.

## Разыменование идентификаторов URI во вложенных подписях

**Adobe AIR 1.5 и более поздних версий**

При создании вложенной XML-подписи ее элементы вставляются в подписанные данные. Например, если следующее сообщение подписывается с помощью структуры вложенной подписи.

```
<message>
    <data>...</data>
</message>
```

Получившийся в результате подписанный документ будет выглядеть следующим образом.

```
<message>
    <data>...</data>
     <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
            <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <Reference URI="">
                <Transforms>
                    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <DigestValue>yv6...Z0Y=</DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue>cCY...LQ==</SignatureValue>
        <KeyInfo>
            <X509Data>
                <X509Certificate>MII...4e</X509Certificate>
            </X509Data>
        </KeyInfo>
    </Signature>
</message>
```

Обратите внимание, что подпись содержит один элемент Reference с пустой строкой в качестве его идентификатора URI. Пустая строка в данном контексте указывает на корневой каталог документа.

Также обратите внимание, что алгоритм преобразования указывает на то, что было применено преобразование вложенной подписи. После преобразования вложенной подписи класс XMLSignatureValidator автоматически удаляет ее из документа перед расчетом дайджеста. Это означает, что объект dereferencer не должен удалять элемент Signature при возвращении данных.

В следующем примере проиллюстрирована работа объекта dereferencer для вложенных подписей:

```
package
{
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class EnvelopedDereferencer
        extends EventDispatcher implements IURIDereferencer
    {
        private var signedMessage:XML;

        public function EnvelopedDereferencer( signedMessage:XML )
        {
            this.signedMessage = signedMessage;
        }

        public function dereference( uri:String ):IDataInput
        {
            try
            {
```

```
                if( uri.length != 0 )
                {
                    throw( new Error("Unsupported signature type.") );
                }
                var data:ByteArray = new ByteArray();
                data.writeUTFBytes( signedMessage.toXMLString() );
                data.position = 0;
            }
            catch (e:Error)
                {
                var error:ErrorEvent =
                    new ErrorEvent("Ref error " + uri + " ", false, false, e.message);
                this.dispatchEvent(error);
                data = null;
                throw new Error("Reference not resolvable: " + uri + ", " + e.message);
            }
            finally
            {
                return data;
            }
        }
    }
}
```

Класс dereferencer использует функцию конструктора с параметром `signedMessage` для создания документа со вложенной подписью, доступного для метода `dereference()`. Поскольку ссылка во вложенной подписи всегда указывает на корневой каталог подписанных данных, метод `dereferencer()` записывает документ в Byte Array и возвращает его.

## Разыменовывание идентификаторов URI в обертывающих и отдельных подписях

**Adobe AIR 1.5 и более поздних версий**

Если подписанные данные расположены в том же документе, что и сама подпись, то идентификаторы URI в ссылках обычно используют синтаксис XPath или XPointer для адресации подписанных элементов. В рекомендациях W3C для синтаксиса и порядка обработки XML-подписей даются только общие рекомендации синтаксиса, поэтому на практике необходимо реализовывать обработку тех подписей, которые предположительно будут встречаться (и добавлять достаточные механизмы проверки ошибок для обработки неподдерживаемого синтаксиса).

Подпись в приложении AIR является примером обертывающей подписи. Файлы приложения перечисляются в элементе Manifest. Элемент Manifest адресуется атрибутом Reference URI, используя строку "#PackageContents", которая ссылается на идентификатор элемента Manifest:

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#" Id="PackageSignature">
    <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
        <SignatureMethod Algorithm="http://www.w3.org/TR/xmldsig-core#rsa-sha1"/>
        <Reference URI="#PackageContents">
            <Transforms>
                <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <DigestValue>ZMGqQdaRKQc1HirIRsDpeBDlaElS+pPotdziIAyAYDk=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue Id="PackageSignatureValue">cQK...7Zg==</SignatureValue>
    <KeyInfo>
        <X509Data>
            <X509Certificate>MII...T4e</X509Certificate>
        </X509Data>
    </KeyInfo>
    <Object>
    <Manifest Id="PackageContents">
        <Reference URI="mimetype">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>0/oCb84THKMagtI0Dy0KogEu92TegdesqRr/clXct1c=</DigestValue>
        </Reference>
        <Reference URI="META-INF/AIR/application.xml">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>P9MqtqSdqcqnFgeoHCJysLQu4PmbUW2JdAnc1WLq8h4=</DigestValue>
        </Reference>
        <Reference URI="XMLSignatureValidation.swf">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>OliRHRAgc9qt3Dk0m0Bi53Ur5ur3fAweIFwju74rFgE=</DigestValue>
        </Reference>
    </Manifest>
</Object>
</Signature>
```

Объект dereferencer для проверки этой подписи должен брать строку URI, содержащую `"#PackageContents"` из элемента Reference, а возвращать элемент Manifest в объекте ByteArray. Символ # указывает на значение атрибута идентификатора элемента.

В следующем примере реализуется объект dereferencer для проверки подписей приложения AIR. Эта реализация достаточно проста, т.к. основана на известной структуре подписи в AIR. Многоцелевой объект dereferencer может быть значительно более сложным.

```
package
{
    import flash.events.ErrorEvent;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class AIRSignatureDereferencer implements IURIDereferencer {
        private const XML_SIG_NS:Namespace =
            new Namespace( "http://www.w3.org/2000/09/xmldsig#" );
        private var airSignature:XML;

        public function AIRSignatureDereferencer( airSignature:XML ) {
            this.airSignature = airSignature;
        }

        public function dereference( uri:String ):IDataInput {
            var data:ByteArray = null;
            try
            {
                if( uri != "#PackageContents" )
                {
                    throw( new Error("Unsupported signature type.") );
                }
                var manifest:XMLList =
                    airSignature.XML_SIG_NS::Object.XML_SIG_NS::Manifest;
                data = new ByteArray();
                data.writeUTFBytes( manifest.toXMLString());
                data.position = 0;
            }
            catch (e:Error)
            {
                data = null;
                throw new Error("Reference not resolvable: " + uri + ", " + e.message);
            }
            finally
            {
                return data;
            }

        }
    }
}
```

Когда проверяется такой тип подписей, фактически проверяются только данные в элементе Manifest. Фактические же файлы в пакете не проверяются совсем. Чтобы проверить файлы пакета на предмет фальсификации, необходимо считать эти файлы, рассчитать дайджест SHA256 и сравнить результат с дайджестом, записанным в манифесте. Класс XMLSignatureValidator не выполняет автоматической проверки таких вторичных ссылок.

***Примечание.*** *Этот пример представлен только, чтобы проиллюстрировать процесс проверки подписи. Нет никакого смысла в том, чтобы приложение AIR проверяло свою собственную подпись. Ведь если приложение уже подверглось вмешательству, фальсификатор запросто мог удалить такую проверку.*

## Вычисления значений дайджестов для внешних ресурсов

**Adobe AIR 1.5 и более поздних версий**

Adobe AIR не содержит встроенных функций для расчета дайджестов SHA256, но в пакет Flex SDK включен служебный класс SHA256. Этот пакет также содержит служебный класс с кодировщиком Base64, который помогает при сравнении расчетного дайджеста с дайджестом, хранимым в подписи.

В следующем примере функция считывает и проверяет файлы, указанные в манифесте пакета AIR:

```
import mx.utils.Base64Encoder;
import mx.utils.SHA256;

private function verifyManifest( sigFile:File, manifest:XML ):Boolean
{
    var result:Boolean = true;
    var message:String = '';
    var nameSpace:Namespace = manifest.namespace();

    if( manifest.nameSpace::Reference.length() <= 0 )
    {
        result = false;
        message = "Nothing to validate.";
    }
    for each (var reference:XML in manifest.nameSpace::Reference)
    {
        var file:File = sigFile.parent.parent.resolvePath( reference.@URI );
        var stream:FileStream = new FileStream();
        stream.open(file, FileMode.READ);
        var fileData:ByteArray = new ByteArray();
        stream.readBytes( fileData, 0, stream.bytesAvailable );

        var digestHex:String = SHA256.computeDigest( fileData );
        //Convert hexidecimal string to byte array
        var digest:ByteArray = new ByteArray();
        for( var c:int = 0; c < digestHex.length; c += 2 ){
            var byteChar:String = digestHex.charAt(c) + digestHex.charAt(c+1);
            digest.writeByte( parseInt( byteChar, 16 ));
        }
        digest.position = 0;
```

```
        var base64Encoder:Base64Encoder = new Base64Encoder();
        base64Encoder.insertNewLines = false;
        base64Encoder.encodeBytes( digest, 0, digest.bytesAvailable );
        var digestBase64:String = base64Encoder.toString();
        if( digestBase64 == reference.nameSpace::DigestValue )
        {
            result = result && true;
            message += "   " + reference.@URI + " verified.\n";
        }
        else
        {
            result = false;
            message += " ---- " + reference.@URI + " has been modified!\n";
        }
        base64Encoder.reset();
    }
    trace( message );
    return result;
}
```

Функция циклически обрабатывает все ссылки в элементе Manifest. Для каждой ссылки рассчитывается дайджест SHA256, который кодируется в формат base64, а затем сравнивается с дайджестом в манифесте. Идентификаторы URI в пакете AIR указывают пути относительно каталога приложения. Эти пути разрешаются с учетом местоположения файла подписи, который всегда находится в подкаталоге META-INF в каталоге приложения. Обратите внимание, что класс Flex SHA256 возвращает дайджест в виде строки шестнадцатеричных символов. Это строка должна преобразовываться в массив ByteArray, содержащий байты, передающие эту шестнадцатеричную строку.

Чтобы использовать классы mx.utils.SHA256 и Base64Encoder в Flash CS4, можно либо найти и скопировать эти классы в каталог разработки приложения, либо скомпилировать библиотеку SWF, содержащую классы с помощью Flex SDK.

## Разыменование идентификаторов URI в отдельных подписях, ссылающихся на внешние данные

**Adobe AIR 1.5 и более поздних версий**

Если идентификатор URI ссылается на внешний источник, необходимо выполнить доступ к данным и загрузить их в объект ByteArray. Если идентификатор URI содержит абсолютный адрес URL, то все сводится к простому считыванию файла или запросу URL. Если, как это, вероятно, бывает в большинстве случаев, идентификатор URI содержит относительный путь, то реализация интерфейса IURIDereferencer должна включать способ разрешения путей для подписанных файлов.

В следующем примере используется объект File, инициализируемый при создании экземпляра класса dereferencer в качестве основы для разрешения подписанных файлов.

```
package
{
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.filesystem.File;
    import flash.filesystem.FileMode;
    import flash.filesystem.FileStream;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;
    public class RelativeFileDereferencer
        extends EventDispatcher implements IURIDereferencer
    {
        private var base:File;

        public function RelativeFileDereferencer( base:File )
        {
            this.base = base;
        }

        public function dereference( uri:String ):IDataInput
        {
            var data:ByteArray = null;
            try{
                var referent:File = this.base.resolvePath( uri );
                var refStream:FileStream = new FileStream();
                data = new ByteArray();
                refStream.open( referent, FileMode.READ );

                refStream.readBytes( data, 0, data.bytesAvailable );

            } catch ( e:Error ) {
                data = null;
                throw new Error("Reference not resolvable: " + referent.nativePath + ", " +
e.message );
            } finally {
                return data;
            }
        }
    }
}
```

Функция `dereference()` просто находит файл, адресуемый идентификатором URI ссылки, загружает содержимое этого файла в байтовый массив и возвращает объект ByteArray.

***Примечание.*** *Прежде чем выполнять проверку удаленных внешних ссылок, оцените уязвимость своего приложения для «звонков домой» или подобного типа атак со стороны злоумышленно созданного документа подписи.*

# Глава 49. Среда клиентской системы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Здесь обсуждается взаимодействие с системой пользователя. Здесь объясняется, как определить поддерживаемые функции и создать многоязычные приложения с помощью установленного пользователем редактора методов ввода (IME), если он имеется. Также в главе приводятся примеры типичного использования для доменов приложений.

### Дополнительные разделы справки

flash.system.System

flash.system.Capabilities

## Основные сведения о среде клиентской системы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для создания приложений с дополнительными возможностями, может потребоваться информация об операционной системе пользователей и доступ к ее функциям. Пакет flash.system содержит набор классов, позволяющих обращаться к функциональным возможностям системы, таким как:

- определение, в каком домене приложения и безопасности выполняется код;
- определение возможностей пользовательского экземпляра среды выполнения Flash (такой как Flash® Player или Adobe® AIR™), например размера экрана (разрешение), и наличия определенных функций, например звука mp3;
- создание многоязычных сайтов с использованием редактора методов вода (IME);
- взаимодействие с контейнером среды выполнения Flash (который может быть HTML-страницей или контейнером приложения).
- сохранение информации в буфере обмена пользователя.

Пакет flash.system также включает классы IMEConversionMode и SecurityPanel. Эти классы содержат статические константы, используемые с классами IME и Security соответственно.

#### Важные понятия и термины

В следующем справочном списке содержатся важные термины.

**Операционная система** Основная программа, выполняемая компьютером, в рамках которой работают все остальные приложения Microsoft Windows, Mac OS X или Linux®.

**Буфер обмена** Системный контейнер для хранения скопированных или вырезанных элементов (включая текст), из которого они вставляются в приложения.

**Домен приложения** Механизм для отделения классов, используемых в разных SWF-файлах. Он предотвращает перезапись классов в том случае, если SWF-файлы содержат разные классы с одинаковыми названиями.

**Редактор метода ввода (IME)** Программа (или инструмент операционной системы), используемая для ввода сложных символов или знаков с помощью стандартной клавиатуры.

**Клиентская система** В программировании клиентом называется часть приложения (или все приложение), которое работает на отдельном компьютере и используется одним человеком. Клиентская система — это базовая операционная система компьютера пользователя.

# Использование класса System

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс System содержит методы и свойства, позволяющие взаимодействовать с операционной системой пользователя и получать сведения о текущем использовании памяти среды выполнения. Кроме того, методы и свойства класса System позволяют прослушивать события `imeComposition`, давать команду среде выполнения на загрузку внешних текстовых файлов с использованием текущей кодовой страницы пользователя или кодировки Юникод и настройку содержимого буфера обмена пользователя.

## Получение данных о системе пользователя во время выполнения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Проверив свойство `System.totalMemory`, можно определить объем памяти (в байтах), который в данный момент используется средой выполнения Это свойство позволяет отслеживать использование памяти и оптимизировать приложения на основе изменений уровня памяти. Например, если определенный видеоэффект вызывает значительное увеличение использования памяти, возможно, этот эффект требуется изменить или убрать совсем.

Свойство `System.ime` ссылается на текущий установленный редактор метода ввода (IME). Оно позволяет прослушивать события `imeComposition` (`flash.events.IMEEvent.IME_COMPOSITION`) с помощью метода `addEventListener()`.

Третьим свойством класса System является `useCodePage`. Если для свойства `useCodePage` установлено значение `true`, в среде выполнения для загрузки внешних текстовых файлов используется стандартная кодовая страница операционной системы. Если установить для этого свойства значение `false`, в среде выполнения внешний файл интерпретируется в кодировке Юникод.

Если для свойства `System.useCodePage` установлено значение `true`, помните, что стандартная кодовая страница операционной системы должна включать символы, используемые во внешнем текстовом файле, чтобы можно было отобразить данный текст. Например, если загружается внешний текстовый файл, содержащий китайские символы, эти символы нельзя отобразить в системе, где используется кодовая страница Windows для английского языка, поскольку она не содержит символов для китайского языка.

Чтобы обеспечить пользователям на всех платформах возможность просматривать внешние текстовые файлы, используемые в приложениях, необходимо сохранить все внешние файлы в кодировке «Юникод» и оставить для свойства `System.useCodePage` значение `false`. Таким образом, среда выполнения интерпретирует текст в кодировке Юникод.

## Сохранение текста в буфере обмена

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс System включает метод `setClipboard()`, позволяющий среде выполнения Flash заполнять буфер обмена пользователя определенной строкой. В целях безопасности метод `Security.getClipboard()` не существует, так как подобный метод потенциально мог бы позволить вредоносным сайтам получать доступ к данным, в последний раз скопированным пользователем в буфер обмена.

Следующий код демонстрирует, как сообщение об ошибке можно скопировать в буфер обмена пользователя, когда происходит ошибка безопасности. Сообщение об ошибке может пригодиться, если пользователь захочет сообщить о потенциальной проблеме в приложении.

```
private function securityErrorHandler(event:SecurityErrorEvent):void
{
    var errorString:String = "[" + event.type + "] " + event.text;
    trace(errorString);
    System.setClipboard(errorString);
}
```

**Flash Player 10 и AIR 1.0**

Класс Clipboard можно использовать для чтения и записи данных буфера обмена в ответ на событие пользователя. В AIR событие пользователя не является обязательным для того, чтобы код, выполняемый в изолированной программной среде приложения, мог получить доступ к буферу обмена.

---

# Использование класса Capabilities

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс Capabilities позволяет разработчикам определить среду, в которой выполняется приложение. С помощью различных свойства класса Capabilities можно определить разрешение пользовательской системы, наличие поддержки ПО для обеспечения специальных возможностей, а также узнать язык операционной системы и текущую установленную версию среды выполнения Flash.

Проверяя свойства класса Capabilities, можно настроить приложение для оптимальной работы в среде конкретного пользователя. Например, с помощью свойств `Capabilities.screenResolutionX` и `Capabilities.screenResolutionY` можно определить разрешение дисплея в системе пользователя и указать наиболее подходящий формат для видео. А еще можно задать свойство `Capabilities.hasMP3`, чтобы узнать, поддерживает ли система пользователя воспроизведение mp3, прежде чем загружать внешний mp3-файл.

В следующем коде используется регулярное выражение для определения версии среды выполнения Flash в клиентской системе.

```
var versionString:String = Capabilities.version;
var pattern:RegExp = /^(\w*) (\d*),(\d*),(\d*),(\d*)$/;
var result:Object = pattern.exec(versionString);
if (result != null)
{
    trace("input: " + result.input);
    trace("platform: " + result[1]);
    trace("majorVersion: " + result[2]);
    trace("minorVersion: " + result[3]);
    trace("buildNumber: " + result[4]);
    trace("internalBuildNumber: " + result[5]);
}
else
{
    trace("Unable to match RegExp.");
}
```

Если требуется отправить данные о возможностях клиентской системы сценарию сервера, чтобы сохранить их в базе данных, можно использовать следующий код ActionScript:

```
var url:String = "log_visitor.cfm";
var request:URLRequest = new URLRequest(url);
request.method = URLRequestMethod.POST;
request.data = new URLVariables(Capabilities.serverString);
var loader:URLLoader = new URLLoader(request);
```

# Пример возможностей: определение возможностей системы

**Flash Player 9 или более поздних версий**

Пример CapabilitiesExplorer демонстрирует, как с помощью класса flash.system.Capabilities можно определить, какие функции поддерживает пользовательская версия среды выполнения Flash. В этом примере используются следующие приемы:

- определение возможностей пользовательской версии среды выполнения Flash с помощью класса Capabilities;
- использование класса ExternalInterface для определения настроек, поддерживаемых обозревателем пользователя;

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения CapabilitiesExplorer находятся в папке Samples/CapabilitiesExplorer. Приложение состоит из следующих файлов:

| File | Описание |
|---|---|
| CapabilitiesExplorer.fla<br>или<br>CapabilitiesExplorer.mxml | Основной файл приложения Flash (FLA) или Flex (MXML). |
| com/example/programmingas3/capabilities/CapabilitiesGrabber.as | Класс, обеспечивающий основные функции приложения, включая добавление возможностей системы в массив, сортировку элементов и использование класса ExternalInterface для получения данных о возможностях обозревателя. |
| capabilities.html | HTML-контейнер, содержащий необходимый код JavaScript для взаимодействия с внешним API-интерфейсом. |

## Обзор файла CapabilitiesExplorer

**Flash Player 9 или более поздних версий**

Файл CapabilitiesExplorer.mxml отвечает за настройку пользовательского интерфейса для приложения CapabilitiesExplorer. Возможности пользовательской версии среды выполнения Flash будут отображаться в экземпляре компонента DataGrid в рабочей области. Также будут отображаться их возможности взаимодействия с обозревателем, если приложение запущено из HTML-контейнера и если доступен внешний API-интерфейс.

После отправки события `creationComplete` основного файла приложения вызывается метод `initApp()`. Метод `initApp()` вызывает метод `getCapabilities()` класса com.example.programmingas3.capabilities.CapabilitiesGrabber. Ниже приводится код для метода `initApp()`.

```
private function initApp():void
{
    var dp:Array = CapabilitiesGrabber.getCapabilities();
    capabilitiesGrid.dataProvider = dp;
}
```

Метод `CapabilitiesGrabber.getCapabilities()` возвращает упорядоченный массив возможностей среды выполнения Flash и браузера, который затем назначается в качестве свойства `dataProvider` для экземпляра компонента DataGrid с именем `capabilitiesGrid` в рабочей области.

## Обзор класса CapabilitiesGrabber

**Flash Player 9 или более поздних версий**

Статический метод `getCapabilities()` класса CapabilitiesGrabber добавляет каждое свойство класса flash.system.Capabilities в массив (`capDP`). Затем он вызывает статический метод `getBrowserObjects()` класса CapabilitiesGrabber. Метод `getBrowserObjects()` использует внешний API-интерфейс для получения объекта navigator обозревателя, который содержит данные о его возможностях. Метод `getCapabilities()` реализуется следующим образом:

```
public static function getCapabilities():Array
{
    var capDP:Array = new Array();
    capDP.push({name:"Capabilities.avHardwareDisable", value:Capabilities.avHardwareDisable});
    capDP.push({name:"Capabilities.hasAccessibility", value:Capabilities.hasAccessibility});
    capDP.push({name:"Capabilities.hasAudio", value:Capabilities.hasAudio});
    ...
    capDP.push({name:"Capabilities.version", value:Capabilities.version});
    var navArr:Array = CapabilitiesGrabber.getBrowserObjects();
    if (navArr.length > 0)
    {
        capDP = capDP.concat(navArr);
    }
    capDP.sortOn("name", Array.CASEINSENSITIVE);
    return capDP;
}
```

Метод `getBrowserObjects()` возвращает массив каждого свойства в объекте navigator обозревателя. Если этот массив содержит один или несколько элементов, то массив возможностей обозревателя (`navArr`) добавляется к массиву возможностей Flash Player (`capDP`), и полученный массив сортируется в алфавитном порядке. В завершение, упорядоченный массив передается основному файлу приложения, который затем заполняет таблицу данных. Ниже приводится код для метода `getBrowserObjects()`.

```
private static function getBrowserObjects():Array
{
    var itemArr:Array = new Array();
    var itemVars:URLVariables;
    if (ExternalInterface.available)
    {
        try
        {
            var tempStr:String = ExternalInterface.call("JS_getBrowserObjects");
            itemVars = new URLVariables(tempStr);
            for (var i:String in itemVars)
            {
                itemArr.push({name:i, value:itemVars[i]});
            }
        }
        catch (error:SecurityError)
        {
            // ignore
        }
    }
    return itemArr;
}
```

Если в текущей пользовательской среде доступен внешний API-интерфейс, среда выполнения Flash вызывает метод `JS_getBrowserObjects()` кода JavaScript, который циклически обрабатывает объект navigator браузера и возвращает строку значений в кодировке URL коду ActionScript. Затем эта строка преобразуется в объект URLVariables (`itemVars`) и добавляется в массив `itemArr`, который передается вызывающему сценарию.

## Взаимодействие с JavaScript

**Flash Player 9 или более поздних версий**

На последнем этапе разработки приложения CapabilitiesExplorer создается сценарий JavaScript, необходимый для циклической обработки всех элементов в объекте navigator обозревателя и добавления пары «имя-значение» во временный массив. Ниже приводится код для метода JavaScript `JS_getBrowserObjects()` в файле container.html.

```
<script language="JavaScript">
    function JS_getBrowserObjects()
    {
        // Create an array to hold each of the browser's items.
        var tempArr = new Array();

        // Loop over each item in the browser's navigator object.
        for (var name in navigator)
        {
            var value = navigator[name];

            // If the current value is a string or Boolean object, add it to the
            // array, otherwise ignore the item.
            switch (typeof(value))
            {
                case "string":
                case "boolean":

                    // Create a temporary string which will be added to the array.
                    // Make sure that we URL-encode the values using JavaScript's
                    // escape() function.
                    var tempStr = "navigator." + name + "=" + escape(value);
                    // Push the URL-encoded name/value pair onto the array.
                    tempArr.push(tempStr);
                    break;
            }
        }
        // Loop over each item in the browser's screen object.
        for (var name in screen)
        {
            var value = screen[name];

            // If the current value is a number, add it to the array, otherwise
            // ignore the item.
            switch (typeof(value))
            {
                case "number":
                    var tempStr = "screen." + name + "=" + escape(value);
                    tempArr.push(tempStr);
                    break;
            }
        }
        // Return the array as a URL-encoded string of name-value pairs.
        return tempArr.join("&");
    }
</script>
```

Сначала код создает временный массив для хранения всех пар «имя-значение» в объекте navigator. Затем объект navigator проходит цикл `for..in`, в котором тип данных текущего значения оценивается для отсеивания нежелательных результатов. В этом приложении нас интересуют только строковые или логические значения, остальные типы данных (такие как функции или массивы) игнорируются. Все строковые или логические значения в объекте navigator добавляются в массив `tempArr`. Затем объект screen обозревателя проходит цикл `for..in`, и все числовые значения добавляются в массив `tempArr`. В завершение, временный массив преобразуется в строку с помощью метода `Array.join()`. В этом массиве в качестве разделителя используется амперсанд (&), что облегчает для кода ActionScript анализ данных с помощью класса URLVariables.

# Глава 50. Запуск и завершение приложения AIR

**Adobe AIR 1.0 и более поздних версий**

В настоящем разделе обсуждаются способы запуска установленного приложения Adobe® AIR®, а также параметры закрытия, и, наконец, соображения относительно закрытия работающего приложения.

***Примечание.*** *Для запуска содержимого SWF, работающего в изолированной программной среде приложения AIR, доступны только объекты NativeApplication, InvokeEvent и BrowserInvokeEvent. Содержимое SWF, воспроизводимое приложением Flash Player, браузером, отдельным проигрывателем (проектором) или приложением AIR вне изолированной программной среды, не имеет доступа к эти классам.*

Краткое разъяснение и примеры кодов с запуском и завершением работы приложений AIR см. в следующих статьях по быстрому началу работы в центре Adobe Developer Connection:

- Параметры запуска

### Дополнительные разделы справки

flash.desktop.NativeApplication

flash.events.InvokeEvent

flash.events.BrowserInvokeEvent

## Вызов приложения

**Adobe AIR 1.0 и более поздних версий**

Приложение AIR вызывается, когда пользователь (или операционная система):

- запускает приложение с рабочего стола;
- Указывает приложение в качестве команды в командной строке.
- открывает тип файла, для которого данное приложение является приложением по умолчанию;
- щелкает по значку приложения в панели задач Dock в Mac OS (неважно, запущено ли уже приложение);
- запускает приложение из программы установки (в конце процесса установки либо после двойного щелчка по файлу AIR уже установленного приложения);
- начинает обновление приложения AIR, когда установленная версия сообщает, что уже обрабатывает обновления (с помощью объявления `<customUpdateUI>true</customUpdateUI>` в файле дескриптора);
- в ОС iOS: получает уведомление через службу push-уведомлений Apple (Apple Push Notification service, APNs);
- вызывает приложение посредством URL-ссылки;

- посещает веб-страницу с элементом Flash badge или приложением, вызывающим метод `com.adobe.air.AIR launchApplication()`, задающий информацию для приложения AIR. (Дескриптор приложения должен содержать объявление `<allowBrowserInvocation>true</allowBrowserInvocation>` для вызова обозревателя.)

При вызове приложения AIR среда AIR отправляет объект InvokeEvent типа `invoke` через единственный объект NativeApplication. Чтобы приложение успело инициализироваться и зарегистрировать прослушиватель событий, события `invoke` ставятся в очередь, а не сбрасываются. Как только прослушиватель регистрируется, все события в очереди доставляются.

***Примечание.*** *При вызове приложения с помощью функции запуска обозревателя объект NativeApplication отправляет событие* `invoke` *только в случае, если приложение еще не запущено.*

Для получения событий `invoke` вызовите метод `addEventListener()` объекта NativeApplication (`NativeApplication.nativeApplication)`. Когда прослушиватель события регистрируется на событие `invoke`, он также получает все события `invoke`, которые произошли до регистрации. События `invoke`, поставленные в очередь, отправляются по одному с небольшим интервалом после возвращения результатов вызова метода `addEventListener()`. Если во время этого процесса происходит новое событие `invoke`, оно может быть отправлено прежде, чем события из очереди. Постановка событий в очередь позволяет обрабатывать события `invoke`, которые произошли до выполнения кода инициализации. Помните, что при добавлении прослушивателя далее в процессе выполнения (после инициализации приложения), он все равно будет получать все события `invoke`, случившиеся с момента запуска приложения.

Запускается только один экземпляр приложения AIR. При повторном вызове уже запущенного приложения среда AIR отправляет новое событие `invoke` работающему экземпляру. Приложение AIR должно ответить на событие `invoke` соответствующим действием (например, открыть документ в новом окне).

Объект `InvokeEvent` может содержать любые аргументы, передаваемые приложению, а также каталог, из которого приложение запускается. Если приложение запущено в результате открытия ассоциированного с ним типа файла, то полный путь к файлу включается в аргументы командной строки. Точно так же, если приложение запущено в результате обновления, предоставляется полный путь к файлу обновления AIR.

Если в одной операции открывается сразу несколько файлов, в Mac OS X создается один объект InvokeEvent. Каждый файл включается в массив `arguments`. В операционных системах Windows и Linux отдельный объект InvokeEvent создается для каждого файла.

Приложение может обрабатывать события `invoke` путем регистрации прослушивателя с объектом NativeApplication:

```
NativeApplication.nativeApplication.addEventListener(InvokeEvent.INVOKE, onInvokeEvent);
```

и определения прослушивателя событий:

```
 var arguments:Array;
var currentDir:File;
public function onInvokeEvent(invocation:InvokeEvent):void {
    arguments = invocation.arguments;
    currentDir = invocation.currentDirectory;
}
```

# Захват аргументов командной строки

**Adobe AIR 1.0 и более поздних версий**

Аргументы командной строки, ассоциируемые с вызовом приложения AIR, доставляются в объект InvokeEvent, который отправляется объектом NativeApplication. Свойство InvokeEvent `arguments` содержит массив аргументов, передаваемых операционной системой при вызове приложения AIR. Если аргументы содержат относительные пути к файлам, то их обычно можно обработать с помощью свойства `currentDirectory`.

Аргументы, передаваемые приложению AIR, обрабатываются как строки, разделенные пробелами, если только они не заключены в двойные кавычки:

| Аргументы | Массив |
|---|---|
| tick tock | {tick,tock} |
| tick "tick tock" | {tick,tick tock} |
| "tick" “tock” | {tick,tock} |
| \"tick\" \"tock\" | {"tick","tock"} |

Свойство `currentDirectory` объекта InvokeEvent содержит объект File, представляющий каталог, из которого запускалось приложение.

Когда приложение вызывается в результате открытия типа файла, ассоциированного с приложением, исходный путь к файлу включается в строковом виде в аргументы командной строки. (Приложение отвечает за открытие файла или выполнение любой другой операции над ним). Точно так же, когда механизм обновления содержится в самом приложении (а не в стандартном пользовательском интерфейсе AIR), исходный путь к файлу AIR включается, когда пользователь дважды щелкает по файлу AIR, содержащему приложение с совпадающим ID.

Доступ к файлу можно получить методом `resolve()` объекта File `currentDirectory`:

```
 if((invokeEvent.currentDirectory != null)&&(invokeEvent.arguments.length > 0)){
    dir = invokeEvent.currentDirectory;
    fileToOpen = dir.resolvePath(invokeEvent.arguments[0]);
}
```

Также необходимо проверить, действительно ли аргумент является путем к файлу.

## Пример: вызов журнала событий

**Adobe AIR 1.0 и более поздних версий**

В этом примере показано, как регистрировать прослушиватели и обрабатывать события `invoke`. В примере все полученные события вызова заносятся в журнал, после чего отображается текущий каталог и аргументы командной строки.

### Пример ActionScript

```
package
{
    import flash.display.Sprite;
    import flash.events.InvokeEvent;
    import flash.desktop.NativeApplication;
    import flash.text.TextField;

    public class InvokeEventLogExample extends Sprite
    {
        public var log:TextField;

        public function InvokeEventLogExample()
        {
            log = new TextField();
            log.x = 15;
            log.y = 15;
            log.width = 520;
            log.height = 370;
            log.background = true;

            addChild(log);

            NativeApplication.nativeApplication.addEventListener(InvokeEvent.INVOKE, onInvoke);
        }

        public function onInvoke(invokeEvent:InvokeEvent):void
        {
            var now:String = new Date().toTimeString();
            logEvent("Invoke event received: " + now);

            if (invokeEvent.currentDirectory != null)
            {
                logEvent("Current directory=" + invokeEvent.currentDirectory.nativePath);
            }
            else
            {
```

```
                logEvent("--no directory information available--");
            }

            if (invokeEvent.arguments.length > 0)
            {
                logEvent("Arguments: " + invokeEvent.arguments.toString());
            }
            else
            {
                logEvent("--no arguments--");
            }
        }

        public function logEvent(entry:String):void
        {
            log.appendText(entry + "\n");
            trace(entry);
        }
    }
}
```

### Пример Flex

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
    invoke="onInvoke(event)" title="Invocation Event Log">
    <mx:Script>
    <![CDATA[
    import flash.events.InvokeEvent;
    import flash.desktop.NativeApplication;

    public function onInvoke(invokeEvent:InvokeEvent):void {
        var now:String = new Date().toTimeString();
        logEvent("Invoke event received: " + now);

        if (invokeEvent.currentDirectory != null){
            logEvent("Current directory=" + invokeEvent.currentDirectory.nativePath);
        } else {
            logEvent("--no directory information available--");
        }

        if (invokeEvent.arguments.length > 0){
            logEvent("Arguments: " + invokeEvent.arguments.toString());
        } else {
            logEvent("--no arguments--");
        }
    }

    public function logEvent(entry:String):void {
        log.text += entry + "\n";
        trace(entry);
    }
    ]]>
    </mx:Script>
    <mx:TextArea id="log" width="100%" height="100%" editable="false"
        valueCommit="log.verticalScrollPosition=log.textHeight;"/>
</mx:WindowedApplication>
```

# Запуск приложения AIR по имени входа пользователя

**Adobe AIR 1.0 и более поздних версий**

Приложение AIR может запускаться автоматически, когда текущий пользователь входит в систему. Для этого необходимо задать свойству `startAtLogin` класса NativeApplication значение `true`. После этого приложение будет автоматически запускаться при входе пользователя в систему. Такое положение дел сохранится до тех пор, пока значение не будет изменено на `false`, пока пользователь не изменит этот параметр вручную в операционной системе, или пока приложение не будет удалено. Запуск при входе является параметром времени выполнения. Этот параметр относится только к конкретному пользователю. Чтобы задать для свойства `startAtLogin` значение `true`, приложение должно быть установлено. Если свойство задается, пока приложение не установлено (а, например, запущено через ADL), генерируется ошибка.

***Примечание.*** *Приложение открывается не при запуске системы, а при входе в нее пользователя.*

Чтобы определить, запущено ли приложение автоматически или в результате действий пользователя, нужно проверить значение свойства `reason` объекта InvokeEvent. Если значение свойства равно `InvokeEventReason.LOGIN`, приложение запускается автоматически. В случае если приложение вызвано другим путем, свойство `reason` получает следующие значения:

- `InvokeEventReason.NOTIFICATION` (только для iOS) — приложение было вызвано через службу APNs. Дополнительные сведения о службе APNs приведены в разделе Использование push-уведомлений.
- `InvokeEventReason.OPEN_URL` — приложение было вызвано из другого приложения или системы.
- `InvokeEventReason.Standard` — все прочие случаи.

Для доступа к свойству `reason` приложение должно быть предназначено для AIR 1.5.1 или более поздней версии (путем задания правильного значения пространства имен в файле дескриптора приложения).

Приведенное ниже простое приложение использует свойство reason объекта InvokeEvent, чтобы определить дальнейшее поведение при реализации события запуска. Если свойство reason имеет значение «login», приложение остается в фоновом режиме. В противном случае оно делает главное приложение видимым. Приложение, использующее этот путь, как правило, запускается при вводе имени входа, так что оно может работать в фоновом режиме или производить мониторинг событий, и открывает окно в ответ на событие запуска, инициированное пользователем.

```
package {
    import flash.desktop.InvokeEventReason;
    import flash.desktop.NativeApplication;
    import flash.display.Sprite;
    import flash.events.InvokeEvent;

    public class StartAtLogin extends Sprite
    {
        public function StartAtLogin()
        {
            try
            {
                NativeApplication.nativeApplication.startAtLogin = true;
            }
            catch ( e:Error )
            {
                trace( "Cannot set startAtLogin:" + e.message );
            }

            NativeApplication.nativeApplication.addEventListener( InvokeEvent.INVOKE, onInvoke );
        }

        private function onInvoke( event:InvokeEvent ):void
        {
            if( event.reason == InvokeEventReason.LOGIN )
            {
                //do background processing...
                trace( "Running in background..." );
            }
            else
            {
                this.stage.nativeWindow.activate();
            }
        }
    }
}
```

***Примечание.*** *Чтобы увидеть разницу в поведении приложения, упакуйте и установите приложение. Свойство* `startAtLogin` *может быть задано только для установленных приложений.*

# Запуск приложения AIR из браузера

**Adobe AIR 1.0 и более поздних версий**

Веб-сайт может с помощью функции вызова из обозревателя запускать установленное приложение AIR, запуск которого предусмотрен из обозревателя. Вызов из обозревателя разрешается, только если в файле дескриптора приложения `allowBrowserInvocation` равно `true`:

```
<allowBrowserInvocation>true</allowBrowserInvocation>
```

Когда приложение вызывается из обозревателя, его объект NativeApplication отправляет объект BrowserInvokeEvent.

Для получения событий BrowserInvokeEvent вызовите метод `addEventListener()` объекта NativeApplication (`NativeApplication.nativeApplication`) в приложении AIR. Когда прослушиватель события регистрируется на событие BrowserInvokeEvent, он также получает все события BrowserInvokeEvent, которые произошли до регистрации. Эти события отправляются после возврата вызова `addEventListener()`, но необязательно раньше остальных событий BrowserInvokeEvent, которые могут быть получены после регистрации. Это позволяет обрабатывать события BrowserInvokeEvent, которые происходят до выполнения кода инициализации (например, когда приложение изначально запускается из обозревателя). Помните, что при добавлении прослушивателя далее в процессе выполнения (после инициализации приложения), он все равно будет получать все события BrowserInvokeEvent, случившиеся с момента запуска приложения.

Ниже перечислены свойства объекта BrowserInvokeEvent.

| **Свойство** | **Описание** |
|---|---|
| arguments | Массив аргументов (строк), передаваемых приложению. |
| isHTTPS | Указывает, использует ли содержимое в обозревателе схему URL-адресов https (`true`, если да, `false`, если нет). |
| isUserEvent | Указывает, привел ли вызов обозревателя к пользовательскому действию (например, щелчку мыши). В AIR 1.0 оно всегда имеет значение `true`; в AIR пользовательское действие обязательно при вызове обозревателя. |
| sandboxType | Тип изолированной программной среды для содержимого в обозревателе. Действительны такие же значения, как для свойства `Security.sandboxType`, т. е. значения могут быть следующими:<br>• `Security.APPLICATION` — содержимое находится в изолированной программной среде приложения.<br>• `Security.LOCAL_TRUSTED` — содержимое находится в локальной изолированной программной среде файловой системы.<br>• `Security.LOCAL_WITH_FILE` — содержимое находится в локальной изолированной программной среде файловой системы.<br>• `Security.LOCAL_WITH_NETWORK` — содержимое находится в локальной изолированной программной средой с сетевым подключением.<br>• `Security.REMOTE` — содержимое находится на удаленном (сетевом) домене. |
| securityDomain | Безопасный домен для содержимого в обозревателе, например `"www.adobe.com"` или `"www.example.org"`. Это свойство задается только для содержимого в удаленной защищенной изолированной программной среде (с сетевого домена). Для содержимого в локальной изолированной программной среде или изолированной программной среде приложения оно не задается. |

Если используется функция вызова обозревателя, не забывайте об ограничениях безопасности. Когда веб-сайт запускает приложение AIR, данные могут передаваться через свойство `arguments` объекта BrowserInvokeEvent. При выполнении операций с конфиденциальными данными, например в API-интерфейсах файла или загрузки кода, с ними нужно обращаться очень осторожно. Уровень риска зависит от того, как приложение распоряжается данными. Если только определенный сайт может запускать приложение, необходимо проверить свойство `securityDomain` объекта BrowserInvokeEvent. Также можно требовать, чтобы веб-сайт, вызывающий приложение, использовал протокол HTTPs — такая информация содержится в свойстве `isHTTPS` объекта BrowserInvokeEvent.

Приложение должно проверять все передаваемые ему данные. Например, если приложение ожидает передачи URL-адресов определенного домена, необходимо удостовериться, что URL-адреса действительно ведут на этот домен. Таким образом можно оградить приложение от случайной отправки конфиденциальных данных злоумышленнику.

Приложение не должно использовать аргументы BrowserInvokeEvent, которые указывают на локальные ресурсы. Например, приложение не должно создавать объекты File на основе пути, передаваемого из обозревателя. Если от обозревателя ожидается передача удаленных путей, приложение должно проверить, не используется ли в них протокол `file://` вместо протокола удаленного доступа.

# Завершение работы приложения

**Adobe AIR 1.0 и более поздних версий**

Самым быстрым способом завершения приложения является вызов метода exit() класса NativeApplication. Он отлично работает, когда приложение не содержит данных, которые нужно сохранить, или внешних ресурсов, которые нужно очистить. Вызов метода `exit()` закрывает все окна и завершает работу приложения. Однако, чтобы окна и другие компоненты могли вмешиваться в процесс завершения работы, например для сохранения важных данных, отправляйте перед вызовом метода `exit()` соответствующие события уведомлений.

Другой способ закрыть приложение без лишних проблем — предоставить единый путь выполнения; при этом неважно, как начинается процесс закрытия. Пользователь (или операционная система) могут вызывать завершение приложения несколькими способами:

- Закрытие последнего окна приложения, если `NativeApplication.nativeApplication.autoExit` равно `true`.
- Выбор команды выхода из приложения через операционную систему, например, когда пользователь выбирает команду выхода из меню по умолчанию. (Это возможно только в Mac OS. В Windows и Linux нет команды выхода из приложения через системный Chrome.)
- Выключение компьютера.

Когда команда выхода передается через операционную систему одним из указанных маршрутов, NativeApplication отправляет событие `exiting`. Если ни один из прослушивателей не отменяет событие `exiting`, закрываются все открытые окна. Каждое окно отправляет событие `closing`, а затем `close`. Если какие-либо окна отменяют событие `closing`, то процесс завершения работы прерывается.

Если в написанном вами приложении важен порядок закрытия, прослушивайте событие `exiting` объекта NativeApplication и закрывайте окна в нужном порядке. Вам это может потребоваться, например, при работе с окном документа, содержащим панель инструментов. Если система уже закрыла панели, но пользователь решил не выходить из приложения, а сначала сохранить данные, могут возникнуть сложности. В Windows событие `exiting` отправляется только после закрытия последнего окна (когда свойство `autoExit` объекта NativeApplication равно `true`).

Чтобы поведение было одинаковым на всех платформах, следуйте приведенным ниже рекомендациям (неважно, каким образом было закрыто приложение: через системный Chrome, команды меню или логику приложения).

1. Всегда отправляйте событие `exiting` через объект NativeApplication, и только потом вызывайте метод `exit()` в коде приложения, а также проверяйте, чтобы другой компонент приложения не отменил это событие.

```
public function applicationExit():void {
    var exitingEvent:Event = new Event(Event.EXITING, false, true);
    NativeApplication.nativeApplication.dispatchEvent(exitingEvent);
    if (!exitingEvent.isDefaultPrevented()) {
        NativeApplication.nativeApplication.exit();
    }
}
```

2 Прослушивайте событие `exiting` в приложении из объекта `NativeApplication.nativeApplication`, а все окна закрывайте в обработчике (сначала необходимо отправить событие `closing`). После закрытия окон выполняйте все необходимые задачи по очистке, например сохранение данных приложения, удаление временных файлов. Для очистки используйте только синхронные методы, чтобы они наверняка завершились, прежде чем закроется приложение.

   Если порядок закрытия окон неважен, можно пройти массив `NativeApplication.nativeApplication.openedWindows` и закрыть окна по очереди. Если порядок *важен*, включите механизм для закрытия окон в нужной последовательности.

```
private function onExiting(exitingEvent:Event):void {
    var winClosingEvent:Event;
    for each (var win:NativeWindow in NativeApplication.nativeApplication.openedWindows) {
        winClosingEvent = new Event(Event.CLOSING,false,true);
        win.dispatchEvent(winClosingEvent);
        if (!winClosingEvent.isDefaultPrevented()) {
            win.close();
        } else {
            exitingEvent.preventDefault();
        }
    }

    if (!exitingEvent.isDefaultPrevented()) {
        //perform cleanup
    }
}
```

3 Все окна должны обрабатывать собственную очистку путем прослушивания своих событий `closing`.

4 Не используйте в приложении более одного прослушивателя `exiting`, так как ранее вызванные обработчики не могут знать, отменят ли последующие обработчики событие `exiting` (а на порядок выполнения полагаться не стоит).

# Глава 51. Работа с информацией среды выполнения AIR и операционной системы

**Adobe AIR 1.0 и более поздних версий**

В данном разделе описываются способы, которыми приложение AIR может управлять сопоставлением файлов операционной системы, обнаруживать активность пользователя и получать информацию о среде выполнения Adobe® AIR™.

### Дополнительные разделы справки

flash.desktop.NativeApplication

## Управление сопоставлением файлов

**Adobe AIR 1.0 и более поздних версий**

Сопоставление приложения и типа файла должно быть объявлено в дескрипторе приложения. Во время процесса установки программа установки приложения AIR назначает приложение AIR в качестве приложения по умолчанию для открытия каждого из объявленных типов файлов, если уже не назначено другое приложение по умолчанию. Процесс установки приложения AIR не переопределяет существующее сопоставление типов файлов. Чтобы сопоставить другое приложение, необходимо вызвать метод `NativeApplication.setAsDefaultApplication()` во время выполнения.

Рекомендуется убедиться в правильности сопоставления файлов перед запуском приложения. Причина в том, что программа установки приложения AIR не переопределяет существующие сопоставления файлов, к тому же сопоставления файлов в системе пользователя могут измениться в любое время. Если файлам сопоставлено другое приложение, считается вежливым спросить пользователя перед изменением существующего сопоставления.

Следующие методы класса NativeApplication позволяют приложению управлять сопоставлением файлов. В каждом из методов расширение типа файла используется в качестве параметра:

| Метод | Описание |
|---|---|
| isSetAsDefaultApplication() | Возвращает значение true, если приложение AIR в данный момент сопоставлено указанному типу файла. |
| setAsDefaultApplication() | Назначает приложение AIR приложением для открытия заданного типа файла. |
| removeAsDefaultApplication() | Удаляет сопоставление приложения AIR заданному типу файла. |
| getDefaultApplication() | Сообщает путь к приложению, которое в настоящий момент сопоставлено типу файла. |

AIR может управлять сопоставлением только тем типам файлов, которые исходно объявлены в дескрипторе приложения. Информацию о сопоставлении необъявленным типам файлов получить невозможно, даже если пользователь вручную установил такое сопоставление. При вызове одного из методов управления сопоставлением файлов с использованием расширения типа файла, не объявленного в дескрипторе приложения, приложение выдает исключение этапа выполнения.

# Получение версии среды выполнения и уровня исправлений

**Adobe AIR 1.0 и более поздних версий**

Объект NativeApplication имеет свойство `runtimeVersion`, которое представляет версию среды, в которой выполняется приложение (строка, например `"1.0.5"`). Объект NativeApplication также имеет свойство `runtimePatchLevel`, которое представляет уровень исправлений среды выполнения (число, например 2960). В следующем коде используются эти свойства:

```
trace(NativeApplication.nativeApplication.runtimeVersion);
trace(NativeApplication.nativeApplication.runtimePatchLevel);
```

# Обнаружение возможностей AIR

**Adobe AIR 1.0 и более поздних версий**

Для файла, входящего в пакет приложения Adobe AIR, свойство `Security.sandboxType` установлено на значение, определяемое константой `Security.APPLICATION`. Можно загружать содержимое (которое может содержать или не содержать API-интерфейсы AIR) в зависимости от того, находится ли файл в изолированной программной среде безопасности Adobe AIR, как показано в следующем коде:

```
if (Security.sandboxType == Security.APPLICATION)
{
    // Load SWF that contains AIR APIs
}
else
{
    // Load SWF that does not contain AIR APIs
}
```

Все ресурсы, не установленные с приложением AIR, назначаются тем же изолированным программным средам безопасности, которым они были бы назначены проигрывателем Adobe® Flash® Player в веб-обозревателе. Удаленные ресурсы помещаются в изолированные программные среды в соответствии с исходными доменами, а локальные ресурсы помещаются в одну из трех изолированных программных сред: локальную с сетевым подключением, локальную файловой системы или локальную доверенную.

Можно проверить, установлено ли статическое свойство `Capabilities.playerType` на значение `"Desktop"`, чтобы понять, выполняется ли содержимое в среде выполнения (а не в проигрывателе Flash Player, запущенном в обозревателе).

Дополнительные сведения см. в разделе ««Безопасность AIR» на странице 1160».

# Отслеживание присутствия пользователя

**Adobe AIR 1.0 и более поздних версий**

Объект NativeApplication отправляет два события, которые позволяют отслеживать активность пользователя. Если в течение периода времени, заданного свойством `NativeApplication.idleThreshold`, не обнаружена активность мыши или клавиатуры, объект NativeApplication отправляет событие `userIdle`. При следующем вводе с клавиатуры или мыши объект NativeApplication отправляет событие `userPresent`. Интервал `idleThreshold` измеряется в секундах и имеет значение по умолчанию, равное 300 (5 минут). При помощи свойства `NativeApplication.nativeApplication.lastUserInput` можно определить количество секунд, прошедших с момента последнего ввода данных пользователем.

Следующие строки кода устанавливают предел времени ожидания на 2 минуты и прослушивают оба события: `userIdle` и `userPresent`:

```
NativeApplication.nativeApplication.idleThreshold = 120;
NativeApplication.nativeApplication.addEventListener(Event.USER_IDLE, function(event:Event) {
    trace("Idle");
});
NativeApplication.nativeApplication.addEventListener(Event.USER_PRESENT,
function(event:Event) {
    trace("Present");
});
```

***Примечание.*** *Только одно событие `userIdle` отправляется между двумя событиями `userPresent`.*

# Глава 52. Работа с собственными окнами AIR

**Adobe AIR 1.0 и более поздних версий**

При помощи классов, предоставляемых API-интерфейсом собственных окон Adobe® AIR®, можно создавать окна рабочего стола и управлять ими.

## Основные сведения о собственных окнах в AIR

**Adobe AIR 1.0 и более поздних версий**

Краткое описание работы с собственными окнами в AIR и примеры кода можно найти в следующих статьях по быстрому началу работы в центре Adobe Developer Connection:

- Создание приложения с прозрачными окнами (Flex)
- Взаимодействие с окном (Flex)
- Настройка внешнего вида и функций собственного окна (Flex)
- Запуск окон (Flex)
- Создание всплывающих окон в системном лотке уведомлений (Flex)
- Управление порядком отображения окон (Flex)
- Создание непрямоугольных окон с возможностью изменения размера (Flex)
- Взаимодействие с окном (Flash)
- Настройка внешнего вида и функций собственного окна (Flash)
- Создание всплывающих окон в системном лотке уведомлений (Flash)
- Управление порядком отображения окон (Flash)
- Создание непрямоугольных окон с возможностью изменения размера (Flash)

AIR предоставляет удобный API-интерфейс окон для использования на различных платформах для создания собственных окон операционной системы при помощи методов программирования Flash®, Flex™ и HTML.

AIR предоставляет широкие возможности для разработки внешнего вида ваших приложений. Создаваемые окна могут выглядеть как стандартные приложения рабочего стола, отвечающие стилю Apple, реализованному на компьютерах Mac, соответствующие канонам Microsoft при работе в Windows, гармонично сочетающиеся с диспетчером окон Linux, причем все это реализуется без добавления каких-либо дополнительных платформенно-зависимых программных кодов. Вместо этого можно использовать расширяемый Chrome, предоставляемый платформой Flex, с изменяемыми обложками для создания собственного стиля, не зависимого от среды выполнения приложения. Можно даже нарисовать собственный Chrome окна с полной поддержкой прозрачности и полупрозрачности относительно рабочего стола при помощи векторных и растровых объектов. Надоели прямоугольные окна? Нарисуйте круглое.

## Окна в AIR

**Adobe AIR 1.0 и более поздних версий**

Adobe AIR поддерживает три различных прикладных интерфейса программирования для работы с окнами.

- Ориентированный на ActionScript класс NativeWindow предлагает прикладной интерфейс программирования для работы с окнами на самом нижнем уровне. Используйте класс NativeWindow в приложениях, создаваемых в средах ActionScript и Flash Professional. Рассмотрите возможность применения класса NativeWindow для определения окон, используемых в приложении.
- В среде разработки HTML можно использовать класс JavaScript Window так же, как он применяется в веб-приложениях для обозревателя. Вызовы методов JavaScript Window перенаправляются к нижележащему объекту NativeWindow.
- Классы mx:WindowedApplication и mx:Window инфраструктуры Flex обеспечивают «обертку» Flex для класса NativeWindow. Компонент WindowedApplication замещает компонент Application при создании приложения AIR с помощью Flex и всегда должен использоваться в качестве исходного окна в приложении Flex.

### Окна ActionScript

При создании окон при помощи класса NativeWindow используйте рабочую область и список отображения Flash Player напрямую. Чтобы добавить визуальный объект в NativeWindow, добавьте объект в список отображения в рабочей области окна или в другой контейнер экранного объекта в рабочей области.

### Окна HTML

При создании окон HTML для отображения содержимого используется HTML, CSS и JavaScript. Чтобы добавить визуальный объект в окно HTML, это содержимое добавляется в HTML DOM. Окна HTML представляют собой особую категорию NativeWindow. Главный узел AIR задает свойство `nativeWindow` для окон HTML, которое обеспечивает доступ к соответствующему экземпляру NativeWindow. При помощи этого свойства можно получить доступ к свойствам, методам и событиям NativeWindow, описанным здесь.

***Примечание.*** *Объект JavaScript Window также содержит методы для выполнения сценариев окон-контейнеров, например* `moveTo()` *и* `close()`*. Если доступны похожие методы, можно использовать тот, который является более удобным.*

### Окна платформы Flex

При создании окон при помощи Flex для заполнения окна обычно используются компоненты MXML. Чтобы добавить компонент Flex в окно, элемент компонента добавляется в MXML-определение окна. Для динамического добавления содержимого можно также использовать ActionScript. Компоненты mx:WindowedApplication и mx:Window представляют собой контейнеры Flex и могут непосредственно принимать компоненты Flex, тогда как объекты NativeWindow — не могут. При необходимости можно получить доступ к свойствам и методам NativeWindow при помощи объектов WindowedApplication и Window, используя свойство `nativeWindow`.

### Начальное окно приложения

AIR автоматически создает первое окно приложения. AIR задает свойства и содержимое окна при помощи параметров, заданных элементом `initialWindow` файла дескриптора приложения.

Если основным содержимым является SWF-файл, AIR создает экземпляр NativeWindow, загружает SWF-файл и добавляет его в рабочую область окна. Если основным содержимым является файл HTML, AIR создает окно HTML и загружает файл HTML.

## Классы собственных окон

**Adobe AIR 1.0 и более поздних версий**

API-интерфейс собственных окон содержит следующие классы:

| Пакет | Классы |
|---|---|
| flash.display | • NativeWindow<br>• NativeWindowInitOptions<br>• NativeWindowDisplayState<br>• NativeWindowResize<br>• NativeWindowSystemChrome<br>• NativeWindowType |
| flash.events | • NativeWindowBoundsEvent<br>• NativeWindowDisplayStateEvent |

## Схема событий собственных окон

**Adobe AIR 1.0 и более поздних версий**

Собственные окна отправляют события соответствующим компонентам с целью их оповещения о предстоящем важном изменении или о том, что такое изменение уже произошло. Многие события, связанные с окнами, отправляются парами. Первое событие предупреждает о предстоящем изменении. Второе событие сообщает о том, что это изменение уже произошло. Предупреждающее событие можно отменить, а оповещающее событие — нет. Далее приводится последовательность событий, которая реализуется при нажатии пользователем на кнопку разворачивания окна.

1. Объект NativeWindow отправляет событие `displayStateChanging`.
2. Если ни один из зарегистрированных прослушивателей не отменяет это событие, окно разворачивается.
3. Объект NativeWindow отправляет событие `displayStateChange`.

   Помимо этого объект NativeWindow также отправляет события, связанные с соответствующими изменениями размера и положения окна. Окно не отправляет предупреждающие события для таких изменений. Связанными событиями являются следующие:

   a. Событие `move` отправляется при перемещении верхнего левого угла окна в результате операции разворачивания.
   b. Событие `resize` отправляется при изменении размера окна в результате операции разворачивания.

   Объект NativeWindow отправляет схожую последовательность событий при сворачивании, восстановлении, закрытии, перемещении и изменении размера окна.

   Предупреждающие события отправляются, только если изменение было инициировано посредством Chrome окна или другого механизма, управляемого операционной системой. При вызове метода окна для изменения размера окна, его положения или состояния отображения окно отправляет только события, оповещающие о произошедшем изменении. При желании можно отправить предупреждающее событие при помощи метода окна `dispatchEvent()` и перед продолжением процесса изменения проверить, не было ли отменено предупреждающее событие.

Дополнительные сведения о классах, методах, свойствах и событиях API-интерфейса окон см. в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

## Свойства, управляющие стилем и поведением собственных окон

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Следующие свойства управляют основным оформлением и поведением окна:

- `type`
- `systemChrome`
- `transparent`
- `owner`

При создании окна эти свойства задаются для объекта NativeWindowInitOptions, передаваемого конструктору окна. AIR считывает свойства начального окна приложения из дескриптора приложения. (Кроме свойства `type`, которое невозможно задать в дескрипторе приложения и которое всегда имеет значение `normal`.) Эти свойства невозможно изменить после создания окна.

Некоторые настройки этих свойств взаимно несовместимы: `systemChrome` не может иметь значение `standard`, когда свойство `transparent` имеет значение `true` или свойство `type` имеет значение `lightweight`.

### Типы окон

**Adobe AIR 1.0 и более поздних версий**

Типы окон AIR сочетают в себе атрибуты Chrome и видимости собственной операционной системы для создания трех функциональных типов окон. При помощи констант, определенных в классе NativeWindowType, можно создавать ссылки на названия типов в коде. AIR предоставляет следующие типы окон:

| Тип | Описание |
|---|---|
| Normal | Типичное окно. Окна типа normal имеют полноформатный стиль Chrome и отображаются на панели задач Windows и в меню окна в Mac OS X. |
| Utility | Палитра инструментов. Окна типа utility имеют более простую версию системного Chrome и не отображаются на панели задач Windows и в меню окон в Mac OS X. |
| Lightweight | Окна типа lightweight не имеют Chrome и не отображаются на панели задач Windows и в меню окон в Mac OS X. Кроме того, окна типа lightweight не имеют системного меню, вызываемого сочетанием клавиш Alt+Пробел, в Windows. Окна типа lightweight подходят для уведомлений и элементов управления типа «комбинированное окно», которые открывают кратковременную область отображения. Когда используется значение lightweight для свойства `type`, свойство `systemChrome` должно иметь значение `none`. |

### Chrome окна

**Adobe AIR 1.0 и более поздних версий**

Chrome окна состоит из набора элементов управления, при помощи которых пользователи управляют окном в среде рабочего стола. К элементам Chrome относятся: строка заголовка, кнопки строки заголовка, границы и индикатор изменения размера.

### Системный Chrome

Свойству `systemChrome` можно присвоить значение `standard` или `none`. Системный Chrome `standard` обеспечивает окно набором стандартных элементов управления, созданных и стилизованных операционной системой пользователя. Выберите значение `none`, чтобы применить собственный Chrome к окну. При помощи констант, определенных в классе NativeWindowSystemChrome, можно создать ссылку на настройки системного Chrome в коде.

Системный Chrome управляется системой. Приложение не имеет прямого доступа к самим элементам управления, но может реагировать на события, отправляемые при использовании этих элементов управления. При использовании стандартного Chrome окна свойство `transparent` должно иметь значение `false`, а свойство `type` должно иметь значение `normal` или `utility`.

### Chrome платформы Flex

При использовании компонентов Flex WindowedApplication или Window окно может иметь либо системную рамку, либо рамку, предоставляемую платформой Flex. Для использования Chrome платформы Flex задайте для свойства `systemChrome`, используемого при создании окна, значение `none`. При использовании во Flex 4 компонентов spark, а не mx необходимо указать класс обложки, чтобы использовать Chrome из Flex. Можно использовать встроенные обложки, или создать собственные. Следующий пример демонстрирует использование встроенного класса обложки WindowedApplication компонента spark для определения хрома окна:

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx">
<fx:Style>
@namespace "library://ns.adobe.com/flex/spark";
WindowedApplication
{
    skinClass:ClassReference("spark.skins.spark.SparkChromeWindowedApplicationSkin");
}
</fx:Style>
</s:WindowedApplication>
```

Дополнительные сведения см. в документе Использование Flex 4: о контейнерах окон AIR: управление Chrome окна

### Пользовательский Chrome

При создании окна без использования системного Chrome необходимо добавить собственные элементы управления рамки для осуществления взаимодействия пользователя с окном. Можно также создавать прозрачные, не прямоугольные окна.

Для использования пользовательского Chrome с компонентами mx:WindowedApplication или mx:Window необходимо задать для стиля `showFlexChrome` значение `false`. В противном случае среда Flex добавит собственный Chrome к вашим окнам.

## Прозрачность окон

**Adobe AIR 1.0 и более поздних версий**

Для реализации альфа-наложения окна на поверхность рабочего стола или других окон необходимо задать для свойства `transparent` окна значение `true`. Значение свойства `transparent` должно быть задано до того, как окно будет создано, после чего его нельзя будет изменить.

Прозрачное окно не имеет фона по умолчанию. Любая область окна, не содержащая объект, нарисованный приложением, является невидимой. Если экранный объект имеет значение альфа меньше единицы, то сквозь него просвечивают другие экранные объекты в том же окне, другие окна и рабочий стол.

Прозрачные окна полезны при создании приложений с границами, имеющими неправильную форму, исчезающими или невидимыми. Визуализация больших областей альфа-наложения может осуществляться медленно, поэтому такой эффект следует применять умеренно.

***Важная информация.*** *В среде Linux события мыши не передаются через полностью прозрачные пиксели экрана. Поэтому не следует создавать окна с большими, полностью прозрачными областями, поскольку таким образом можно заблокировать доступ к другим окнам или экранным элементам, сделав их невидимыми для пользователя. В Mac OS X и Windows события мыши успешно передаются через прозрачные пиксели.*

Прозрачность нельзя применить к окнам, имеющим системный Chrome. Кроме того, содержимое SWF и PDF в HTML может не отображаться в прозрачных окнах. Дополнительные сведения см. в разделе ««Примечания по загрузке содержимого SWF или PDF на страницу HTML» на странице 1081.

Статическое свойство `NativeWindow.supportsTransparency` сообщает о том, поддерживается ли прозрачность окна. Если прозрачность не поддерживается, приложение размещается на черном фоне. В таких случаях любые прозрачные области приложения отображаются непрозрачным черным. Оптимальным способом во время проверки этого свойства со значением `false` является определение возврата. Например, можно отобразить предупреждающее диалоговое окно или резервный прямоугольный непрозрачный пользовательский интерфейс.

Заметьте, что прозрачность всегда поддерживается операционными системами Mac и Windows. Поддержка операционной системы Linux требует использования диспетчера компоновки окон, но даже если диспетчер компоновки окон активен, не всегда можно реализовать прозрачность из-за пользовательских параметров экрана пользователя или конфигурации аппаратного обеспечения.

## Прозрачность окон в MXML-приложении

**Adobe AIR 1.0 и более поздних версий**

По умолчанию фон MXML-окна непрозрачный, даже если окно создано как *transparent*. (Обратите внимание на эффект прозрачности в углах окна.) Чтобы обеспечить прозрачный фон окна, необходимо задать фоновый цвет и значение альфа в таблице стилей или элементе <mx:Style>, содержащемся в файле MXML приложения. Например, следующее объявление стиля придает фону слегка прозрачный зеленый оттенок:

```
WindowedApplication
{
    background-alpha:".8";
    background-color:"0x448234";
}
```

## Прозрачность окон в HTML-приложении

**Adobe AIR 1.0 и более поздних версий**

По умолчанию фон содержимого HTML, отображаемого в окнах HTML, и объектов HTMLLoader непрозрачный, даже если окно-контейнер прозрачное. Чтобы отключить фон, отображаемый по умолчанию для содержимого HTML, задайте для свойства `paintsDefaultBackground` значение `false`. Следующий пример создает HTMLLoader и отключает фон по умолчанию:

```
var htmlView:HTMLLoader = new HTMLLoader();
htmlView.paintsDefaultBackground = false;
```

В данном примере используется JavaScript для отключения фона по умолчанию окна HTML:

```
window.htmlLoader.paintsDefaultBackground = false;
```

Если для элемента документа HTML задать фоновый цвет, фон этого элемента не будет прозрачным. Установка значения частичной прозрачности (или непрозрачности) не поддерживается. Однако можно использовать прозрачную графику формата PNG в качестве фона страницы или элемента страницы для достижения подобного визуального эффекта.

### Владение окном

Одно окно может быть *владельцем* одного или нескольких других окон. Подчиненные окна всегда отображаются на переднем плане основного окна, сворачиваются и восстанавливаются вместе с основным окном, и закрываются при закрытии основного окна. Окно-владелец не может быть перенесено в другое окно или удалено. Окно может принадлежать только одному основному окну, но может являться владельцем любого количества других окон.

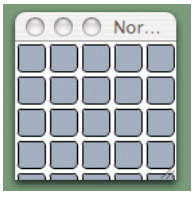
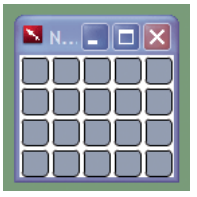
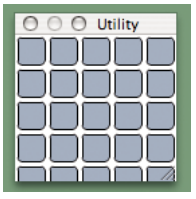
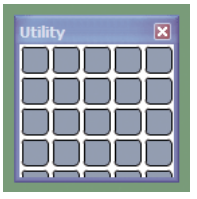
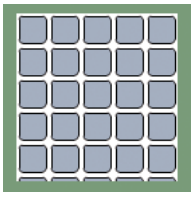
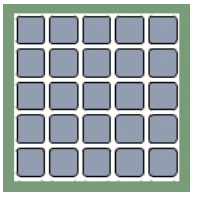
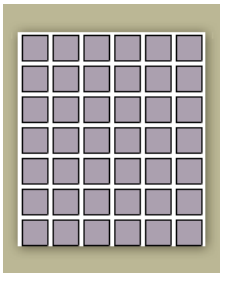
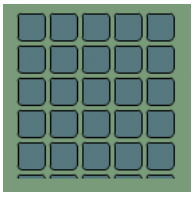
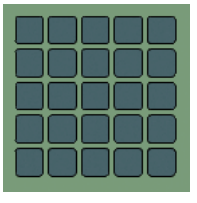
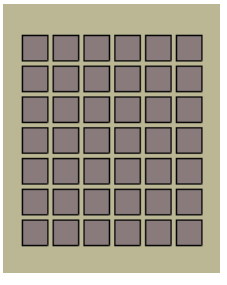
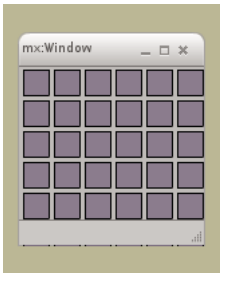
Функция владения окнами позволяет упростить процедуру управления окнами, которые используются для палитр инструментов и диалоговых окон. Например, если диалоговое окно «Сохранить» отображается в окне документа, в результате назначения окна документа владельцем диалогового окна последнее автоматически будет оставаться перед окном документа.

- NativeWindow.owner
- Кристиан Кантрелл (Christian Cantrell). Собственные окна в AIR 2.6

## Таблица визуальных эффектов окна

**Adobe AIR 1.0 и более поздних версий**

Следующая таблица иллюстрирует визуальные эффекты, достигаемые путем различных сочетаний настроек свойств окна в операционных системах Mac OS X, Windows и Linux:

| **Настройки окна** | **Mac OS X** | **Microsoft Windows** | **Linux*** |
|---|---|---|---|
| Type: normal<br>SystemChrome: standard<br>Transparent: false |  |  | |
| Type: utility<br>SystemChrome: standard<br>Transparent: false |  |  | |
| Type: Any<br>SystemChrome: none<br>Transparent: false |  |  |  |
| Type: Any<br>SystemChrome: none<br>Transparent: true |  |  |  |
| mxWindowedApplication или mx:Window<br>Type: Any<br>SystemChrome: none<br>Transparent: true | | |  |

**Ubuntu с диспетчером окон Compiz*

***Примечание.*** *Следующие элементы системного Chrome не поддерживаются AIR: панель инструментов Mac OS X, значки-заменители Mac OS X, значки строки заголовка в Windows и альтернативный системный Chrome.*

# Создание окон

**Adobe AIR 1.0 и более поздних версий**

AIR автоматически создает первое окно приложения, но можно создать дополнительные окна. Для создания собственного окна используется метод конструктора NativeWindow.

Окно HTML создается либо при помощи метода HTMLLoader `createRootWindow()`, либо путем вызова метода JavaScript `window.open()` из документа HTML. Созданное окно является объектом NativeWindow, в списке отображения которого содержится объект HTMLLoader. Объект HTMLLoader интерпретирует и отображает содержимое HTML и JavaScript для окна. Доступ к свойствам базового объекта NativeWindow из JavaScript можно получить с помощью свойства `window.nativeWindow`. (Это свойство доступно только коду, выполняемому в изолированной программной среде AIR.)

При инициализации окна, включая начальное окно приложения, следует учитывать создание окна в невидимом состоянии, загрузку содержимого или выполнение любых графических обновлений и последующее преобразование окна видимое. Такая последовательность позволяет сделать так, чтобы любые резкие визуальные изменения не были заметны пользователю. Создание начального окна приложения в невидимом состоянии можно указать путем задания тега `<visible>false</visible>` в дескрипторе приложения (или путем пропуска тега, поскольку значением по умолчанию является `false`). Новые объекты NativeWindow являются невидимыми по умолчанию. При создании окна HTML с использованием метода `createRootWindow()` объекта HTMLLoader можно задать для аргумента `visible` значение `false`. Вызовите метод `activate()` объекта NativeWindow или задайте для свойства `visible` значение `true`, чтобы сделать окно видимым.

## Задание свойств инициализации окна

**Adobe AIR 1.0 и более поздних версий**

Свойства инициализации собственного окна невозможно изменить после создания окна рабочего стола. Ниже приводятся такие неизменяемые свойства и их значения по умолчанию.

| Свойство | Значение по умолчанию |
|---|---|
| systemChrome | standard |
| type | normal |
| transparent | false |
| owner | null |
| maximizable | true |
| minimizable | true |
| resizable | true |

Свойства начального окна, создаваемого AIR, задаются в файле дескриптора приложения. Главное окно приложения AIR всегда относится к типу *normal*. (В файле дескриптора можно задать дополнительные свойства окон, например `visible`, `width` и `height`, которые можно изменить в любое время.)

Свойства других собственных окон и окон HTML, создаваемых приложением, задаются с использованием класса NativeWindowInitOptions. При создании окна необходимо передать объект NativeWindowInitOptions, задающий свойства окна, либо функции конструктора NativeWindow, либо методу HTMLLoader `createRootWindow()`.

Следующий код создает объект NativeWindowInitOptions для окна типа utility:

```
var options:NativeWindowInitOptions = new NativeWindowInitOptions();
options.systemChrome = NativeWindowSystemChrome.STANDARD;
options.type = NativeWindowType.UTILITY
options.transparent = false;
options.resizable = false;
options.maximizable = false;
```

Свойство *systemChrome* не может иметь значение *standard*, если свойство *transparent* имеет значение `true` или свойство `type` имеет значение `lightweight`.

***Примечание.*** *Свойства инициализации окна, созданного при помощи функции JavaScript `window.open()`, задать невозможно. Однако можно переопределить способ создания этих окон путем реализации собственного класса HTMLHost. Дополнительные сведения см. в разделе ««Обработка вызовов сценарием JavaScript метода window.open()» на странице 1093».*

При создании окна с использованием класса Flex mx:Window свойства инициализации необходимо задать для самого объекта окна, либо в MXML-объявлении окна, либо в коде для создания окна. Базовый объект NativeWindow не создается, пока не вызван метод `open()`. После открытия окна эти свойства инициализации нельзя будет изменить.

## Создание начального окна приложения

**Adobe AIR 1.0 и более поздних версий**

AIR создает начальное окно приложения на основе свойств, заданных в дескрипторе приложения, и загружает файл, на который ссылается элемент содержимого. Элемент содержимого должен представлять собой SWF- или HTML-файл.

Начальное окно может быть главным окном приложения или служить средством запуска других окон. Его вовсе не обязательно делать видимым.

### Создание начального окна при помощи ActionScript

**Adobe AIR 1.0 и более поздних версий**

При создании приложения AIR с использованием Flex SDK и ActionScript основной класс приложения должен расширять класс Sprite (или подкласс класса Sprite). Этот класс служит основной точкой входа для приложения.

При запуске приложения AIR создает окно, создает экземпляр основного класса и добавляет этот экземпляр в рабочую область окна. Для получения доступа к окну можно прослушивать событие `addedToStage`, затем при помощи свойства `nativeWindow` объекта Stage получить ссылку на объект NativeWindow.

Следующий пример иллюстрирует структуру основного класса приложения AIR, построенного при помощи ActionScript:

```
package {
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.events.Event;

    public class MainClass extends Sprite
    {
        private var mainWindow:NativeWindow;
        public function MainClass(){
            this.addEventListener(Event.ADDED_TO_STAGE, initialize);
        }

        private function initialize(event:Event):void{
            mainWindow = this.stage.nativeWindow;
            //perform initialization...
            mainWindow.activate(); //show the window
        }
    }
}
```

***Примечание.*** *Технически МОЖНО получить доступ к свойству* `nativeWindow` *в функции конструктора основного класса. Однако этот особый случай относится только к начальному окну приложения.*

При создании приложения в среде Flash Professional основной класс документа создается автоматически, если не создан собственный класс в отдельном файле ActionScript. Доступ к объекту NativeWindow начального окна можно получить с помощью свойства `nativeWindow` рабочей области. Например, следующий код позволяет активировать главное окно в развернутом состоянии (из временной шкалы):

```
import flash.display.NativeWindow;

var mainWindow:NativeWindow = this.stage.nativeWindow;
mainWindow.maximize();
mainWindow.activate();
```

## Создание начального окна при помощи Flex

**Adobe AIR 1.0 и более поздних версий**

При создании приложения AIR с использованием платформы Flex используйте mx:WindowedApplication в качестве корневого элемента основного MXML-файла. (Можно использовать компонент mx:Application, но он не поддерживает все доступные в AIR функции.) Компонент WindowedApplication служит начальной точкой входа для приложения.

При запуске приложения AIR создает собственное окно, инициализирует платформу Flex и добавляет объект WindowedApplication в рабочую область окна. По завершении выполнения последовательности запуска WindowedApplication отправляет событие `applicationComplete`. Доступ к объекту окна рабочего стола осуществляется при помощи свойства `nativeWindow` экземпляра WindowedApplication.

Следующий пример создает простой компонент WindowedApplication с заданием его координат x и y:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    applicationComplete="placeWindow()">
    <mx:Script>
        <![CDATA[
            private function placeWindow():void{
                this.nativeWindow.x = 300;
                this.nativeWindow.y = 300;
            }
        ]]>
    </mx:Script>
    <mx:Label text="Hello World" horizontalCenter="0" verticalCenter="0"/>
</mx:WindowedApplication>
```

## Создание NativeWindow

**Adobe AIR 1.0 и более поздних версий**

Для создания NativeWindow необходимо передать объект NativeWindowInitOptions конструктору NativeWindow:

```
var options:NativeWindowInitOptions = new NativeWindowInitOptions();
options.systemChrome = NativeWindowSystemChrome.STANDARD;
options.transparent = false;
var newWindow:NativeWindow = new NativeWindow(options);
```

Окно не отображается, пока свойству `visible` не будет присвоено значение `true` или пока не будет вызван метод `activate()`.

После создания окна можно инициализировать его свойства и загрузить содержимое в окно, используя свойство рабочей области и методы списка отображения Flash.

Практически во всех случаях необходимо задать для свойства рабочей области `scaleMode` нового собственного окна значение `noScale` (при помощи константы `StageScaleMode.NO_SCALE`). Режимы масштабирования Flash предназначены для случаев, когда автор приложения не знает пропорции пространства отображения приложения заранее. Режимы масштабирования позволяют автору выбрать из нескольких зол меньшее: обрезать содержимое, растянуть или сжать его или выровнять при помощи пустого пространства. Так как пространством отображения в AIR (рамкой окна) можно управлять, можно подогнать размер окна под содержимое или содержимое под размер окна без ущерба для того или другого.

Режим масштабирования окон Flex и HTML автоматически принимает значение `noScale`.

***Примечание.*** *Определить максимальные и минимальные размеры окон, допустимые в текущей операционной системе, можно при помощи следующих статических свойств NativeWindow:*

```
var maxOSSize:Point = NativeWindow.systemMaxSize;
var minOSSize:Point = NativeWindow.systemMinSize;
```

## Создание окна HTML

**Adobe AIR 1.0 и более поздних версий**

Для создания окна HTML можно либо вызвать метод JavaScript `Window.open()`, либо вызвать метод `createRootWindow()` класса AIR HTMLLoader.

Содержимое HTML из любой изолированной программной среды безопасности может использовать стандартный метод JavaScript `Window.open()`. Если содержимое выполняется за пределами изолированной программной среды приложения, метод `open()` можно вызвать только в ответ на действия пользователя, например щелчок мышью или нажатие клавиши. При вызове метода `open()` создается окно с системным Chrome для отображения содержимого с заданного URL-адреса. Например:

```
newWindow = window.open("xmpl.html", "logWindow", "height=600, width=400, top=10, left=10");
```

***Примечание.*** *Можно расширить класс HTMLHost в ActionScript, чтобы настроить окно, созданное при помощи функции JavaScript `window.open()`. См. раздел «О расширении класса HTMLHost» на странице 1085.*

Содержимое, входящее в изолированную программную среду безопасности приложения, имеет доступ к более мощному методу создания окон — `HTMLLoader.createRootWindow()`. Используя этот метод, можно задать все параметры создания нового окна. Например, следующий код JavaScript создает окно типа lightweight без системного Chrome размером 300x400 пикселов:

```
 var options = new air.NativeWindowInitOptions();
options.systemChrome = "none";
options.type = "lightweight";

var windowBounds = new air.Rectangle(200,250,300,400);
newHTMLLoader = air.HTMLLoader.createRootWindow(true, options, true, windowBounds);
newHTMLLoader.load(new air.URLRequest("xmpl.html"));
```

***Примечание.*** *Если содержимое, загруженное новым окном, не входит в изолированную программную среду безопасности приложения, объект окна не имеет таких свойств AIR, как `runtime`, `nativeWindow` или `htmlLoader`.*

Если создать прозрачное окно, то содержимое SWF, внедренное на загружаемую в этом окне страницу HTML, будет отображаться не всегда. Необходимо присвоить параметру `wmode` объекта или тега внедрения, который ссылается на файл SWF, значение `opaque` или `transparent`. Так как для параметра `wmode` по умолчанию используется значение `window`, то содержимое SWF не отображается в прозрачных окнах. Содержимое PDF не будет отображаться в прозрачных окнах независимо от значения параметра `wmode`. (В версиях до AIR 1.5.2 содержимое SWF также не отображалось в прозрачных окнах.)

Окна, созданные при помощи метода `createRootWindow()`, не зависят от открывающего окна. Свойства `parent` и `opener` объекта JavaScript Window имеют значение `null`. Открывающее окно может получить доступ к объекту Window нового окна при помощи ссылки HTMLLoader, возвращенной функцией `createRootWindow()`. В контексте предыдущего примера выражение `newHTMLLoader.window` ссылалось бы на объект JavaScript Window созданного окна.

***Примечание.*** *Функцию `createRootWindow()` можно вызывать из JavaScript и из ActionScript.*

## Создание mx:Window

**Adobe AIR 1.0 и более поздних версий**

Для создания mx:Window можно создать MXML-файл с использованием mx:Window в качестве корневого тега, а также напрямую вызвать конструктор класса Window.

В следующем примере mx:Window создается и отображается путем вызова конструктора Window:

```
 var newWindow:Window = new Window();
newWindow.systemChrome = NativeWindowSystemChrome.NONE;
newWindow.transparent = true;
newWindow.title = "New Window";
newWindow.width = 200;
newWindow.height = 200;
newWindow.open(true);
```

## Добавление содержимого в окно

**Adobe AIR 1.0 и более поздних версий**

Способ добавления содержимого в окно AIR зависит от типа окна. Например, MXML и HTML позволяют описательно определить основное содержимое окна. Можно встроить ресурсы в SWF-файл приложения или загрузить из отдельных файлов приложения. Содержимое Flex, Flash и HTML может быть тут же создано и добавлено в окно динамически.

При загрузке содержимого SWF или содержимого HTML, содержащего сценарий JavaScript, необходимо принимать во внимание модель безопасности AIR. Любое содержимое, входящее в изолированную программную среду безопасности приложения, т. е. установленное с приложением и загружаемое при помощи схемы URL-адресов app:, имеет полные права доступа ко всем API-интерфейсам AIR. Содержимое, загружаемое не из этой изолированной программной среды, не имеет доступа к API-интерфейсам AIR. Содержимое JavaScript за пределами изолированной программной среды приложения не может использовать свойства `runtime`, `nativeWindow` или `htmlLoader` объекта JavaScript Window.

Для безопасного перекрестного выполнения сценариев можно использовать мост изолированной программной среды, который обеспечивает ограниченный интерфейс между содержимым приложения и другим содержимым. В случае с содержимым HTML можно отобразить страницы приложения в изолированную программную среду, отличную от среды приложения, чтобы разрешить коду этих страниц перекрестное выполнение сценариев во внешнем содержимом. См. раздел «Безопасность AIR» на странице 1160.

### Загрузка SWF-файла или изображения

Можно загружать SWF-файлы Adobe Flash или изображения в список отображения собственного окна при помощи класса `flash.display.Loader`:

```
package {
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.display.Loader;

    public class LoadedSWF extends Sprite
    {
        public function LoadedSWF(){
            var loader:Loader = new Loader();
            loader.load(new URLRequest("visual.swf"));
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE,loadFlash);
        }

        private function loadFlash(event:Event):void{
            addChild(event.target.loader);
        }
    }
}
```

***Примечание.*** *SWF-файлы, созданные при помощи ActionScript 1 или 2, имеют общие глобальные состояния, например определения классов, одноэлементные множества и глобальные переменные, если загружаются в одно и то же окно. Если для правильной работы такого SWF-файла нельзя изменять глобальные состояния, его нельзя многократно загружать в одно и то же окно или загружать в качестве другого SWF-файла, используя перекрывающиеся определения классов и переменные. Такое содержимое можно загружать в отдельные окна.*

### Загрузка содержимого HTML в NativeWindow

Для загрузки содержимого HTML в NativeWindow можно либо добавить объект HTMLLoader в рабочую область окна и загрузить содержимое HTML в объект HTMLLoader, либо создать окно, которое уже содержит объект HTMLLoader, при помощи метода `HTMLLoader.createRootWindow()`. Следующий пример отображает содержимое HTML в области отображения размером 300 на 500 пикселов в рабочей области собственного окна:

```
 //newWindow is a NativeWindow instance
var htmlView:HTMLLoader = new HTMLLoader();
htmlView.width = 300;
htmlView.height = 500;

//set the stage so display objects are added to the top-left and not scaled
newWindow.stage.align = "TL";
newWindow.stage.scaleMode = "noScale";
newWindow.stage.addChild( htmlView );

//urlString is the URL of the HTML page to load
htmlView.load( new URLRequest(urlString) );
```

Для загрузки страницы HTML в приложение Flex можно использовать компонент Flex HTML.

При использовании прозрачного окна содержимое SWF не отображается в HTML-файле (когда свойство `transparent` окна принимает значение `true`), если параметр `wmode` объекта или тега внедрения, который ссылается на SWF-файл, не принимает значение `opaque` или `transparent`. Так как для параметра `wmode` по умолчанию используется значение `window`, то содержимое SWF не отображается в прозрачных окнах. Содержимое PDF не будет отображаться в прозрачных окнах независимо от значения параметра `wmode`.

К тому же, содержимое SWF или PDF никогда не отображается при изменении масштаба или повороте объекта HTMLLoader, или если свойство `alpha` объекта HTMLLoader принимает значение, отличное от 1,0.

**Добавление содержимого SWF в качестве перекрытия окна HTML**

Поскольку окна HTML содержатся в экземпляре NativeWindow, экранные объекты Flash можно добавлять поверх слоя HTML в списке отображения и под него.

Для добавления экранного объекта поверх слоя HTML используйте метод `addChild()` свойства `window.nativeWindow.stage`. Метод `addChild()` добавляет содержимое поверх другого существующего в окне содержимого.

Для добавления экранного объекта под слой HTML используйте метод `addChildAt()` свойства `window.nativeWindow.stage`, передав нулевое значение параметру `index`. Присвоение нулевого индекса объекту перемещает существующее содержимое, в том числе страницу HTML, на один слой вверх и добавляет новое содержимое снизу. Чтобы содержимое под страницей HTML было видимым, необходимо присвоить свойству `paintsDefaultBackground` объекта `HTMLlLoader` значение `false`. Кроме того, все элементы страницы, для которых задан фоновый цвет, будут непрозрачными. Если задать фоновый цвет для тела страницы, вся страница будет непрозрачной.

Следующий пример иллюстрирует добавление экранных объектов Flash поверх страницы HTML и под нее. Пример создает две простые фигуры, добавляя одну из них под содержимое HTML, а другую — поверх этого содержимого. Пример также обновляет положение фигуры на основе события `enterFrame`.

```
 <html>
<head>
<title>Bouncers</title>
<script src="AIRAliases.js" type="text/javascript"></script>
<script language="JavaScript" type="text/javascript">
air.Shape = window.runtime.flash.display.Shape;

function Bouncer(radius, color){
    this.radius = radius;
    this.color = color;

    //velocity
    this.vX = -1.3;
    this.vY = -1;

    //Create a Shape object and draw a circle with its graphics property
    this.shape = new air.Shape();
    this.shape.graphics.lineStyle(1,0);
    this.shape.graphics.beginFill(this.color,.9);
    this.shape.graphics.drawCircle(0,0,this.radius);
    this.shape.graphics.endFill();

    //Set the starting position
    this.shape.x = 100;
    this.shape.y = 100;


    //Moves the sprite by adding (vX,vY) to the current position
    this.update = function(){
        this.shape.x += this.vX;
        this.shape.y += this.vY;

        //Keep the sprite within the window
```

```
        if( this.shape.x - this.radius < 0){
            this.vX = -this.vX;
        }
        if( this.shape.y - this.radius < 0){
            this.vY = -this.vY;
        }
        if( this.shape.x  + this.radius > window.nativeWindow.stage.stageWidth){
            this.vX = -this.vX;
        }
        if( this.shape.y  + this.radius > window.nativeWindow.stage.stageHeight){
            this.vY = -this.vY;
        }

    };
}

function init(){
    //turn off the default HTML background
    window.htmlLoader.paintsDefaultBackground = false;
    var bottom = new Bouncer(60,0xff2233);
    var top = new Bouncer(30,0x2441ff);

    //listen for the enterFrame event
    window.htmlLoader.addEventListener("enterFrame",function(evt){
        bottom.update();
        top.update();
    });

    //add the bouncing shapes to the window stage
    window.nativeWindow.stage.addChildAt(bottom.shape,0);
    window.nativeWindow.stage.addChild(top.shape);
}
</script>
<body onload="init();">
<h1>de Finibus Bonorum et Malorum</h1>
<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis
et quasi architecto beatae vitae dicta sunt explicabo.</p>
<p style="background-color:#FFFF00; color:#660000;">This paragraph has a background color.</p>
<p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis
praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias
excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui
officia deserunt mollitia animi, id est laborum et dolorum fuga.</p>
</body>
</html>
```

Данный пример дает первичное представление о некоторых усовершенствованных методах, для которых не существует границы между JavaScript и ActionScript в AIR. Если у вас нет опыта работы с экранными объектами ActionScript, см. раздел «Программирование отображаемого содержимого» на странице 162» в *руководстве разработчика ActionScript 3.0*.

## Пример: создание собственного окна

**Adobe AIR 1.0 и более поздних версий**

Следующий пример иллюстрирует создание собственного окна:

```
public function createNativeWindow():void {
    //create the init options
    var options:NativeWindowInitOptions = new NativeWindowInitOptions();
    options.transparent = false;
    options.systemChrome = NativeWindowSystemChrome.STANDARD;
    options.type = NativeWindowType.NORMAL;

    //create the window
    var newWindow:NativeWindow = new NativeWindow(options);
    newWindow.title = "A title";
    newWindow.width = 600;
    newWindow.height = 400;

    newWindow.stage.align = StageAlign.TOP_LEFT;
    newWindow.stage.scaleMode = StageScaleMode.NO_SCALE;

    //activate and show the new window
    newWindow.activate();
}
```

# Управление окнами

**Adobe AIR 1.0 и более поздних версий**

При помощи свойств и методов класса NativeWindow можно управлять внешним видом, поведением и жизненным циклом окон рабочего стола.

***Примечание.*** *При использовании среды Flex, как правило, лучше управлять поведением окон с помощью классов среды. Доступ к большинству свойств и методов NativeWindow можно получить из классов mx:WindowedApplication и mx:Window.*

## Получение экземпляра NativeWindow

**Adobe AIR 1.0 и более поздних версий**

Для управления окном необходимо сначала получить экземпляр окна. Экземпляр окна можно получить следующим образом:

- Для создания окна используется конструктор класса NativeWindow.

  ```
  var win:NativeWindow = new NativeWindow(initOptions);
  ```

- Свойство `nativeWindow` рабочей области окна.

  ```
  var win:NativeWindow = stage.nativeWindow;
  ```

- Свойство `stage` объекта display в окне.

  ```
  var win:NativeWindow = displayObject.stage.nativeWindow;
  ```

- Свойство `target` события NativeWindow, размещенного в окне.

  ```
  private function onNativeWindowEvent(event:NativeWindowBoundsEvent):void
  {
      var win:NativeWindow = event.target as NativeWindow;
  }
  ```

- Свойство `nativeWindow` для страницы HTML, отображаемой в окне.

  ```
  var win:NativeWindow = htmlLoader.window.nativeWindow;
  ```

- Свойства `activeWindow` и `openedWindows` объекта NativeApplication.

  ```
  var nativeWin:NativeWindow = NativeApplication.nativeApplication.activeWindow;
  var firstWindow:NativeWindow = NativeApplication.nativeApplication.openedWindows[0];
  ```

  `NativeApplication.nativeApplication.activeWindow` содержит ссылку на активное окно приложения (но возвращает `null`, если активное окно не является окном данного приложения AIR). Массив `NativeApplication.nativeApplication.openedWindows` содержит все окна приложения AIR, которые не были закрыты.

Поскольку объекты Flex mx:WindowedApplication и mx:Window являются экранными объектами, можно легко создать ссылку на окно приложения в MXML-файле при помощи свойства `stage`, как показано ниже:

```
 <?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
applicationComplete="init();">
    <mx:Script>
        <![CDATA[
            import flash.display.NativeWindow;

            public function init():void{
                var appWindow:NativeWindow = this.stage.nativeWindow;
                //set window properties
                appWindow.visible = true;
            }
        ]]>
    </mx:Script>
</WindowedApplication
```

***Примечание.*** *Пока компонент WindowedApplication или Window не будет добавлен в рабочую область окна при помощи платформы Flex, свойство `stage` этого компонента будет иметь значение `null`. Такое поведение соответствует поведению компонента Flex Application, но подтверждает тот факт, что невозможно получить доступ к рабочей области или к экземпляру NativeWindow через прослушивателей событий, которые произошли ранее в цикле инициализации компонентов WindowedApplication и Window, например `creationComplete`. Доступ к рабочей области и экземпляру NativeWindow при отправке события `applicationComplete` безопасен.*

## Активация, отображение и скрытие окон

**Adobe AIR 1.0 и более поздних версий**

Для активации окна необходимо вызвать метод NativeWindow `activate()`. Активация окна переносит его на передний план, окно получает фокус клавиатуры и мыши и при необходимости становится видимым за счет восстановления или присвоения свойству `visible` значения `true`. Активация окна не меняет порядок других окон в приложении. При вызове метода `activate()` окно отправляет событие `activate`.

Для отображения скрытого окна без его активации задайте для свойства `visible` значение `true`. При этом окно переходит на передний план, но не получает фокус.

Чтобы скрыть окно, задайте для свойства `visible` значение `false`. Скрывание окна подавляет отображение как самого окна, так и соответствующих значков панели задач, а также, в Mac OS X, входа в меню Windows.

При изменении видимости окна видимость всех остальных окон, также изменяется. Например, если окно было скрыто, все принадлежащие ему окна также скрываются.

***Примечание.*** *В Mac OS X невозможно полностью скрыть свернутое окно, которое имеет значок в экранной области дока. Если для свернутого окна свойство* `visible` *имеет значение* `false`*, значок этого окна в доке все равно отображается. При щелчке мышью по значку окно восстанавливается и отображается.*

## Изменение порядка отображения окон

**Adobe AIR 1.0 и более поздних версий**

AIR предоставляет несколько методов прямого изменения порядка отображения окон. Можно переместить окно в начало или конец порядка отображения, поверх другого окна или за него. В то же время пользователь может изменить порядок окон путем их активации.

Чтобы окно всегда отображалось поверх других окон, необходимо задать для свойства `alwaysInFront` значение `true`. Если такую настройку имеют несколько окон, то между ними происходит отдельная сортировка порядка отображения, но они всегда будут отображаться поверх окон, свойство `alwaysInFront` которых имеет значение false.

Окна, занимающие верхние позиции в порядке отображения, также отображаются поверх окон других приложений, даже если приложение AIR неактивно. Поскольку такое поведение может мешать пользователю, задание для свойства `alwaysInFront` значения `true` должно быть оправдано. Примерами оправданного использования такой настройки являются следующие:

- Временные всплывающие окна элементов управления, например подсказки, раскрывающиеся списки, пользовательские меню или комбинированные окна. Поскольку эти окна закрываются, когда теряют фокус, они не загораживают другие окна и не представляют помехи для пользователя.
- Сверхважные сообщения об ошибках и уведомления. Выдвижение окна уведомления на передний план оправдано, если в случае несвоевременной реакции пользователя может произойти неисправимое изменение. Однако большинство ошибок и уведомлений могут обрабатываться в обычном порядке отображения окон.
- Кратковременные всплывающие в системном лотке уведомления.

***Примечание.*** *AIR не навязывает правила использования свойства* `alwaysInFront`*. Однако если приложение нарушает рабочий процесс пользователя, велика вероятность того, что оно будет отправлено в корзину.*

Если окно является владельцем других окон, эти окна всегда располагаются перед ним. Если для окна, которое является владельцем других окон, вызывается метод `orderToFront()` или для свойства `alwaysInFront` устанавливается значение `true`, порядок подчиненных окон изменяется: окно-владелец располагается перед другими окнами, но подчиненные окна по-прежнему отображаются перед окном-владельцем.

Вызов методов изменения порядка окон для подчиненных окон действует обычным образом для окон, которые имеют одного владельца, но также при этом может изменяться порядок всей группы подчиненных окон относительно окон, не входящих в данную группу. Например, при вызове метода `orderToFront()` для подчиненного окна данное окно, его владелец и любые другие окна, которые принадлежат тому же владельцу, перемещаются на передний план.

Класс NativeWindow содержит следующие свойства и методы для установки порядка отображения окон:

| Член | Описание |
|---|---|
| свойство alwaysInFront | Указывает на то, входит ли окно в группу окон, отображаемых поверх других окон.<br><br>В большинстве случаев рекомендуется установить значение `false`. Изменение значения с `false` на `true` приводит к отображению окна поверх других окон (без его активации). Изменение значения с `true` на `false` располагает окно позади окон, входящих в группу отображаемых сверху окон, но все еще поверх остальных окон. Задание для этого свойства текущего значения окна не меняет порядок отображения окон.<br><br>Параметр `alwaysInFront` не влияет на окна, принадлежащие другому окну. |
| orderToFront() | Перемещает окно на передний план. |
| orderInFrontOf() | Помещает окно поверх другого указанного окна. |
| orderToBack() | Располагает окно позади других окон. |
| orderBehind() | Располагает окно позади другого указанного окна. |
| activate() | Перемещает окно на передний план (при этом окно становится видимым и получает фокус). |

***Примечание.*** *Если окно скрыто (`visible` имеет значение `false`) или свернуто, то методы определения порядка отображения не действуют.*

В операционной системе Linux различные диспетчеры окон реализуют различные правила в отношении порядка отображения окон:

- В некоторых таких диспетчерах окна служебных программ всегда отображаются поверх нормальных окон.
- В некоторых диспетчерах окон полноэкранные окна, в которых для свойства `alwaysInFront` установлено значение `true`, всегда отображаются поверх других окон, которые также имеют в свойстве `alwaysInFront` значение `true`.

## Закрытие окна

**Adobe AIR 1.0 и более поздних версий**

Чтобы закрыть окно, используйте метод `NativeWindow.close()`.

При закрытии окна выгружается его содержимое, но если другие объекты содержат ссылку на это содержимое, такие объекты содержимого не будут разрушены. Метод `NativeWindow.close()` выполняется асинхронно, в процессе закрытия приложение, содержащееся в окне, продолжает работать. Метод close отправляет событие close по завершении операции закрытия. Объект NativeWindow остается технически действующим, но при попытке получить доступ к большинству свойств и методов при закрытом окне выдается ошибка IllegalOperationError. Закрытое окно нельзя снова открыть. Чтобы проверить, закрыто ли окно, нужно обратиться к свойству `closed` окна. Если требуется просто скрыть окно, установите свойство `NativeWindow.visible` на значение `false`.

Если свойство `Nativeapplication.autoExit` имеет значение `true`, которое является значением по умолчанию, то приложение закрывается при закрытии последнего окна.

Все окна, которые имеют владельца, закрываются при закрытии окна-владельца. Подчиненные окна не отправляют событие closing и, соответственно, не могут предотвратить закрытие. Отправляется событие закрытия.

## Разрешение отмены операций с окном

**Adobe AIR 1.0 и более поздних версий**

Если окно использует системный Chrome, действия с окном, совершаемые пользователем, можно отменить, прослушав и отменив поведение по умолчанию соответствующих событий. Например, при нажатии пользователем кнопки закрытия на системном Chrome отправляется событие `closing`. Если какой-либо зарегистрированный прослушиватель вызывает метод `preventDefault()` события, окно не закрывается.

Если окно не использует системный Chrome, перед изменением уведомляющие об этих изменениях события не отправляются автоматически. Следовательно, при вызове методов для закрытия окна, изменения состояния окна или при задании свойств границ окна такое изменение невозможно отменить. Для уведомления компонентов приложения о предстоящих изменениях окна логикой приложения может быть предусмотрена отправка соответствующего уведомляющего события при помощи метода `dispatchEvent()` окна.

Например, следующая логика реализует отменяемый обработчик событий, вызванных кнопкой закрытия окна:

```
public function onCloseCommand(event:MouseEvent):void{
    var closingEvent:Event = new Event(Event.CLOSING,true,true);
    dispatchEvent(closing);
    if(!closingEvent.isDefaultPrevented()){
        win.close();
    }
}
```

Метод `dispatchEvent()` возвращает `false`, если прослушиватель вызывает метод события `preventDefault()`. Однако этот метод может возвращать значение `false` и по другим причинам, поэтому рекомендуется использовать метод `isDefaultPrevented()` явным образом, чтобы проверить, действительно ли отменяется изменение.

## Разворачивание, сворачивание и восстановление окна

**Adobe AIR 1.0 и более поздних версий**

Чтобы развернуть окно, используйте метод NativeWindow `maximize()`.

```
myWindow.maximize();
```

Чтобы свернуть окно, используйте метод NativeWindow `minimize()`.

```
myWindow.minimize();
```

Чтобы восстановить окно (т. е. восстановить его размер до сворачивания или разворачивания), используйте метод NativeWindow `restore()`.

```
myWindow.restore();
```

Окно, которое имеет владельца, сворачивается и восстанавливается при минимизации и восстановлении окна-владельца. При сворачивании подчиненное окно не отправляет никаких событий, так как сворачивается его владелец.

***Примечание.*** *Поведение при разворачивании окна AIR отличается от подобного стандартного поведения в Mac OS X. Вместо переключения между «стандартным» размером, определенным приложением, и последним размером, заданным пользователем, окна AIR переключаются между размером, который был последним задан приложением или пользователем, и общей площадью экрана, доступной для использования.*

В операционной системе Linux различные диспетчеры окон реализуют различные правила в отношении установки состояния отображения окон:

- В некоторых диспетчерах окон нельзя развернуть до максимального размера окна служебных программ.
- Если для окна задан максимальный размер, то некоторые окна не позволяют окну развернуться до максимального размера. Некоторые другие диспетчеры окон устанавливают максимальное состояние отображения, но не изменяют размер окна. В любом из случаев событие изменения состояния отображения отправляться не будет.
- Некоторые диспетчеры окон не учитывают параметры окон `maximizable` или `minimizable`.

***Примечание.*** *В Linux свойства окна изменяются асинхронно. Если состояние отображения изменяется в одной строке программы, а значение считывается в другой строке, считываемое значение будет соответствовать старому значению. На всех платформах объект NativeWindow отправляет событие* `displayStateChange` *при изменении состояния отображения. Если необходимо выполнить какое-либо действие в зависимости от нового состояния, для этого всегда следует использовать обработчик событий* `displayStateChange`*. См. раздел ««Прослушивание событий окна» на странице 983».*

## Пример: сворачивание, разворачивание, восстановление и закрытие окна

**Adobe AIR 1.0 и более поздних версий**

Следующее небольшое MXML-приложение иллюстрирует методы `maximize()`, `minimize()`, `restore()` и `close()` окна:

```
<?xml version="1.0" encoding="utf-8"?>

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


    <mx:Script>
    <![CDATA[
    public function minimizeWindow():void
    {
        this.stage.nativeWindow.minimize();
    }

    public function maximizeWindow():void
    {
        this.stage.nativeWindow.maximize();
    }

    public function restoreWindow():void
    {
        this.stage.nativeWindow.restore();
    }

    public function closeWindow():void
    {
        this.stage.nativeWindow.close();
    }
    ]]>
    </mx:Script>

    <mx:VBox>
        <mx:Button label="Minimize" click="minimizeWindow()"/>
        <mx:Button label="Restore" click="restoreWindow()"/>
        <mx:Button label="Maximize" click="maximizeWindow()"/>
        <mx:Button label="Close" click="closeWindow()"/>
    </mx:VBox>

</mx:WindowedApplication>
```

Следующий пример сценария ActionScript для Flash создает четыре гипертекстовых поля, инициирующих методы NativeWindow `minimize()`, `maximize()`, `restore()` и `close()`:

```
package
{
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.text.TextField;

    public class MinimizeExample extends Sprite
    {
        public function MinimizeExample():void
        {
            var minTextBtn:TextField = new TextField();
            minTextBtn.x = 10;
            minTextBtn.y = 10;
            minTextBtn.text = "Minimize";
            minTextBtn.background = true;
            minTextBtn.border = true;
            minTextBtn.selectable = false;
            addChild(minTextBtn);
            minTextBtn.addEventListener(MouseEvent.CLICK, onMinimize);

            var maxTextBtn:TextField = new TextField();
            maxTextBtn.x = 120;
            maxTextBtn.y = 10;
            maxTextBtn.text = "Maximize";
            maxTextBtn.background = true;
            maxTextBtn.border = true;
            maxTextBtn.selectable = false;
            addChild(maxTextBtn);
            maxTextBtn.addEventListener(MouseEvent.CLICK, onMaximize);

            var restoreTextBtn:TextField = new TextField();
            restoreTextBtn.x = 230;
            restoreTextBtn.y = 10;
            restoreTextBtn.text = "Restore";
            restoreTextBtn.background = true;
            restoreTextBtn.border = true;
            restoreTextBtn.selectable = false;
            addChild(restoreTextBtn);
            restoreTextBtn.addEventListener(MouseEvent.CLICK, onRestore);

            var closeTextBtn:TextField = new TextField();
            closeTextBtn.x = 340;
            closeTextBtn.y = 10;
            closeTextBtn.text = "Close Window";
            closeTextBtn.background = true;
            closeTextBtn.border = true;
            closeTextBtn.selectable = false;
            addChild(closeTextBtn);
```

```
            closeTextBtn.addEventListener(MouseEvent.CLICK, onCloseWindow);
        }
        function onMinimize(event:MouseEvent):void
        {
            this.stage.nativeWindow.minimize();
        }
        function onMaximize(event:MouseEvent):void
        {
            this.stage.nativeWindow.maximize();
        }
        function onRestore(event:MouseEvent):void
        {
            this.stage.nativeWindow.restore();
        }
        function onCloseWindow(event:MouseEvent):void
        {
            this.stage.nativeWindow.close();
        }
    }
}
```

## Изменение размера и перемещение окна

**Adobe AIR 1.0 и более поздних версий**

Если окно использует системный Chrome, то Chrome предоставляет элементы управления изменением размера окна и его перемещением на рабочем столе. Если окно не использует системный Chrome, необходимо добавить собственные элементы управления, при помощи которых пользователь может изменять размер окна и перемещать его.

***Примечание.*** *Для изменения размера или перемещения окна необходимо сначала получить ссылку на экземпляр NativeWindow. Сведения о том, как получить ссылку на окно, см. в разделе ««Получение экземпляра NativeWindow» на странице 972».*

### Изменение размера окна

Чтобы позволить пользователю интерактивно изменять размер окна, используйте метод `startResize()` класса NativeWindow. Когда этот метод вызывается событием `mouseDown`, операция изменения размера осуществляется при помощи мыши и завершается при получении операционной системой события `mouseUp`. При вызове метода `startResize()` ему передается аргумент, задающий край или угол, от которого осуществляется изменение.

Чтобы установить размер окна программно, установите желаемые размеры для свойств окна: `width`, `height` или `bounds`. При установке границ размер и позиция окна также могут одновременно измениться. Но порядок, в котором произойдут изменения, не гарантирован. Некоторые диспетчеры окон Linux не позволяют окнам расширяться за пределы экрана рабочего стола. В этих случаях размер окон может быть ограничен порядком, в котором устанавливались свойства. Хотя их совокупный эффект мог бы привести к допустимому изменению окна. Например, если изменяется одновременно высота и положение по оси Y окна в нижней части экрана, то изменение полной высоты может не произойти в том случае, когда изменение высоты было выполнено до изменения положения по оси Y.

***Примечание.*** *В Linux свойства окна изменяются асинхронно. Если размер окна изменяется в одной строке программы, а размеры считываются в следующей строке, считанные значения будут соответствовать старым настройкам. На всех платформах объект NativeWindow отправляет событие* `resize` *при изменении размера окна. Если необходимо выполнить какое-либо действие, например размещение элементов управления в окне, в зависимости от нового размера или состояния окна, для этого всегда необходимо использовать обработчик событий* `resize`*. См. раздел ««Прослушивание событий окна» на странице 983».*

Режим масштабирования рабочей области определяет поведение рабочей области окна и его содержимого при изменении размера окна. Следует иметь в виду, что режимы масштабирования рабочей области предназначены для работы, например, в веб-обозревателе, когда приложение не может управлять размером и пропорциями области отображения. В общем, наилучшие результаты достигаются при установке свойства `scaleMode` на значение `StageScaleMode.NO_SCALE`. Если требуется масштабирование содержимого окна, можно задать параметры `scaleX` и `scaleY` содержимого в ответ на изменение границ окна.

**Перемещение окна**

Чтобы переместить окно, не меняя его размера, используйте метод NativeWindow `startMove()`. Как и в случае с методом `startResize()`, при вызове метода `startMove()` событием `mouseDown` перемещение осуществляется при помощи мыши и завершается при получении операционной системой события `mouseUp`.

Дополнительные сведения о методах `startResize()` и `startMove()` см. в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

Чтобы переместить окно в программном коде, задайте для свойств `x`, `y` или `bounds` окна необходимое расположение. При установке границ размер и расположение окна также могут одновременно измениться.

***Примечание.*** *В Linux свойства окна изменяются асинхронно. Если окно перемещается в одной строке программы, а значение считывается в следующей строке, считываемое значение будет соответствовать старому значению. На всех платформах объект NativeWindow отправляет событие* `move` *при изменении расположения. Если необходимо выполнить какое-либо действие в зависимости от нового расположения окна, для этого всегда следует использовать обработчик событий* `move`*. См. раздел ««Прослушивание событий окна» на странице 983».*

## Пример: изменение размера и перемещение окон

**Adobe AIR 1.0 и более поздних версий**

Следующий пример иллюстрирует инициацию операций изменения размера и перемещения окна:

```
package
{
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.display.NativeWindowResize;

    public class NativeWindowResizeExample extends Sprite
    {
        public function NativeWindowResizeExample():void
        {
            // Fills a background area.
            this.graphics.beginFill(0xFFFFFF);
            this.graphics.drawRect(0, 0, 400, 300);
            this.graphics.endFill();

            // Creates a square area where a mouse down will start the resize.
            var resizeHandle:Sprite =
                createSprite(0xCCCCCC, 20, this.width - 20, this.height - 20);
            resizeHandle.addEventListener(MouseEvent.MOUSE_DOWN, onStartResize);

            // Creates a square area where a mouse down will start the move.
            var moveHandle:Sprite = createSprite(0xCCCCCC, 20, this.width - 20, 0);
            moveHandle.addEventListener(MouseEvent.MOUSE_DOWN, onStartMove);
        }

        public function createSprite(color:int, size:int, x:int, y:int):Sprite
        {
            var s:Sprite = new Sprite();
            s.graphics.beginFill(color);
            s.graphics.drawRect(0, 0, size, size);
            s.graphics.endFill();
            s.x = x;
            s.y = y;
            this.addChild(s);
            return s;
        }

        public function onStartResize(event:MouseEvent):void
        {
            this.stage.nativeWindow.startResize(NativeWindowResize.BOTTOM_RIGHT);
        }

        public function onStartMove(event:MouseEvent):void
        {
            this.stage.nativeWindow.startMove();
        }
    }
}
```

# Прослушивание событий окна

**Adobe AIR 1.0 и более поздних версий**

Для прослушивания событий, отправляемых окном, необходимо зарегистрировать прослушивателя для экземпляра окна. Например, для прослушивания события closing зарегистрируйте прослушиватель для окна следующим образом:

```
myWindow.addEventListener(Event.CLOSING, onClosingEvent);
```

При отправке события свойство `target` содержит ссылку на окно, отправляющее событие.

Большинство событий окна имеют два связанных сообщения. Первое сообщение сигнализирует о предстоящем изменении окна (которое еще может быть отменено), а второе сообщение сигнализирует о том, что изменение уже произошло. Например, при нажатии пользователем кнопки закрытия окна отправляется сообщение события closing. Если прослушиватели не отменили событие, окно закрывается и прослушивателям отправляется событие close.

Как правило, предупреждающие события, такие как `closing`, отправляются, только если событие было инициировано системным Chrome. При вызове метода `close()` окна, например, событие `closing` не отправляется автоматически — отправляется только событие `close`. Однако можно создать объект события closing и отправить его при помощи метода `dispatchEvent()` окна.

К событиям окна, которые отправляют объект Event, относятся следующие:

| Событие | Описание |
|---|---|
| activate | Отправляется при получении окном фокуса. |
| deactivate | Отправляется, когда окно теряет фокус. |
| closing | Отправляется при предстоящем закрытии окна. Автоматически это происходит только при нажатии кнопки закрытия на системном Chrome или (в Mac OS X) при вызове команды Quit. |
| close | Отправляется при закрытии окна. |

К событиям окна, которые отправляют объект NativeWindowBoundsEvent, относятся следующие:

| Событие | Описание |
|---|---|
| moving | Отправляется непосредственно перед изменением положения верхнего левого угла окна в результате перемещения, изменения размера или изменения состояния отображения окна. |
| move | Отправляется после изменения положения верхнего левого угла окна. |
| resizing | Отправляется непосредственно перед изменением ширины или высоты окна в результате изменения размера или изменения состояния отображения окна. |
| resize | Отправляется после изменения размера окна. |

Для событий NativeWindowBoundsEvent можно использовать свойства `beforeBounds` и `afterBounds` для определения границ окна до и после предстоящего или произошедшего изменения.

К событиям окна, которые отправляют объект NativeWindowDisplayStateEvent, относятся следующие:

| Событие | Описание |
|---|---|
| displayStateChanging | Отправляется непосредственно перед изменением состояния отображения окна. |
| displayStateChange | Отправляется после изменения состояния отображения окна. |

Для событий NativeWindowDisplayStateEvent можно использовать свойства `beforeDisplayState` и `afterDisplayState` для определения состояния отображения окна до и после предстоящего или произошедшего изменения.

В некоторых диспетчерах окон Linux событие изменения состояния отображения не отправляется, когда предпринимается попытка развернуть до максимального размера окно, уже имеющее максимальные параметры размера. (Для окна устанавливается максимальное состояние отображения, но его размер не изменяется.)

# Отображение окон во весь экран

**Adobe AIR 1.0 и более поздних версий**

Задание для свойства `displayState` рабочей области значения `StageDisplayState.FULL_SCREEN_INTERACTIVE` вызывает режим отображения окна во весь экран, и в этом режиме ввод с клавиатуры *разрешен*. (В содержимом SWF, выполняющемся в обозревателе, ввод с клавиатуры не разрешен.) Для выхода из полноэкранного режима пользователь должен нажать клавишу Escape.

***Примечание.*** *Некоторые диспетчеры окон в Linux не будут изменять размеры окон для заполнения экрана, если для окна установлен максимальный размер (но при этом удаляется системный Chrome окна).*

Например, следующий код Flex определяет простое приложение AIR, устанавливающее простой полноэкранный терминал:

```
 <?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    applicationComplete="init()" backgroundColor="0x003030" focusRect="false">
    <mx:Script>
        <![CDATA[
            private function init():void
            {
                stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
                focusManager.setFocus(terminal);
                terminal.text = "Welcome to the dumb terminal app. Press the ESC key to
exit..\n";
                terminal.selectionBeginIndex = terminal.text.length;
                terminal.selectionEndIndex = terminal.text.length;
            }
        ]]>
    </mx:Script>
    <mx:TextArea
        id="terminal"
        height="100%" width="100%"
        scroll="false"
        backgroundColor="0x003030"
        color="0xCCFF00"
        fontFamily="Lucida Console"
        fontSize="44"/>
</mx:WindowedApplication>
```

Следующий пример сценария ActionScript для Flash имитирует простой полноэкранный текстовый терминал:

```
    import flash.display.Sprite;
    import flash.display.StageDisplayState;
    import flash.text.TextField;
    import flash.text.TextFormat;

    public class FullScreenTerminalExample extends Sprite
    {
        public function FullScreenTerminalExample():void
        {
            var terminal:TextField = new TextField();
            terminal.multiline = true;
            terminal.wordWrap = true;
            terminal.selectable = true;
            terminal.background = true;
            terminal.backgroundColor = 0x00333333;

            this.stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;

            addChild(terminal);
            terminal.width = 550;
            terminal.height = 400;

            terminal.text = "Welcome to the dumb terminal application. Press the ESC key to
exit.\n_";

            var tf:TextFormat = new TextFormat();
            tf.font = "Courier New";
            tf.color = 0x00CCFF00;
            tf.size = 12;
            terminal.setTextFormat(tf);

            terminal.setSelection(terminal.text.length - 1, terminal.text.length);
        }
    }
```

# Глава 53. Экраны дисплея в AIR

**Adobe AIR 1.0 и более поздних версий**

При помощи класса Screen в Adobe® AIR® можно получить доступ к информации о экранах дисплея, подключенных к компьютеру или другому устройству.

### Дополнительные разделы справки

flash.display.Screen

## Основные сведения об экранах дисплея в AIR

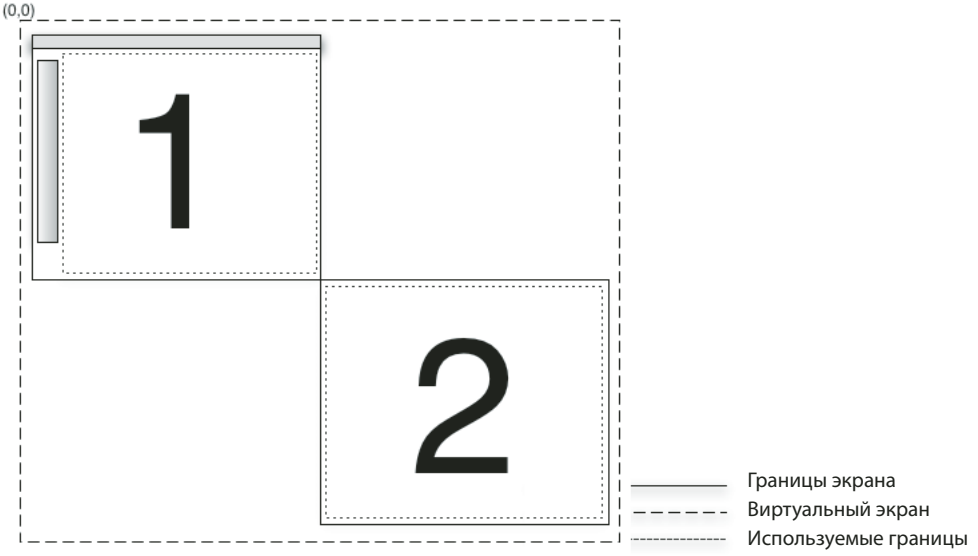
**Adobe AIR 1.0 и более поздних версий**

- Измерение виртуального рабочего стола (Flex)
- Измерение виртуального рабочего стола) (Flash)

API-интерфейс экрана содержит один класс, Screen, состоящий из статических членов для получения информации о системном экране и экземпляров для описания конкретного экрана.

К компьютерной системе может быть подключено несколько мониторов или дисплеев, которые могут соответствовать нескольким экранам рабочего стола, расположенным в виртуальном пространстве. Класс AIR Screen предоставляет информацию об экранах, их относительном расположении и доступном пространстве. Если несколько мониторов соответствуют одному и тому же экрану, существует только один экран. Если размер экрана превышает площадь отображения монитора, невозможно определить, какая часть экрана видима в данный момент.

Экран представляет собой независимую область дисплея настольного компьютера. Экраны имеют вид прямоугольников на виртуальном рабочем столе. Верхний левый угол экрана, обозначенного как основной дисплей, представляет собой начало отсчета в системе координат виртуального рабочего стола. Все значения, используемые при описании экрана, выражены в пикселах.




*При таком расположении экранов на виртуальном рабочем столе существует два экрана. Координаты верхнего левого угла основного экрана (№1) всегда равны (0,0). При изменении приоритета экранов и назначении экрана №2 в качестве основного координаты экрана №1 станут отрицательными. Строки меню, панели задач и доки выходят за границы полезной площади экрана.*

Дополнительные сведения о классе, методах, свойствах и событиях API-интерфейса экранов см. в справочнике ActionScript® 3.0 для платформы Flash® Platform.

# Перечисление экранов

**Adobe AIR 1.0 и более поздних версий**

Перечислить экраны виртуального рабочего стола можно при помощи следующих методов и свойств экрана:

| Метод или свойство | Описание |
|---|---|
| Screen.screens | Возвращает массив объектов Screen, описывающий доступные экраны. Порядок следования элементов в массиве не имеет значения. |
| Screen.mainScreen | Возвращает объект Screen основного экрана. В системе Mac OS X основным экраном является экран, отображающий строку меню. В системе Windows основным экраном является системный основной экран. |
| Screen.getScreensForRectangle() | Возвращает массив объектов Screen, описывающий экраны, входящие в заданный прямоугольник. Прямоугольник, передаваемый этому методу, обозначен при помощи координат виртуального рабочего стола, выраженных в пикселах. Если в заданный прямоугольник не попадает ни один экран, массив будет пуст. При помощи этого метода можно узнать, на каких экранах отображается окно. |

Не сохраняйте значения, возвращаемые методами и свойствами класса Screen. Доступные экраны и их расположение могут измениться пользователем или операционной системой в любое время.

В следующем примере API-интерфейс экрана используется для перемещения окна между экранами в ответ на нажатие клавиш со стрелками. Чтобы переместить окно в следующий экран, пример возвращает массив `screens` и сортирует его по вертикали или по горизонтали (в зависимости от нажатой клавиши со стрелкой). Затем код проходит через отсортированный массив, сопоставляя каждый экран координатам текущего экрана. Чтобы определить экран, на котором отображается окно, пример вызывает метод `Screen.getScreensForRectangle()`, передавая ему границы окна.

```
package {
    import flash.display.Sprite;
    import flash.display.Screen;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;

    public class ScreenExample extends Sprite
    {
        public function ScreenExample()
        {
                stage.align = StageAlign.TOP_LEFT;
                stage.scaleMode = StageScaleMode.NO_SCALE;

                stage.addEventListener(KeyboardEvent.KEY_DOWN,onKey);
        }

        private function onKey(event:KeyboardEvent):void{
            if(Screen.screens.length > 1){
                switch(event.keyCode){
                    case Keyboard.LEFT :
                        moveLeft();
                        break;
                    case Keyboard.RIGHT :
                        moveRight();
                        break;
                    case Keyboard.UP :
                        moveUp();
                        break;
                    case Keyboard.DOWN :
                        moveDown();
                        break;
                }
            }
        }

        private function moveLeft():void{
            var currentScreen = getCurrentScreen();
            var left:Array = Screen.screens;
            left.sort(sortHorizontal);
            for(var i:int = 0; i < left.length - 1; i++){
                if(left[i].bounds.left < stage.nativeWindow.bounds.left){
                    stage.nativeWindow.x +=
                        left[i].bounds.left - currentScreen.bounds.left;
                    stage.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;
                }
            }
        }
```

```
private function moveRight():void{
    var currentScreen:Screen = getCurrentScreen();
    var left:Array = Screen.screens;
    left.sort(sortHorizontal);
    for(var i:int = left.length - 1; i > 0; i--){
        if(left[i].bounds.left > stage.nativeWindow.bounds.left){
            stage.nativeWindow.x +=
                left[i].bounds.left - currentScreen.bounds.left;
            stage.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;
        }
    }
}

private function moveUp():void{
    var currentScreen:Screen = getCurrentScreen();
    var top:Array = Screen.screens;
    top.sort(sortVertical);
    for(var i:int = 0; i < top.length - 1; i++){
        if(top[i].bounds.top < stage.nativeWindow.bounds.top){
            stage.nativeWindow.x += top[i].bounds.left - currentScreen.bounds.left;
            stage.nativeWindow.y += top[i].bounds.top - currentScreen.bounds.top;
            break;
        }
    }
}

private function moveDown():void{
    var currentScreen:Screen = getCurrentScreen();

    var top:Array = Screen.screens;
    top.sort(sortVertical);
    for(var i:int = top.length - 1; i > 0; i--){
        if(top[i].bounds.top > stage.nativeWindow.bounds.top){
            stage.nativeWindow.x += top[i].bounds.left - currentScreen.bounds.left;
            stage.nativeWindow.y += top[i].bounds.top - currentScreen.bounds.top;
            break;
        }
    }
}

private function sortHorizontal(a:Screen,b:Screen):int{
    if (a.bounds.left > b.bounds.left){
        return 1;
    } else if (a.bounds.left < b.bounds.left){
```

```
                return -1;
            } else {return 0;}
        }

        private function sortVertical(a:Screen,b:Screen):int{
            if (a.bounds.top > b.bounds.top){
                return 1;
            } else if (a.bounds.top < b.bounds.top){
                return -1;
            } else {return 0;}
        }

        private function getCurrentScreen():Screen{
            var current:Screen;
            var screens:Array = Screen.getScreensForRectangle(stage.nativeWindow.bounds);
            (screens.length > 0) ? current = screens[0] : current = Screen.mainScreen;
            return current;
        }
    }
}
```

# Глава 54. Печать

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Среды выполнения Flash (такие как Adobe® Flash® Player и Adobe® AIR™) могут взаимодействовать с интерфейсом печати операционной системы, чтобы можно было передавать страницы спулеру печати. Каждая страница, отправляемая спулеру, может включать видимое, динамическое или находящееся за пределами экрана содержимое, включая значения баз данных и динамический текст. Кроме того, свойства класса flash.printing.PrintJob содержат значения, определяемые настройками принтера пользователя и позволяющие соответствующим образом отформатировать страницы.

Ниже описываются стратегии использования методов и свойств класса flash.printing.PrintJob с целью создания задания печати, считывания пользовательских настроек печати и корректирования задания печати на основе данных среды выполнения Flash и операционной системы пользователя.

## Основные сведения о печати

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В ActionScript 3.0 класс PrintJob служит для создания снимков содержимого на экране, которое преобразуется в представление для печати. В некотором смысле подготовка содержимого для печати — это то же самое, что и подготовка для отображения на экране: элементы требуется разместить и подогнать по размеру под требуемый макет. Однако печать имеет особенности, отличающие ее от раскладки на экране. Например, разрешение принтеров отличается от разрешения компьютерных мониторов. Содержимое экрана является динамическим и может изменяться, а напечатанное содержимое статично по определению. При планировании печати необходимо учитывать возможность многостраничной печати.

Несмотря на очевидность этих отличий, о них следует постоянно помнить, настраивая печать с помощью ActionScript. Точность печати зависит от комбинации значений, заданных разработчиком, и технических характеристик принтера пользователя. Класс PrintJob включает свойства, которые позволяют определить важные характеристики принтера пользователя.

### Важные понятия и термины

Ниже приводится список важных терминов, необходимых при работе с печатью.

**Спулер** Компонент операционной системы или ПО драйвера принтера, который отслеживает страницы, ожидающие печати, и отправляет их принтеру, когда он свободен.

**Ориентация страницы** Поворот печатаемого содержимого относительно страницы. Ориентация может быть горизонтальной (альбомной) и вертикальной (книжной).

**Задание печати** Страница или группа страниц, представляющая собой один отпечаток.

# Печать страницы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Управлять печатью можно с помощью экземпляра класса PrintJob. Чтобы напечатать простую страницу через проигрыватель Flash Player или AIR, необходимо последовательно вызвать четыре метода:

- `new PrintJob()`: создает новый экземпляр задания печати с указанным именем;
- `PrintJob.start()`: запускает процесс печати в операционной системе, вызывая диалоговое окно печати для пользователя, которое заполняется свойствами задания печати, доступными только для чтения.
- `PrintJob.addPage()`: содержит сведения о содержимом задания печати, включая объект Sprite (и всех его дочерних элементов), размер области печати и тип печати (векторный или растровый). Можно несколько раз вызывать метод `addPage()` для печати нескольких спрайтов на отдельных страницах;
- `PrintJob.send()`: отправляет страницу принтеру операционной системы.

Ниже приводится пример простого сценария задания печати (которое включает инструкции `package`, `import` и `class` для компиляции).

```
package
{
    import flash.printing.PrintJob;
    import flash.display.Sprite;

    public class BasicPrintExample extends Sprite
    {
        var myPrintJob:PrintJob = new PrintJob();
        var mySprite:Sprite = new Sprite();

        public function BasicPrintExample()
        {
            myPrintJob.start();
            myPrintJob.addPage(mySprite);
            myPrintJob.send();
        }
    }
}
```

***Примечание.*** *Этот пример демонстрирует основные элементы сценария задания печати и не включает обработки ошибок. Чтобы создать сценарий, правильно реагирующий на отмену задания печати пользователем, см. раздел «Работа с исключениями и возвращаемыми значениями» на странице 994.*

Чтобы очистить свойства объекта PrintJob, задайте переменной PrintJob значение `null` (например, `myPrintJob = null`).

# Задачи среды выполнения Flash и системная печать

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Так как среда выполнения Flash отправляет страницы интерфейсу печати операционной системы, необходимо понимать, какими задачами управляет среда выполнения Flash, а какими — собственный интерфейс печати операционной системы. Среда выполнения Flash может запускать задание печати, считывать некоторые параметры страницы с принтера, передавать содержимое задания печати в операционную систему и проверять, не отменено ли задание печати пользователем или системой. Другие процессы, такие как отображение диалоговых окон принтера, отмена отправленного задания печати или сообщение состояния принтера, управляются операционной системой. Среды выполнения Flash могут реагировать на возникновение проблем, связанных с запуском или форматированием задания печати, но могут сообщать лишь о некоторых свойствах или условиях интерфейса печати операционной системы. Код необходимо разработать таким образом, чтобы он реагировал на эти доступные свойства и условия.

## Работа с исключениями и возвращаемыми значениями

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Необходимо убедится, что метод `PrintJob.start()` возвращает значение `true`, прежде чем вызывать методы `addPage()` и `send()`, на тот случай, если пользователь решил отменить задание печати. Проверить, не отменены ли эти методы, перед продолжением выполнения можно с помощью инструкции `if`.

```
if (myPrintJob.start())
{
    // addPage() and send() statements here
}
```

Если метод `PrintJob.start()` возвращает значение `true`, пользователь выбрал команду «Печать» (либо она инициирована средой выполнения Flash, например Flash Player или AIR). В таком случае можно вызывать методы `addPage()` и `send()`.

Кроме того, чтобы облегчить управление процессом печати, среда выполнения Flash выдает исключения для метода `PrintJob.addPage()`, чтобы можно было перехватывать ошибки и предоставлять пользователю сведения и возможные пути решения. Если выполнение метода `PrintJob.addPage()` завершается ошибкой, можно вызвать другую функцию или остановить текущее задание печати. Для перехвата таких исключений нужно внедрить вызовы `addPage()` в оператор `try..catch`, как показано ниже. В этом примере `[params]` — это место для параметров, которые определяют фактическое содержимое, предназначенное для печати.

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error,
    }
    myPrintJob.send();
}
```

После запуска задания печати можно добавлять содержимое с помощью метода `PrintJob.addPage()` и следить за появлением исключений (например, если пользователь отменил задание печати). Если выдается ошибка, можно добавить в инструкцию `catch` логику, которая будет предоставлять пользователю (или среде выполнения Flash) сведения и возможные решения, или же можно отменить текущее задание печати. В случае успешного добавления страницы можно перейти к отправке страниц принтеру с помощью метода `PrintJob.send()`.

Если среда выполнения Flash сталкивается с проблемой при отправке задания печати принтеру (например, если принтер отключен), такое исключение также можно перехватить и получить дополнительные сведения или основу для возможных решений (например, путем вывода текстового сообщения или показа предупреждения в рамках анимации). Например, можно присвоить новое содержимое текстовому полю, внедренному в оператор `if..else`, как показано ниже.

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error.
    }
    myPrintJob.send();
}
else
{
    myAlert.text = "Print job canceled";
}
```

Рабочий пример см. в разделе ««Пример печати: масштабирование, обрезка и динамическая настройка» на странице 1003».

## Работа со свойствами страницы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Когда пользователь нажимает кнопку «OK» в диалоговом окне «Печать» и метод `PrintJob.start()` возвращает значение `true`, можно получить свойства, определенные параметрами принтера. К таким параметрам относится ширина и высота бумаги (`pageHeight` и `pageWidth`), а также ориентация содержимого на бумаге. Так как настройки принтера не контролируются средой выполнения Flash, их нельзя изменять. Однако их можно использовать для согласования содержимого, отправляемого принтеру, с текущими настройками. Дополнительные сведения см. в разделе ««Настройка размера, масштаба и ориентации страницы» на странице 997».

## Выбор векторной или растровой визуализации

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Задание печати можно настроить вручную так, чтобы каждая страница передавалась спулеру как векторная графика или как растровое изображение. В некоторых случаях векторная печать генерирует меньший по размеру файл подкачки и позволяет получить более качественное изображение, чем при растровой печати. Однако если содержимое включает растровое изображение и требуется сохранить альфа-прозрачность и цветовые эффекты, страницу необходимо печатать как растровое изображение. Кроме того, если принтер не использует драйвер PostScript, он автоматически преобразует векторную графику в растровое изображение.

Растровая печать задается путем передачи объекта PrintJobOptions в качестве третьего параметра `PrintJob.addPage()`.

Для версий Flash Player и AIR, предшествующих AIR 2, задайте для параметра `printAsBitmap` объекта PrintJobOptions значение `true`, как показано ниже:

```
var options:PrintJobOptions = new PrintJobOptions();
options.printAsBitmap = true;
myPrintJob.addPage(mySprite, null, options);
```

Если третий параметр не определен, задание печати будет использовать значение по умолчанию, то есть векторную печать.

Для AIR 2 и более поздних версий используйте свойство `printMethod` объекта для указания метода печати. Для этого свойства допустимы три значения, которые заданы как константы в классе PrintMethod:

- `PrintMethod.AUTO`: автоматический выбор наиболее подходящего метода печати в зависимости от печатаемого содержимого. Например, если на странице содержится текст, выбирается метод векторной печати. Однако, если изображение водяного знака с альфа-прозрачностью накладывается на текст, выбирается метод растровой печати для сохранения прозрачности.
- `PrintMethod.BITMAP`: принудительный выбор метода растровой печати вне зависимости от содержимого
- `PrintMethod.VECTOR`: принудительный выбор метода векторной печати вне зависимости от содержимого

## Время ожидания инструкций задания печати

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

ActionScript 3.0 не ограничивает объект PrintJob одним кадром (как это было в предыдущих версиях программы ActionScript). Однако поскольку операционная система отображает информацию о состоянии печати пользователю после нажатия кнопки «OK» в диалоговом окне «Печать», необходимо как можно быстрее вызвать методы `PrintJob.addPage()` и `PrintJob.send()`, чтобы отправить страницы в спулер печати. Задержка при обращении к кадру, содержащему вызов `PrintJob.send()`, приведет к задержке процесса печати.

В ActionScript 3.0 установлен предел времени ожидания, равный 15 секундам. Поэтому время между двумя значимыми инструкциями в последовательности задания печати не может превышать 15 секунд. Другими словами, предел времени ожидания сценария (15 секунд) применяется к следующим интервалам:

- между вызовом `PrintJob.start()` и первым вызовом `PrintJob.addPage()`;
- между одним вызовом `PrintJob.addPage()` и следующим вызовом `PrintJob.addPage()`;
- между последним вызовом `PrintJob.addPage()` и вызовом `PrintJob.send()`.

Если любой из этих интервалов длится более 15 секунд, следующий вызов метода `PrintJob.start()` для экземпляра PrintJob возвращает значение `false`, а следующее обращение к методу `PrintJob.addPage()` для этого экземпляра PrintJob вызывает в проигрывателе Flash Player или AIR исключение при выполнении.

# Настройка размера, масштаба и ориентации страницы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В разделе «Печать страницы» на странице 993 подробно описываются действия для выполнения базового задания печати, где вывод является непосредственным печатным эквивалентом размера экрана и позиции заданного спрайта. Однако принтеры имеют другое разрешение, и их настройки могут отрицательно сказаться на внешнем виде напечатанного спрайта.

Среда выполнения Flash может считывать параметры печати операционной системы, но эти свойства доступны только для чтения: их можно использовать в ходе настройки печати, но нельзя изменять. Например, можно узнать настройку принтера для размера страницы и соответствующим образом отформатировать содержимое. Также можно получить настройки принтера для полей и ориентации страницы. Чтобы адекватно реагировать на настройки принтера, укажите область печати, внесите изменения с учетом разницы между разрешением экрана и принтера или преобразуйте содержимое с учетом настроек принтера для размера или ориентации страницы.

## Использование прямоугольников ограничения области печати

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `PrintJob.addPage()` позволяет указывать область спрайта, предназначенную для печати. Второй параметр, `printArea`, задается как объект Rectangle. Передать значение для этого параметра можно тремя способами.

- Можно создать объект Rectangle с определенными свойствами, а затем использовать полученный прямоугольник в вызове `addPage()`, как показано ниже.

  ```
  private var rect1:Rectangle = new Rectangle(0, 0, 400, 200);
  myPrintJob.addPage(sheet, rect1);
  ```

- Если объект Rectangle еще не определен, это можно сделать в вызове метода, как показано ниже.

  ```
  myPrintJob.addPage(sheet, new Rectangle(0, 0, 100, 100));
  ```

- Если требуется передать значения для третьего параметра в вызове `addPage()`, а прямоугольник определять не нужно, можно задать второму параметру значение `null`, как показано ниже.

  ```
  myPrintJob.addPage(sheet, null, options);
  ```

## Сравнение точек и пикселов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Значения ширины и высоты прямоугольника задаются в пикселах. Принтер использует точки в качестве единиц измерения для печати. Точки имеют фиксированный физический размер (1/72 дюйма), но размер пиксела на экране зависит от разрешения конкретного экрана. Коэффициент преобразования между пикселами и точками зависит от настроек принтера и масштабирования объекта Sprite. Немасштабированный объект Sprite, имеющий ширину 72 пиксела, распечатывается с шириной 1 дюйм, при этом одна точка будет равна одному пикселу независимо от разрешения экрана.

Можно использовать следующие соответствия для преобразования дюймов или сантиметров в твипы или точки (твип равен 1/20 части точки):

- 1 точка = 1/72 дюйма = 20 твипов
- 1 дюйм = 72 пункта = 1440 твипов
- 1 сантиметр = 567 твипов

Если параметр `printArea` пропущен или неверно передан, распечатывается вся область спрайта.

## Масштабирование

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если перед печатью требуется отмасштабировать объект Sprite, установите свойства масштабирования (см. «Изменение размера и масштабирование объектов» на странице 193), прежде чем вызывать метод `PrintJob.addPage()`, а затем верните их в исходное состояние после выполнения печати. Масштаб объекта Sprite не связан со свойством `printArea`. Другими словами, если задать область печати размером 50 x 50 пикселов, будет напечатано 2500 пикселов. Если объект Sprite масштабируется, распечатываются те же 2500 пикселов, но объект Sprite при этом печатается в отмасштабированном размере.

Пример см. в разделе ««Пример печати: масштабирование, обрезка и динамическая настройка» на странице 1003».

## Печать страницы с альбомной или книжной ориентацией

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Поскольку проигрыватель Flash Player и AIR может обнаруживать параметры ориентации, в код ActionScript можно добавить логику для изменения размера содержимого или его поворота в соответствии с настройками принтера, как показано в следующем примере.

```
if (myPrintJob.orientation == PrintJobOrientation.LANDSCAPE)
{
    mySprite.rotation = 90;
}
```

***Примечание.*** *Если планируется считывать системные настройки для ориентации содержимого на странице, необходимо импортировать класс PrintJobOrientation. Класс PrintJobOrientation содержит константы, определяющие ориентацию содержимого на странице. Для импорта класса используется следующая инструкция:*

```
import flash.printing.PrintJobOrientation;
```

## Настройка высоты и ширины страницы

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Используя стратегию, подобную обработке настроек принтера для ориентации, можно считывать настройки для высоты и ширины страницы и учесть их, внедрив логику в инструкцию `if`. В качестве примера можно привести следующий код:

```
if (mySprite.height > myPrintJob.pageHeight)
{
    mySprite.scaleY = .75;
}
```

Кроме того, можно определить настройки полей, сравнив размеры страницы и бумаги, как показано ниже.

```
margin_height = (myPrintJob.paperHeight - myPrintJob.pageHeight) / 2;
margin_width = (myPrintJob.paperWidth - myPrintJob.pageWidth) / 2;
```

# Дополнительные приемы печати

**Adobe AIR 2 и более поздних версий**

Начиная с версии Adobe AIR 2, в классе PrintJob предусмотрены дополнительные свойства и методы и представлена поддержка трех дополнительных классов: PrintUIOptions, PaperSize и PrintMethod. Эти изменения расширяют возможности печати и позволяют авторам лучше управлять процессом печати. Список изменений:

- Диалоговые окна для настройки параметров страницы: могут отображаться стандартные и пользовательские диалоговые окна для настройки параметров страницы. Перед выполнением печати пользователь может задавать диапазон страниц, формат бумаги, ориентацию и масштаб.
- Просмотр печати: режим просмотра, в котором можно увидеть точный формат бумаги, поля и положение содержимого на странице.
- Ограничение печати: авторы могут запретить использование некоторых параметров печати, например выбор диапазона распечатываемых страниц.
- Параметры качества: авторы могут настроить качество печати документа и дать пользователю возможность выбирать разрешение и устанавливать параметры цвета.
- Несколько сеансов печати: теперь для нескольких сеансов печати может использоваться единственный экземпляр PrintJob. Каждый раз при отображении диалоговых окон печати и параметров страницы приложение предоставляет для них соответствующие настройки.

## Изменения в процессе печати

Новый процесс печати состоит из следующих этапов.

- `new PrintJob()`: создает экземпляр PrintJob (или использует уже существующий экземпляр). До начала и в процессе печати можно выбрать новые свойства и методы класса PrintJob, например `selectPaperSize()`.
- `PrintJob.showPageSetupDialog()`: (необязательный) отображение диалогового окна параметров страницы до начала печати.

- `PrintJob.start()` или `PrintJob.start2()`: в добавление к методу `start()`, метод `start2()` используется для запуска процесса буферизации заданий печати. Метод `start2()` позволяет выбрать отображение диалогового окна «Печать» и настройку этого окна, если оно показано.
- `PrintJob.addPage()`: добавление содержимого в задание печати. Осталось без изменений.
- `PrintJob.send()` или `PrintJob.terminate()`: отправка страниц на выбранный принтер или прерывание задания печати без отправки данных. Задания печати прерываются при обнаружении ошибки. Если выполнение экземпляра PrintJob прервано, его по-прежнему можно использовать. Вне зависимости от того, было ли задание печати отправлено на принтер или прервано, текущие параметры печати сохраняются при повторном использовании экземпляра PrintJob.

## Диалоговое окно «Параметры страницы»

Метод `showPageSetupDialog()` отображает диалоговое окно «Параметры страницы» операционной системы, если текущая среда выполнения его поддерживает. Всегда выполняйте проверку свойства `supportsPageSetupDialog` перед использованием данного метода. Ниже приведен простой пример.

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
//check for static property supportsPageSetupDialog of PrintJob class
if (PrintJob.supportsPageSetupDialog) {
    myPrintJob.showPageSetupDialog();
}
```

Этот метод может также быть вызван свойством класса PrintUIOptions для настройки отображаемых параметров в диалоговом окне «Параметры страницы». Можно задать диапазон страниц, доступных для печати. В следующем примере печать ограничена первыми тремя страницами.

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
if (PrintJob.supportsPageSetupDialog) {
    var uiOpt:PrintUIOptions = new PrintUIOptions();
    uiOpt.minPage = 1;
    uiOpt.maxPage = 3;
    myPrintJob.showPageSetupDialog(uiOpt);
}
```

## Изменение параметров печати

Параметры экземпляра PrintJob можно изменять в любое время после его создания. Данная особенность включает изменение параметров между вызовами `addPage()` и после отправки или прерывания задания печати. Некоторые параметры, например свойство `printer`, применяются ко всему заданию печати, а не к отдельным страницам. Эти параметры должны быть заданы перед вызовом метода `start()` или `start2()`.

Метод `selectPaperSize()` можно использовать для выбора формата бумаги по умолчанию в диалоговых окнах «Печать» и «Параметры страницы». Его также можно вызвать для изменения формата бумаги для определенного диапазона страниц во время выполнения задания печати. Метод вызывается с использованием констант, определенных в классе `PaperSize`. В следующем примере этот метод используется для выбора десятого размера конверта.

```
import flash.printing.PrintJob;
import flash.printing.PaperSize;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.selectPaperSize(PaperSize.ENV_10);
```

Свойство `printer` можно использовать для получения или установки имени текущего задания печати. По умолчанию задано имя стандартного принтера. Свойство `printer` имеет значение `null`, если принтеры недоступны или операционная система не поддерживает печать. Чтобы изменить принтер, сначала получите список доступных принтеров с помощью свойства `printers`. Это свойство имеет тип Vector, элементы String которого представляют собой имена доступных принтеров. Задайте для свойства `printer` одно из этих значений String, чтобы сделать этот принтер активным. Свойство `printer` активного задания печати изменить нельзя. Свойство нельзя изменить после успешного вызова метода `start()` или `start2()` и до завершения или прерывания задания печати. Ниже приводится пример настройки этого свойства.

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.printer = "HP_LaserJet_1";
myPrintJob.start();
```

Свойство `copies` получает заданное в диалоговом окне «Печать» операционной системы значение количества копий. Свойствам `firstPage` и `lastPage` присваиваются значения границ диапазона страниц. Свойство `orientation` получает параметр ориентации страницы. Значения этих свойств могут аннулировать значения в диалоговом окне «Печать». Следующий пример иллюстрирует настройку этих свойств.

```
import flash.printing.PrintJob;
import flash.printing.PrintJobOrientation;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.copies = 3;
myPrintJob.firstPage = 1;
myPrintJob.lastPage = 3;
myPrintJob.orientation = PrintJobOrientation.LANDSCAPE;
```

Следующие параметры только для чтения, связанные с `PrintJob`, могут предоставить полезные данные о текущих параметрах принтера.

- `paperArea`: прямоугольные границы носителя для печати, в точках.
- `printableArea`: прямоугольные границы области печати, в точках.
- `maxPixelsPerInch`: физическое разрешение данного принтера, в пикселах на дюйм.
- `isColor`: возможность текущего принтера выполнять цветную печать (возвращает значение `true`, если принтер поддерживает печать в цвете).

См. раздел ««Пример печати: настройка параметров страницы и печати» на странице 1005».

# Пример печати: печать нескольких страниц

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если печатается две и более страницы содержимого, можно связать каждую страницу с отдельным спрайтом (в данном случае `sheet1` и `sheet2`) и вызвать для каждого спрайта метод `PrintJob.addPage()`. Этот прием демонстрируется в следующем примере:

```
package
{
    import flash.display.MovieClip;
    import flash.printing.PrintJob;
    import flash.printing.PrintJobOrientation;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.geom.Rectangle;

    public class PrintMultiplePages extends MovieClip
    {
        private var sheet1:Sprite;
        private var sheet2:Sprite;

        public function PrintMultiplePages():void
        {
            init();
            printPages();
        }

        private function init():void
        {
            sheet1 = new Sprite();
            createSheet(sheet1, "Once upon a time...", {x:10, y:50, width:80, height:130});
            sheet2 = new Sprite();
            createSheet(sheet2, "There was a great story to tell, and it ended quickly.\n\nThe
end.", null);
        }

        private function createSheet(sheet:Sprite, str:String, imgValue:Object):void
        {
            sheet.graphics.beginFill(0xEEEEEE);
            sheet.graphics.lineStyle(1, 0x000000);
            sheet.graphics.drawRect(0, 0, 100, 200);
            sheet.graphics.endFill();

            var txt:TextField = new TextField();
            txt.height = 200;
            txt.width = 100;
            txt.wordWrap = true;
            txt.text = str;

            if (imgValue != null)
            {
                var img:Sprite = new Sprite();
                img.graphics.beginFill(0xFFFFFF);
                img.graphics.drawRect(imgValue.x, imgValue.y, imgValue.width, imgValue.height);
                img.graphics.endFill();
                sheet.addChild(img);
            }
            sheet.addChild(txt);
        }

        private function printPages():void
        {
            var pj:PrintJob = new PrintJob();
```

```
            var pagesToPrint:uint = 0;
            if (pj.start())
            {
                if (pj.orientation == PrintJobOrientation.LANDSCAPE)
                {
                    throw new Error("Page is not set to an orientation of portrait.");
                }

                sheet1.height = pj.pageHeight;
                sheet1.width = pj.pageWidth;
                sheet2.height = pj.pageHeight;
                sheet2.width = pj.pageWidth;

                try
                {
                    pj.addPage(sheet1);
                    pagesToPrint++;
                }
                catch (error:Error)
                {
                // Respond to error.
                }

                try
                {
                    pj.addPage(sheet2);
                    pagesToPrint++;
                }
                catch (error:Error)
                {
                    // Respond to error.
                }

                if (pagesToPrint > 0)
                {
                    pj.send();
                }
            }
        }
    }
}
```

# Пример печати: масштабирование, обрезка и динамическая настройка

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В некоторых случаях может потребоваться изменить размер (или другие свойства) экранного объекта при печати, чтобы компенсировать различия между тем, как он отображается на экране и печатается на бумаге. Настраивая свойства экранного объекта перед печатью (например, свойства `scaleX` и `scaleY`), помните, что если отмасштабированный объект становится больше прямоугольника, определенного для области печати, он обрезается. Кроме того, после печати страниц, вероятно, потребуется восстановить исходные свойства.

Следующий код масштабирует размеры экранного объекта `txt` (но не зеленого фонового окна), в результате чего текстовое поле обрезается по размерам заданного прямоугольника. После печати размер текстового поля восстанавливается для отображения на экране. Если пользователь отменит задание печати через диалоговое окно «Печать» операционной системы, содержимое в среде выполнения Flash изменится, чтобы сообщить пользователю о том, что задание отменено.

```
package
{
    import flash.printing.PrintJob;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.display.Stage;
    import flash.geom.Rectangle;

    public class PrintScaleExample extends Sprite
    {
        private var bg:Sprite;
        private var txt:TextField;

        public function PrintScaleExample():void
        {
            init();
            draw();
            printPage();
        }

        private function printPage():void
        {
            var pj:PrintJob = new PrintJob();
            txt.scaleX = 3;
            txt.scaleY = 2;
            if (pj.start())
            {
                trace(">> pj.orientation: " + pj.orientation);
                trace(">> pj.pageWidth: " + pj.pageWidth);
                trace(">> pj.pageHeight: " + pj.pageHeight);
                trace(">> pj.paperWidth: " + pj.paperWidth);
                trace(">> pj.paperHeight: " + pj.paperHeight);

                try
                {
                    pj.addPage(this, new Rectangle(0, 0, 100, 100));
                }
                catch (error:Error)
                {
                    // Do nothing.
                }
                pj.send();
            }
            else
            {
                txt.text = "Print job canceled";
            }
            // Reset the txt scale properties.
            txt.scaleX = 1;
            txt.scaleY = 1;
```

```
        }

        private function init():void
        {
            bg = new Sprite();
            bg.graphics.beginFill(0x00FF00);
            bg.graphics.drawRect(0, 0, 100, 200);
            bg.graphics.endFill();

            txt = new TextField();
            txt.border = true;
            txt.text = "Hello World";
        }

        private function draw():void
        {
            addChild(bg);
            addChild(txt);
            txt.x = 50;
            txt.y = 50;
        }
    }
}
```

# Пример печати: настройка параметров страницы и печати

**Adobe AIR 2 и более поздних версий**

В следующем примере инициализируются параметры PrintJob, определяющие число копий, формат бумаги (legal) и ориентацию страницы (альбомная). Согласно настройкам диалоговое окно «Параметры страницы» отображается перед диалоговым окном «Печать» и последующим выполнением задания печати.

```
package
{
    import flash.printing.PrintJob;
    import flash.printing.PrintJobOrientation;
    import flash.printing.PaperSize;
    import flash.printing.PrintUIOptions;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.display.Stage;
    import flash.geom.Rectangle;

    public class PrintAdvancedExample extends Sprite
    {
        private var bg:Sprite = new Sprite();
        private var txt:TextField = new TextField();
        private var pj:PrintJob = new PrintJob();
        private var uiOpt:PrintUIOptions = new PrintUIOptions();

        public function PrintAdvancedExample():void
        {
```

```
        initPrintJob();
        initContent();
        draw();
        printPage();
    }

    private function printPage():void
    {
        //test for dialog support as a static property of PrintJob class
        if (PrintJob.supportsPageSetupDialog)
        {
            pj.showPageSetupDialog();
        }
        if (pj.start2(uiOpt, true))
        {
            try
            {
                pj.addPage(this, new Rectangle(0, 0, 100, 100));
            }
            catch (error:Error)
            {
                // Do nothing.
            }
            pj.send();
        }
        else
        {
            txt.text = "Print job terminated";
            pj.terminate();
        }
    }

    private function initContent():void
    {
        bg.graphics.beginFill(0x00FF00);
        bg.graphics.drawRect(0, 0, 100, 200);
        bg.graphics.endFill();

        txt.border = true;
```

```
            txt.text = "Hello World";
        }

        private function initPrintJob():void
        {
            pj.selectPaperSize(PaperSize.LEGAL);
            pj.orientation = PrintJobOrientation.LANDSCAPE;
            pj.copies = 2;
            pj.jobName = "Flash test print";
        }

        private function draw():void
        {
            addChild(bg);
            addChild(txt);
            txt.x = 50;
            txt.y = 50;
        }
    }
}
```

# Глава 55. Geolocation

Если устройство поддерживает геопозиционирование, можно использовать API-интерфейс геопозиционирования для получения текущего географического расположения устройства. Если устройство поддерживает эту функцию, можно получить информацию о географическом расположении. Эта информация включает данные о высоте, точности, направлении, скорости и метке времени для последнего изменения расположения.

Класс Geolocation отправляет события `update` в ответ на реакцию датчика местоположения устройства. Событие `update` является объектом GeolocationEvent.

**Дополнительные разделы справки**

flash.sensors.Geolocation

flash.events.GeolocationEvent

## Обнаружения изменений геопозиционирования

Чтобы использовать датчик геопозиционирования, создайте экземпляр объекта Geolocation и зарегистрируйте отправляемое им событие `update`. Событие `update` является объектом события Geolocation. Событие имеет восемь свойств:

- `altitude`: высота в метрах.
- `heading`: направление движения (относительно истинного сервера) в градусах.
- `horizontalAccuracy`: точность по горизонтали в метрах.
- `latitude`: широта в градусах.
- `longitude`: долгота в градусах.
- `speed`: скорость в метрах в секунду.
- `timestamp`: число миллисекунд на момент события, начиная с инициализации среды выполнения.
- `verticalAccuracy`: точность нивелирования в метрах.

Свойство `timestamp` является объектом int. Остальные свойства являются объектами Number.

Здесь приводится основной пример отображения данных геопозиционирования в текстовом поле:

```
var geo:Geolocation;
if (Geolocation.isSupported)
{
    geo = new Geolocation();
    geo.addEventListener(GeolocationEvent.UPDATE, updateHandler);
}
else
{
    geoTextField.text = "Geolocation feature not supported";
}
function updateHandler(event:GeolocationEvent):void
{
    geoTextField.text = "latitude: " + event.latitude.toString() + "\n"
            + "longitude: " + event.longitude.toString() + "\n"
            + "altitude: " + event.altitude.toString()
            + "speed: " + event.speed.toString()
            + "heading: " + event.heading.toString()
            + "horizontal accuracy: " + event.horizontalAccuracy.toString()
            + "vertical accuracy: " + event.verticalAccuracy.toString()
}
```

Чтобы воспользоваться этим примером, перед использованием кода создайте текстовое поле `geoTextField` и добавьте его в список отображения.

Можно настроить необходимый интервал времени для событий геопозиционирования, вызвав метод `setRequestedUpdateInterval()` объекта Geolocation. В этом методе используется один параметр `interval`, который представляет собой запрошенный интервал обновления в миллисекундах:

```
var geo:Geolocation = new Geolocation();
geo.setRequestedUpdateInterval(10000);
```

Фактическое время между обновлениями геопозиционирования может быть больше или меньше этого значения. Любое изменение в интервале обновления затрагивает все зарегистрированные прослушиватели. При отсутствии вызова метода `setRequestedUpdateInterval()` приложение получает обновления на основе интервала, заданного в устройстве по умолчанию.

Пользователь может запретить приложению получать доступ к данным геопозиционирования. Например, в устройстве iPhone выводится запрос, когда приложение пытается получить данные геопозиционирования. В ответ на запрос пользователь может запретить приложению доступ к данным геопозиционирования. Объект Geolocation отправляет событие `status`, когда пользователь запрещает доступ к данным геопозиционирования. Кроме того, для свойства `muted` объекта Geolocation устанавливается значение `true`, когда датчик геопозиционирования недоступен. Объект Geolocation отправляет событие `status` при изменении свойства `muted`. Следующий код показывает способ обнаружения отсутствия доступа к данным геопозиционирования:

```
package
{
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.GeolocationEvent;
    import flash.events.MouseEvent;
    import flash.events.StatusEvent;
    import flash.sensors.Geolocation;
    import flash.text.TextField;
    import flash.text.TextFormat;

    public class GeolocationTest extends Sprite
    {

        private var geo:Geolocation;
        private var log:TextField;

        public function GeolocationTest()
        {
            super();
            stage.align = StageAlign.TOP_LEFT;
            stage.scaleMode = StageScaleMode.NO_SCALE;
            setUpTextField();

            if (Geolocation.isSupported)
            {
                geo = new Geolocation();
                if (!geo.muted)
                {
                    geo.addEventListener(GeolocationEvent.UPDATE, geoUpdateHandler);
                }
                geo.addEventListener(StatusEvent.STATUS, geoStatusHandler);
            }
            else
            {
                log.text = "Geolocation not supported";
            }
        }

        public function geoUpdateHandler(event:GeolocationEvent):void
        {
            log.text = "latitude : " + event.latitude.toString() + "\n";
            log.appendText("longitude : " + event.longitude.toString() + "\n");
        }

        public function geoStatusHandler(event:StatusEvent):void
        {
            if (geo.muted)
                geo.removeEventListener(GeolocationEvent.UPDATE, geoUpdateHandler);
            else
                geo.addEventListener(GeolocationEvent.UPDATE, geoStatusHandler);
        }

        private function setUpTextField():void
```

```
        {
            log = new TextField();
            var format:TextFormat = new TextFormat("_sans", 24);
            log.defaultTextFormat = format;
            log.border = true;
            log.wordWrap = true;
            log.multiline = true;
            log.x = 10;
            log.y = 10;
            log.height = stage.stageHeight - 20;
            log.width = stage.stageWidth - 20;
            log.addEventListener(MouseEvent.CLICK, clearLog);
            addChild(log);
        }
        private function clearLog(event:MouseEvent):void
        {
            log.text = "";
        }
    }
}
```

***Примечание.*** *Устройства iPhone первого поколения, которые не оснащены блоком GPS, отправляют события* `update` *только в редких случаях. В этих моделях объект Geolocation может отправлять только одно или два первоначальных события* `update`*. После этого сообщения* `update` *отправляются в случае значительного изменения данных.*

## Проверка наличия поддержки геопозиционирования

Используйте свойство `Geolocation.isSupported`, чтобы проверить наличие поддержки этой функции в среде выполнения:

```
if (Geolocation.isSupported)
{
    // Set up geolocation event listeners and code.
}
```

В настоящее время геопозиционирование поддерживается только в приложениях на основе ActionScript для iPhone и в пакете Flash Lite 4. Если в среде выполнения свойство `Geolocation.isSupported` принимает значение `true`, то поддержка геопозиционирования существует.

В некоторых моделях iPhone блок GPS не предусмотрен. В этих моделях используются другие средства (например, триангуляция мобильного телефона) для получения данных о географическом расположении. В этих моделях и в любых моделях iPhone с отключенным блоком GPS объект Geolocation может отправлять только одно или два первоначальных события `update`.

# Глава 56. Интернационализация приложений

**Flash Player 10.1 и более поздних версий, Adobe AIR 2.0 и более поздних версий**

Пакет flash.globalization облегчает создание международного программного обеспечения, отвечающего требованиям для разных языков и регионов.

### Дополнительные разделы справки

Пакет flash.globalization

«Локализация приложений» на странице 1030

Чарльз Бихис (Charles Bihis): Требуется локализовать приложения Flex/AIR?

## Основы интернационализации приложений

Иногда понятия «глобализация» и «интернационализация» могут выступать в качестве синонимов. Однако в большинстве случаев под глобализацией понимается совокупность инженерных и бизнес-процессов, а под интернационализацией — только совокупность первых.

Определения для некоторых ключевых понятий.

**Глобализация** Совокупность различных инженерных и бизнес-процессов по глобальной подготовке и внедрению продуктов в производственной среде предприятий. Глобализация включает в себя такие операции, как интернационализация, локализация, культурализация (приведение в соответствии с культурными ценностями), а также такие бизнес-процессы, как управление производством товара, финансовое планирование, маркетинг и урегулирование правовых вопросов. Вместо слова «глобализация» иногда используется сокращение *G11n* (от англ. Globalization, G — первая буква, затем следуют 11 букв, n — последняя буква). *«Глобализация — задача предприятий».*

**Интернационализация** Это инженерный процесс, направленный на унификацию продукта для поддержки различных языков, сценариев и принятых условных обозначений (включая форматы валют, дат и чисел, правила сортировки и многое другое) и исключения необходимости изменять его внешний вид или код. Данный процесс можно разделить на два подпроцесса: внедрение и локализация. В понятие «интернационализация» иногда вкладывается смысл фразы *«готов для использования во всем мире»* и используется сокращение *I18n*. *«Интернационализация — задача для инженеров».*

**Локализация** Процесс адаптации продукта или услуги к определенной языковой, культурной и локальной среде. Иногда понятие «локализация» заменяется сокращением *L10n*. *«Локализация — задача для переводчиков».*

**Культурализация** Инженерный процесс, направленных на создания или адаптацию определенных функций под реалии определенной культуры. К числу примеров можно отнести особенности публикации в японской культуре, присутствующие в Adobe InDesign, и поддержка особенностей Ханко в Adobe Acrobat.

Определения других понятий, имеющих отношение к интернационализации.

**Набор символов** Символы, используемые в данном языке или группе языков. В набор символов входят национальные символы, специальные символы (например, знаки препинания и математические символы), числа, а также символы управления компьютером.

**Упорядочение** Сортировка текста согласно принятым нормам данной локали.

**Локаль** Значение определяет нормы использования языковых и культурных элементов, применяемых к данной географической, политической или культурной области (в большинстве случаев — к стране). Данное значение представлено уникальным идентификатором локали (ИД локали). Идентификатор локали служит для поиска набора данных локали, который обеспечивает ее поддержку, включая единицы измерения, синтаксический анализ и форматирование чисел и дат и так далее.

**Набор ресурсов** Набор элементов, свойственных локали, в рамках которого планируется использовать приложение. Обычно сюда входят все элементы текста, являющиеся частью пользовательского интерфейса приложения. В рамках набора данные элементы переводятся на соответствующий язык данной локали. В набор ресурсов также могут входить другие настройки, изменяющие макет или поведение пользовательского интерфейса для определенной локали. Набор ресурсов может другие типы медиасодержимого, специфические для локали, или ссылки на них.

# Обзор пакета flash.globalization

**Flash Player 10.1 и более поздних версий, Adobe AIR 2.0 и более поздних версий**

Пакет flash.globalization объединяет в себе элементы, необходимые для поддержки особенностей языковых стандартов в используемой операционной системе. Это упрощает процесс создания приложений для пользователей из различных стран мира.

В данный пакет входят следующие основные классы.

- Класс Collator отвечает за сортировку и сопоставление строк.
- Класс CurrencyFormatter отвечает за преобразование чисел в формат валюты и считывает сумму и валюту из строк ввода.
- Класс DateTimeFormatter предназначен для форматирования значений дат.
- Класс LocaleID предназначен для получения информации об определенной локали
- Класс NumberFormatter отвечает на преобразование и считывание численных значений.
- Класс StringTools отвечает за преобразование строк с соблюдением норм использования заглавных и прописных букв.

## Пакет flash.globalization и местоположение ресурсов

Пакет flash.globalization не обрабатывает локализацию ресурсов. Однако можно использовать идентификатор локали пакета flash.globalization в качестве ключевого значения для получения локализованных ресурсов другими методами. Например, можно локализовать ресурсы приложения, созданного в Flex с помощью классов ResourceManager и ResourceBundle. Дополнительные сведения см. в статье «Локализация приложений Flex».

В пакете Adobe AIR 1.1 также доступны некоторые функции, помогающие локализовать приложения AIR, как описано в разделе ««Локализация приложений AIR» на странице 1031».

## Общий подход к интернационализации приложения

С помощью следующих действий приводится описание общего подхода высокого уровня к созданию международных приложений с использованием пакета flash.globalization:

1. Определите или задайте локаль.
2. Создайте экземпляр класса обслуживания (Collator, CurrencyFormatter, DateTimeFormatter, NumberFormatter или StringTools).
3. Проверьте наличие ошибок и устранения неисправностей с использованием свойств lastOperationStatus.
4. Отформатируйте и отобразите данные с использованием параметров, соответствующих локали.

Следующим этапом является загрузка и отображение строк и ресурсов пользовательского интерфейса, соответствующих локали. Этот этап может включать такие задачи, как:

- использование функций автокомпоновки для изменения размера элементов пользовательского интерфейса в соответствии с длиной строк;
- выбор правильных шрифтов и обеспечение поддержки резервных шрифтов;
- использование механизма обработки текста FTE для обеспечения поддержки других систем записи;
- обеспечение правильной обработки редакторов метода ввода.

## Проверка наличия ошибок и исправления неисправностей

Во всех классах обслуживания flash.globalization используется аналогичная модель выявления ошибок, а также шаблон перехода от недоступной запрошенной локали к локали, поддерживаемой операционной системой пользователя.

В следующем примере показан способ проверки наличия ошибок и устранения неисправностей при создании экземпляров классов обслуживания. В каждом классе обслуживания предусмотрено свойство lastOperationStatus, показывающее появление ошибок или предупреждений при вызове последнего метода.

```
var nf:NumberFormatter = new NumberFormatter("de-DE");
if(nf.lastOperationStatus != LastOperationStatus.NO_ERROR)
{
    if(nf.lastOperationStatus == LastOperationStatus.USING_FALLBACK_WARNING)
    {
        // perform fallback logic here, if needed
        trace("Warning - Fallback locale ID: " + nf.actualLocaleIDName);
    }
    else
    {
        // perform error handling logic here, if needed
        trace("Error: " + nf.lastOperationStatus);
    }
}
```

В этом примере выполняется отслеживание сообщения, появляющегося при использовании резервного идентификатора локали или при возникновении ошибки. При необходимости в приложении может быть реализован дополнительный алгоритм обработки ошибок. Например, можно вывести сообщение для пользователя или принудительно использовать в приложении определенную поддерживаемую локаль.

# Определение локали

**Flash Player 10.1 и более поздних версий, Adobe AIR 2.0 и более поздних версий**

Локаль задает определенную комбинацию языковых и культурных норм для страны или области.

Идентификатором локали можно безопасно управлять как строкой. Но можно использовать класс LocaleID для получения дополнительной информации, относящейся к локали.

Объект LocaleID создается следующим образом:

```
var locale:LocaleID = new LocaleID("es-MX");
```

После создания объекта LocaleID можно получить данные об идентификаторе локали. Для этого следует использовать методы `getKeysAndValues()`, `getLanguage()`, `getRegion()`, `getScript()`, `getVariant()`, `isRightToLeft()` и свойство `name`.

Значения, полученные с помощью этих методов и свойств, могут содержать дополнительную информацию о локали, извлечение которой из идентификатора локали напрямую может быть невозможно.

Когда приложение создает службу локали, например средство форматирования даты, должна быть указана ожидаемая локаль. Список поддерживаемых локалей варьируется в зависимости от операционной системы, поэтому запрошенная локаль может быть недоступной.

Сначала проигрыватель Flash Player пытается сопоставить код языка запрошенной локали. Затем предпринимается попытка уточнения локали путем нахождения соответствующего набора правил правописания и региона. Например:

```
var loc:LocaleID = new LocaleID("es");
trace(loc.getLanguage()); // es
trace(loc.getScript()); // Latn
trace(loc.getRegion()); // ES
```

В этом примере конструктор `LocaleID()` получает данные о локали, которая лучше всего соответствует запрошенному коду языка es для данного пользователя.

## Установка идентификатора локали

Существует несколько способов установки текущей локали для приложения, включая:

- Жесткое кодирование одного идентификатора локали в приложении. Это общий подход, однако он не поддерживает интернационализацию приложения.
- Использование настроек идентификатора локали из пользовательской операционной системы или браузера либо других пользовательских настроек. Применение этого метода обычно позволяет получить оптимальные настройки локали, однако точность удается обеспечить не всегда. Существует вероятность того, что настройки операционной системы не отражают фактические настройки пользователя. Например, пользователь может работать на общем компьютере и не иметь возможности установить предпочитаемую локаль операционной системы.
- После задания идентификатора локали на основе пользовательских настроек предоставьте пользователю возможность выбора локали из списка поддерживаемых локалей. Обычно эта стратегия является самым оптимальным вариантом, если приложение может поддерживать несколько локалей.

Этот вариант можно реализовать следующим образом:

1 Получите список предпочитаемых пользователем локалей или языков из профиля пользователя, настроек браузера, настроек операционной системы или файла cookie. (В приложении необходимо самостоятельно реализовать этот алгоритм. Библиотека flash.globalization не поддерживает прямое чтение этих настроек.)

2 Определите, какие из этих локалей поддерживаются приложением, и выберите наиболее подходящую локаль по умолчанию. Используйте метод LocaleID.determinePreferredLocales() для нахождения наиболее подходящей локали на основе предпочитаемых локалей и локалей, поддерживаемых операционной системой.

3 Предоставьте пользователю способ изменения параметра локали по умолчанию в случае, если локаль по умолчанию является неподходящей.

## Ограничение на другие локали и классы языков

Класс `fl.lang.Locale` позволяет заменить текстовые строки на основе локали с использованием пакетов ресурсов, содержащих строковые значения. Однако этот класс не поддерживает другие функции интернационализации, такие как форматирование чисел, валют или даты, сортировка и сопоставление и т. д. Помимо того, этот класс доступен только в среде Flash Professional.

Можно также получить текущий параметр языкового кода для операционной системы с помощью свойства `flash.system.Capabilities.language`. Однако при этом извлекается только двухсимвольный языковой код ISO 639-1, а не полный идентификатор локали, и поддерживается только определенный набор локалей.

В AIR 1.5 можно использовать свойство `flash.system.Capabilities.languages`. Это свойство содержит массив предпочитаемых пользователем языков интерфейса. Таким образом, на него не распространяются ограничения `Capabilities.language`.

# Формат чисел

**Flash Player 10.1 и более поздних версий, Adobe AIR 2.0 и более поздних версий**

Формат отображения числовых значений изменяется в зависимости от региона. Например, здесь показано форматирование числа 123456,78 в определенных локалях:

| Локаль | Формат числа |
|---|---|
| en-US (английский, США) | -123,456.78 |
| de-DE (немецкий, Германия) | -123.456,78 |
| fr-FR (французский, Франция) | -123 456,78 |
| de-CH (немецкий, Швейцария) | -123'456.78 |
| en-IN (английский, Индия) | -1,23,456.78 |
| Большинство арабских локалей | 123,456.78- |

Форматы чисел зависят от множества факторов, в том числе:

- Разделители. Десятичный разделитель ставится между целой и дробной частями числа. Он может представлять собой точку, запятую или другой символ. В качестве разделителя групп данных или разделителя тысяч может использоваться точка, запятая, неразрывный пробел или другой символ.

- Шаблоны группировки. Число знаков между каждым разделителем групп слева от десятичной запятой может быть равно двум, трем или другому значению.
- Указатели отрицательных чисел. Отрицательные числа могут быть показаны со знаком минуса слева или справа от числа или в скобках в финансовых приложениях. Например, отрицательное число 19 может отображаться как -19, 19- или (19).
- Нули в начале или конце. Согласно некоторым культурным нормам при отображении чисел в начало или конец добавляются нули. Например, значение 0,17 может отображаться, помимо прочего, как .17, 0.17 или 0.170.
- Наборы символов цифр. Во многих языках, включая Хинди, арабский и японский, используются различные наборы символов цифр. Пакет flash.globalization поддерживает любые наборы символов цифр, которые можно сопоставить с цифрами 0-9.

Класс NumberFormatter позволяет учесть все эти факторы при форматировании числовых значений.

## Использование класса NumberFormatter

Класс NumberFormatter форматирует числовые значения (типа int, uint или Number) в соответствии с нормами, предусмотренными для определенной локали.

В следующем примере показан самый простой способ форматирования числа с использованием свойств форматирования по умолчанию, предоставляемых операционной системой пользователя:

```
var nf:NumberFormatter = new NumberFormatter(LocaleID.DEFAULT);
trace(nf.formatNumber(-123456.789))
```

Результат варьируется в зависимости от пользовательских настроек локали и пользовательских предпочтений. Например, если пользовательской локалью является fr-FR, отформатированным значением будет:

-123.456,789

Если необходимо только отформатировать число в соответствии с определенной локалью вне зависимости от настроек пользователя, задайте локаль явно. Например:

```
var nf:NumberFormatter = new NumberFormatter("de-CH");
trace(nf.formatNumber(-123456.789));
```

В данном случае получен такой результат:

-123'456.789

В методе formatNumber() в качестве параметра используется тип Number. В классе NumberFormatter также предусмотрен метод formatInt(), использующий тип int в качестве входного значения, и метод formatUint(), работающий с типом uint.

Можно явно управлять алгоритмом форматирования путем задания свойств класса NumberFormatter, как показано в этом примере:

```
var nf:NumberFormatter = new NumberFormatter("de-CH");
nf.negativeNumberFormat = 0;
nf.fractionalDigits = 5;
nf.trailingZeros = true;
nf.decimalSeparator = ",";
nf.useGrouping = false;
trace(nf.formatNumber(-123456.789)); //(123456.78900)
```

В этом примере сначала создается объект NumberFormatter, а затем:

- задается запись отрицательных чисел в скобках (как в финансовых приложениях);
- задается число знаков после десятичного разделителя, равное 5;
- задается использование нулей в конце для вывода пяти десятичных знаков;
- задается использование запятой в качестве десятичного разделителя;
- передаются инструкции в средство форматирования о запрете использования разделителей групп.

***Примечание.*** *При изменении некоторых из этих свойств полученный формат чисел не соответствует предпочтительному формату для указанной локали. Некоторые из этих свойств следует использовать, только если данные о локали не имеют значения, если необходимо получить полный контроль над одним аспектом формата, таким как число замыкающих нулей, или если пользователь сам запрашивает изменение, например, через панель управления Windows.*

## Считывание числовых значений из строк

Класс NumberFormatter также извлекает числовые значения из строк, соответствующих требованиям к форматированию для локали. Метод NumberFormatter.parseNumber() извлекает одно числовое значение из строки. Например:

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String =  "-1,234,567.890"
var parsedNumber:Number = nf.parseNumber(inputNumberString);
trace("Value:" + parsedNumber); // -1234567.89
trace("Status:" + nf.lastOperationStatus); // noError
```

Метод parseNumber() обрабатывает строку, содержащую только цифры и символы форматирования числа, такие как знаки отрицательного числа и разделители. Если строка содержит другие символы, устанавливается код ошибки:

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String =  "The value is 1,234,567.890"
var parsedNumber:Number = nf.parseNumber(inputNumberString);
trace("Value:" + parsedNumber); // NaN
trace("Status:" + nf.lastOperationStatus); // parseError
```

Чтобы извлечь числа из строки, содержащей дополнительные буквенные символы, используйте метод NumberFormatter.parse():

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String = "The value is 123,456,7.890";
var parseResult:NumberParseResult = nf.parse(inputNumberString);
trace("Value:" + parseResult.value); // 1234567.89
trace("startIndex: " + parseResult.startIndex); // 14
trace("Status:" + nf.lastOperationStatus); // noError
```

Метод parse() возвращает объект NumberParseResult, содержащий проанализированное числовое значение в свойстве значения. Свойство startIndex показывает указатель первого найденного числового символа. Можно использовать свойства startIndex и endIndex для извлечения частей строки до или после цифр.

# Формат валюты

**Flash Player 10.1 и более поздних версий, Adobe AIR 2.0 и более поздних версий**

Форматы отображения значений валют так же значительно изменяются, как и форматы чисел. Например, здесь показано форматирование долларов США ($123456,78) в определенных локалях:

| Локаль | Формат числа |
|---|---|
| en-US (английский, США) | $123,456.78 |
| de-DE (немецкий, Германия) | 123.456,78 $ |
| en-IN (английский, Индия) | $ 1,23,456.78 |

Форматирование валют зависит от всех факторов, действующих для форматирования чисел, а также следующих дополнительных факторов:

- Код ISO валюты. Трехбуквенный код валюты по ISO 4217 для используемых локалей, например USD или EUR.
- Символ валюты. Обозначение или строка валюты для используемых локалей, например $ или €.
- Формат отрицательных денежных сумм. Местоположение символа валюты и отрицательного символа или круглых скобок относительно числовой части денежной суммы.
- Формат положительных денежных сумм. Местоположение символа валюты относительно числовой части денежной суммы.

## Использование класса CurrencyFormatter

Класс CurrencyFormatter форматирует числовые значения в строках, содержащих строки валют или форматированные числа, в соответствии с нормами для определенной локали.

При создании экземпляра нового объекта CurrencyFormatter в качестве его валюты задается валюта, используемая по умолчанию в указанной локали.

В следующем примере показан объект CurrencyFormatter, созданный с использованием немецкой локали и подразумевающий расчет денежных сумм в евро:

```
var cf:CurrencyFormatter = new CurrencyFormatter( "de-DE" );
trace(cf.format(1234567.89)); // 1.234.567,89 EUR
```

В большинстве случаев не следует полагаться на валюту по умолчанию для локали. Если пользовательская локаль по умолчанию не поддерживается, класс CurrencyFormatter назначает резервную локаль. Резервная локаль может иметь другую валюту по умолчанию. Помимо этого, обычно необходимо, чтобы формат валюты отображался правильно для пользователя, даже если денежные суммы указываются не в местной валюте. Например, пользователь из Канады может захотеть посмотреть цены немецкой компании в евро, но отформатированные в соответствии с канадским языковым стандартом.

Метод CurrencyFormatter.setCurrency() задает точные строку валюты и символ валюты, которые необходимо использовать.

Следующий код показывает денежные суммы в евро для пользователей во французской части Канады:

```
var cf:CurrencyFormatter = new CurrencyFormatter( "fr-CA" );
cf.setCurrency("EUR", "€");
trace(cf.format(1234567.89)); // 1.234.567,89 EUR
```

Метод setCurrency() можно также использовать для предотвращения путаницы путем задания точных символов валют. Например:

```
cf.setCurrency("USD","US$");
```

По умолчанию метод format() отображает трехсимвольный код валюты по ISO 4217 вместо символа валюты. Коды ISO 4217 являются точными и не требуют локализации. Однако многие пользователи предпочитают видеть символы валют, а не коды по ISO.

Класс CurrencyFormatter может помочь решить, какое обозначение следует использовать в форматированной строке валюты: символ валюты, такой как знак доллара или евро, или трехсимвольную строку валюты по ISO, такую как USD или EUR. Например, для пользователя в Канаде сумма в канадских долларах может отображаться как $200. А для пользователя в США она может отображаться как CAD 200. Используйте метод formattingWithCurrencySymbolIsSafe() для определения, будет ли символ валюты для денежных сумм неоднозначным или неправильным с учетом параметров локали пользователя.

В следующем примере значение в евро преобразуется формат для локали en-US. В зависимости от локали пользователя в строке вывода используется код валюты по ISO или символ валюты.

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-CA");

if (cf.formattingWithCurrencySymbolIsSafe("USD"))
{
    trace(cf.format(1234567.89, true)); // $1,234,567.89
}
else
{
    cf.setCurrency("USD", "$");
    trace(cf.format(1234567.89)); // USD1,234,567.89
}
```

## Синтаксический анализ строк, содержащих суммы в валюте

Класс CurrencyFormatter также позволяет извлекать денежную сумму и строку валюты из строки ввода, отвечающей требованиям к форматированию для локали. Метод CurrencyFormatter.parse() хранит проанализированную сумму и строку валюты в объекте CurrencyParseResult, как показано здесь:

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-US" );
var inputCurrencyString:String = "(GBP 123,56,7.890)";
var parseResult:CurrencyParseResult = cf.parse(inputCurrencyString);
trace("parsed amount: " + parseResult.value); // -1234567.89
trace("currencyString: " + parseResult.currencyString ); // GBP
```

Часть строки ввода со строкой валюты может содержать символ валюты, код валюты по ISO и дополнительные текстовые символы. Расположения строки валюты, указателя отрицательного числа и числового значения соответствуют форматам, заданным свойствами negativeCurrencyFormat и positiveCurrencyFormat. Например:

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-US" );
var inputCurrencyString:String = "Total $-123,56,7.890";
var parseResult:CurrencyParseResult = cf.parse(inputCurrencyString);
trace("status: " + cf.lastOperationStatus ); // parseError
trace("parsed amount: " + parseResult.value); // NaN
trace("currencyString: " + parseResult.currencyString ); //
cf.negativeCurrencyFormat = 2;
parseResult = cf.parse(inputCurrencyString);
trace("status: " + cf.lastOperationStatus ); // noError
trace("parsed amount: " + parseResult.value); // -123567.89
trace("currencyString: " + parseResult.currencyString ); // Total $
```

В этом примере строка ввода содержит строку валюты, за которой следуют знак минуса и число. Однако значение negativeCurrencyFormat по умолчанию для локали en-US определяет, что сначала стоит указатель отрицательного числа. Поэтому метод parse() создает ошибку, и проанализированным значением является NaN.

После задания для свойства negativeCurrencyFormat значения 2, которое задает использование строки валюты в начале, метод parse() выполняется успешно.

# Формат даты и времени

**Flash Player 10.1 и более поздних версий, Adobe AIR 2.0 и более поздних версий**

Формат отображения значений даты и времени также значительно изменяется в зависимости от региона. Например, здесь приводится способ отображения значения даты (второе января 1962 года, 13:01) в коротком формате для определенных локалей:

| Локаль | Формат даты и времени |
|---|---|
| en-US (английский, США) | 1/2/62 1:01pm |
| fr-FR (французский, Франция) | 2/1/62 13:01 |
| ja-JP (японский, Япония) | 1962/2/1 13:01 |

## Использование класса DateTimeFormatter

Класс DateTimeFormatter форматирует значения Date в строки даты и времени в соответствии с нормами, принятыми для определенной локали.

Форматирование осуществляется на основе шаблонной строки, содержащей последовательность букв, заменяемых значениями даты или времени. Например, в шаблоне «yyyy/MM» символы «yyyy» заменяются на четырехзначное представление года, после которого следует символ «/» и двухзначное числовое представление месяца.

Шаблонная строка должна быть явно задана с помощью метода setDateTimePattern(). Однако лучше использовать автоматическое задание шаблона в соответствии с пользовательской локалью и настройками операционной системы. Такой метод помогает обеспечить соответствие культурным нормам

Класс DateTimeFormatter может представлять значения даты и времени в трех стандартных стилях (LONG, MEDIUM и SHORT) и также поддерживает использование шаблона CUSTOM. Один стиль можно использовать для значения даты, а второй — для значения времени. Фактические шаблоны, используемые для каждого стиля, незначительно варьируются в зависимости от операционной системы.

Стили можно определить при создании объекта DateTimeFormatter. Если параметры стиля не указаны, для них по умолчанию задается значение DateTimeStyle.LONG. В дальнейшем можно изменить стиль с помощью метода setDateTimeStyles(), как показано в следующем примере:

```
var date:Date = new Date(2009, 2, 27, 13, 1);
var dtf:DateTimeFormatter = new DateTimeFormatter("en-US",
    DateTimeStyle.LONG, DateTimeStyle.LONG);

var longDate:String = dtf.format(date);
trace(longDate); // March 27, 2009 1:01:00 PM

dtf.setDateTimeStyles(DateTimeStyle.SHORT, DateTimeStyle.SHORT);
var shortDate:String = dtf.format(date);
trace(shortDate); // 3/27/09 1:01 PM
```

## Локализация названий месяцев и дней

Во многих приложениях используются списки названий месяцев и дней недели в представлении календаря или раскрывающихся списках.

Локализованный список названий месяцев можно извлечь с помощью метода DateTimeFormatter.getMonthNames(). В зависимости от операционной системы могут быть доступны полные и сокращенные формы. Передайте значение DateTimeNameStyle.FULL для получения полных названий месяцев. Передайте значения DateTimeNameStyle.LONG_ABBREVIATION или DateTimeNameStyle.SHORT_ABBREVIATION для получения более коротких версий.

В некоторых языках название месяца изменяется (преобразуется в родительный падеж), если стоит рядом со значением дня в формате даты. Если планируется использовать только названия месяцев, следует передать значения DateTimeNameContext.STANDALONE в метод getMonthNames(). Однако чтобы использовать названия месяцев в форматированных датах, следует передать значение DateTimeNameContext.FORMAT.

```
var dtf:DateTimeFormatter = new DateTimeFormatter("fr-FR");
var months:Vector.<String> = dtf.getMonthNames(DateTimeNameStyle.FULL,
        DateTimeNameContext.STANDALONE);
trace(months[0]); // janvier
months = dtf.getMonthNames(DateTimeNameStyle.SHORT_ABBREVIATION,
         DateTimeNameContext.STANDALONE);
trace(months[0]); // janv.
```

Метод DateTimeFormatter.getWeekdayNames() предоставляет локализованный список названий дней недели. Метод getWeekdayNames() допускает те же параметры nameStyle и context, что и метод getMonthNames().

```
var dtf:DateTimeFormatter = new DateTimeFormatter("fr-FR");
var weekdays:Vector.<String> = dtf.getWeekdayNames(DateTimeNameStyle.FULL,
           DateTimeNameContext.STANDALONE);
trace(weekdays[0]); // dimanche
weekdays = dtf.getWeekdayNames(DateTimeNameStyle.LONG_ABBREVIATION,
           DateTimeNameContext.STANDALONE);
trace(weekdays[0]); // dim.
```

Помимо этого, метод getFirstWeekday() возвращает значение указателя дня, который традиционно обозначает начало недели в выбранной локали.

# Сортировка и сравнение строк

**Flash Player 10.1 и более поздних версий, Adobe AIR 2.0 и более поздних версий**

Упорядочение — это процесс расположения элементов в правильном порядке. Правила упорядочения значительно отличаются в зависимости от локали. Правила также отличаются в зависимости от того, выполняется сортировка элементов в списке или сопоставление аналогичных элементов, например в алгоритме поиска текста.

При сортировке небольшие различия, например строчные и прописные буквы или диактрические знаки, такие как ударения, часто играют важную роль. Например, буква ö (o с тремой), как правило, считается эквивалентной простой букве o во французском или английском языке. Однако она должна следовать за буквой z в шведском языке. Кроме того, во французском и некоторых других языках ударение на последний слог слова определяет порядок слова в списке.

При поиске часто необходимо проигнорировать различие в регистрах или тремах для повышения вероятности нахождения подходящих совпадений. Например, можно предположить, что поиск символов cote во французском документе должен вернуть совпадения cote, côte и coté.

## Использование класса Collator

Основными методами класса Collator являются compare(), используемый в большей части для сортировки, и equals(), используемый для сопоставления значений.

В следующем примере показано различное поведение методов compare() и equals().

```
var words:Array = new  Array("coté", "côte");

var sorter:Collator = new Collator("fr-FR", CollatorMode.SORTING);
words.sort(sorter.compare);
trace(words); // côte,coté

var matcher:Collator = new Collator("fr-FR", CollatorMode.MATCHING);
if (matcher.equals(words[0], words[1]))
{
    trace(words[0] + " = " + words[1]); // côte = coté
}
```

В примере сначала создается объект Collator в режиме SORTING для локали French-France. Затем выполняется сортировка двух слов, которые отличаются только диактрическими знаками. Этот пример показывает, что сравнение SORTING выполняется с учетом различий ударных и безударных символов.

Сортировка выполняется путем передачи ссылки на метод sort() объекта Collator в качестве параметра в метод Array.sort(). Это один из наиболее эффективных способов использования объекта Collator для управления порядком сортировки.

После этого в примере создается объект Collator в режиме MATCHING. Когда объект Collator сравнивает два слова, он считает их равнозначными. Это показывает, что при сравнении MATCHING ударные и безударные символы считаются равнозначными.

## Настройка поведения класса Collator

По умолчанию в классе Collator используются правила сравнения строк, полученные из операционной системы на основе локали и пользовательских системных настроек. Можно настроить поведение методов compare() и equals() путем явного задания различных свойств. В следующей таблице перечислены свойства и их влияние на результат сравнения:

| Свойство Collator | Эффект |
|---|---|
| numericComparison | Управляет, будут ли символы цифр считаться числами или текстом. |
| ignoreCase | Управляет, будут ли игнорироваться различия символов в верхнем и нижнем регистре. |
| ignoreCharacterWidth | Управляет, будут ли формы полной ширины и половинной ширины некоторых китайских и японских символов оцениваться как равные. |
| ignoreDiacritics | Определяет, считаются ли равными строки, в которых используются одинаковые базовые символы, но с различными ударениями или другими диактрическими знаками. |
| ignoreKanaType | Определяет, считаются ли равными строки, отличающиеся только по типу используемого символа каны. |
| ignoreSymbols | Управляет пропуском символов, таких как пробел, символы валюты, математические знаки и другие. |

Следующий код показывает, что при задании для свойства ignoreDiacritics значения true изменяется порядок сортировки французских слов:

```
var words:Array = new  Array("COTE", "coté", "côte", "Coté","cote");
var sorter:Collator = new Collator("fr-CA", CollatorMode.SORTING);
words.sort(sorter.compare);
trace(words); // cote,COTE,côte,coté,Coté

sorter.ignoreDiacritics = true;
words.sort(sorter.compare);
trace(words); // côte,coté,cote,Coté,COTE
```

# Преобразование регистра

**Flash Player 10.1 и более поздних версий, Adobe AIR 2.0 и более поздних версий**

Языки также отличаются правилами преобразования букв в прописные формы (маюскульные) и строчные формы (минускульные).

Например, в большинстве языков, в которых используется латинский алфавит, строчной формой прописной буквы I является i. Однако в некоторых языках (например, турецком и азербайджанском) традиционно используется буква без точки — ı. Поэтому в этих языках строчная буква ı без точки преобразуется в прописную букву I. В то время как строчная буква i преобразуется в прописную букву İ с точкой.

Класс StringTools предоставляет методы, использующие характерные для языка правила для выполнения подобных преобразований.

## Использование класса StringTools

Класс StringTools предоставляет два метода для выполнения преобразования регистра: toLowerCase() и toUpperCase(). Объект StringTools создается путем вызова конструктора с идентификатором локали. Класс StringTools извлекает правила преобразования регистра для этой локали (или резервной локали) из операционной системы. Дополнительная настройка алгоритма преобразования регистра невозможна.

В следующем примере методы toUpperCase() и toLowerCase() используются для преобразования фразы на немецком языке, включающей букву ß (двойная S).

```
var phrase:String = "Schloß Neuschwanstein";
var converter:StringTools = new StringTools("de-DE");

var upperPhrase:String = converter.toUpperCase(phrase);
trace(upperPhrase); // SCHLOSS NEUSCHWANSTEIN

var lowerPhrase:String = converter.toLowerCase(upperPhrase);
trace(lowerPhrase);// schloss neuschwanstein
```

Метод toUpperCase() преобразует строчную букву ß в прописные буквы SS. Такое преобразование выполняется только в одном направлении. Когда буквы SS преобразуются обратно в строчные, в результате получается ss, а не ß.

# Пример. Интернационализация приложения, передающего котировки ценных бумаг

**Flash Player 10.1 и более поздних версий, Adobe AIR 2.0 и более поздних версий**

Приложение Global Stock Ticker извлекает и отображает фиктивные данные о ценных бумагах на трех различных рынках ценных бумаг: США, Япония и Европа. Оно форматирует данные в соответствии с нормами, принятыми для различных локалей.

Этот пример демонстрирует следующие функции пакета flash.globalization:

- Форматирование чисел с учетом локали.
- Форматирование валют с учетом локали.
- Задание кода валюты по ISO и символов валюты.
- Форматирование дат с учетом локали.
- Получение и отображение соответствующих сокращенных названий месяцев.

Получить файлы приложения для этого примера можно на странице www.adobe.com/go/learn_programmingAS3samples_flash_ru. Файлы приложения Global Stock Ticker находятся в папке Samples/GlobalStockTicker. Приложение состоит из следующих файлов.

| File | Описание |
|---|---|
| GlobalStockTicker.mxml<br>или<br>GlobalStockTicker.fla | Пользовательский интерфейс приложения для Flex (MXML) или Flash (FLA). |
| styles.css | Стили пользовательского интерфейса приложения (только Flex). |
| com/example/programmingas3/stockticker/flex/FinGraph.mxml | Компонент MXML, который отображает диаграмму для смоделированных данных о ценных бумагах (только для Flex). |
| com/example/programmingas3/stockticker/flash/GlobalStockTicker.as | Класс документа, отвечающий за поведение интерфейса пользователя для приложения (только для Flash). |
| comp/example/programmingas3/stockticker/flash/RightAlignedColumn.as | Пользовательское средство визуализации ячеек для компонента Flash DataGrid (только для Flash). |
| com/example/programmingas3/stockticker/FinancialGraph.as | Класс ActionScript, вычерчивающий диаграмму для смоделированных данных о ценных бумагах. |
| com/example/programmingas3/stockticker/Localizer.as | Класс ActionScript, управляющий локалью и валютой и обрабатывающий локализованное форматирование чисел, денежных сумм и дат. |
| com/example/programmingas3/stockticker/StockDataModel.as | Класс ActionScript, хранящий выборочные данные для приложения Global Stock Ticker. |

## Описание пользовательского интерфейса и выборочных данных

Основными элементами пользовательского интерфейса приложения являются:

- раскрывающийся список для выбора локали;
- раскрывающийся список для выбора рынка;
- объект DataGrid, отображающий данные для шести компаний на каждом рынке;
- диаграмма, показывающая смоделированную хронологию данных для выбранной категории ценных бумаг компании.

В приложении все выборочные данные о локалях, рынках и ценных бумагах компании хранятся в классе StockDataModel. Реальное приложение будет извлекать данные с сервера и сохранять их в классе, например StockDataModel. В этом примере все данные жестко закодированы в классе StockDataModel.

***Примечание.*** *Данные, отображаемые на финансовой диаграмме, необязательно должны соответствовать данным, показанным в элементе управления DataGrid. Диаграмма перерисовывается в произвольной форме каждый раз при выборе другой компании. Этот пример предназначен только для наглядного представления.*

## Установка локали

После первоначальной настройки приложение вызывает метод Localizer.setLocale() для создания объекта форматирования для локали по умолчанию. Метод setLocale() также вызывается каждый раз, когда пользователь выбирает новое значение в раскрывающемся списке локалей.

```
public function setLocale(newLocale:String):void
{
    locale  = new LocaleID(newLocale);

    nf = new NumberFormatter(locale.name);
    traceError(nf.lastOperationStatus, "NumberFormatter", nf.actualLocaleIDName);

    cf = new CurrencyFormatter(locale.name);
    traceError(cf.lastOperationStatus, "CurrencyFormatter", cf.actualLocaleIDName);
    symbolIsSafe = cf.formattingWithCurrencySymbolIsSafe(currentCurrency);
    cf.setCurrency(currentCurrency, currentSymbol);
    cf.fractionalDigits = currentFraction;

    df = new DateTimeFormatter(locale.name, DateTimeStyle.LONG, DateTimeStyle.SHORT);
    traceError(df.lastOperationStatus, "DateTimeFormatter", df.actualLocaleIDName);
    monthNames = df.getMonthNames(DateTimeNameStyle.LONG_ABBREVIATION);
}

public function traceError(status:String, serviceName:String, localeID:String) :void
{
    if(status != LastOperationStatus.NO_ERROR)
    {
        if(status == LastOperationStatus.USING_FALLBACK_WARNING)
        {
            trace("Warning - Fallback locale ID used by "
                    + serviceName + ": " + localeID);
        }
        else if (status == LastOperationStatus.UNSUPPORTED_ERROR)
        {
            trace("Error in " + serviceName + ": " + status);
            //abort application
            throw(new Error("Fatal error", 0));
        }
        else
        {
            trace("Error in " + serviceName + ": " + status);
        }
    }
    else
    {
        trace(serviceName + " created for locale ID: " + localeID);
    }
}
```

Сначала метод setLocale() создает объект LocaleID. При необходимости этот объект позволит упростить получение сведений о фактической локали впоследствии.

Затем он создает новые объекты NumberFormatter, CurrencyFormatter и DateTimeFormatter для локали. После создания каждого объекта форматирования он вызывает метод traceError(). Этот метод отображает сообщение об ошибке или предупреждение на консоли, если возникает проблема с запрошенной локалью. (В реальном приложении необходимо не только отслеживать эти ошибки, но и реагировать на них).

После создания объекта CurrencyFormatter метод setLocale() задает код валюты по ISO, символ валюты и свойства fractionalDigits в средстве форматирования для ранее определенных значений. (Эти значения задаются каждый раз, когда пользователь выбирает новый рынок в раскрывающемся списке рынков).

После создания объекта DateTimeFormatter метод setLocale() также извлекает массив локализованных сокращенных названий месяцев.

## Форматирование данных

Отформатированные данные о ценных бумагах представляются в элементе управления DataGrid. Каждый столбец DataGrid вызывает функцию label, которая форматирует значение в столбце с использованием подходящего объекта форматирования.

В версии Flash, например, следующий код устанавливает столбцы DataGrid:

```
var col1:DataGridColumn = new DataGridColumn("ticker");
col1.headerText = "Company";
col1.sortOptions = Array.NUMERIC;
col1.width = 200;

var col2:DataGridColumn = new DataGridColumn("volume");
col2.headerText = "Volume";
col2.width = 120;
col2.cellRenderer = RightAlignedCell;
col2.labelFunction = displayVolume;

var col3:DataGridColumn = new DataGridColumn("price");
col3.headerText = "Price";
col3.width = 70;
col3.cellRenderer = RightAlignedCell;
col3.labelFunction = displayPrice;

var col4:DataGridColumn = new DataGridColumn("change");
col4.headerText = "Change";
col4.width = 120;
col4.cellRenderer = RightAlignedCell;
col4.labelFunction = displayPercent;
```

В версии Flex этого примера объект DataGrid объявляется в файле MXML. В ней также задаются такие же функции label для каждого столбца.

Свойства labelFunction обращаются к следующим функциям, которые вызывают методы форматирования класса Localizer:

```
private function displayVolume(item:Object):String
{
    return localizer.formatNumber(item.volume, 0);
}

private function displayPercent(item:Object):String
{
    return localizer.formatPercent(item.change )  ;
}

  private function displayPrice(item:Object):String
{
    return localizer.formatCurrency(item.price);
}
```

После этого методы Localizer настраивают и вызывают подходящие средства форматирования:

```
public function formatNumber(value:Number, fractionalDigits:int = 2):String
{
    nf.fractionalDigits = fractionalDigits;
    return nf.formatNumber(value);
}

public function formatPercent(value:Number, fractionalDigits:int = 2):String
{
    // HACK WARNING: The position of the percent sign, and whether a space belongs
    // between it and the number, are locale-sensitive decisions. For example,
    // in Turkish the positive format is %12 and the negative format is -%12.
    // Like most operating systems, flash.globalization classes do not currently
    // provide an API for percentage formatting.
    nf.fractionalDigits = fractionalDigits;
    return nf.formatNumber(value) + "%";
}

public function formatCurrency(value:Number):String
{
    return cf.format(value, symbolIsSafe);
}

public function formatDate(dateValue:Date):String
{
    return df.format(dateValue);
}
|
```

# Глава 57. Локализация приложений

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Локализация — это процесс добавления файлов для поддержки нескольких локалей. Локаль представляет собой сочетание языка и кода страны. Например, en_US означает английский язык (США), а fr_FR — французский (Франция). Для локализации приложений согласно этим локалям потребуется два набора файлов: один для локали en_US, а другой — для fr_FR.

У нескольких локалей может быть общий язык. Например, en_US и en_GB (Великобритания) — это две разных локали. Обе они используют английский, но код страны указывает на то, что это разные локали, поэтому они могут обращаться к разным файлам. Например, в локали en_US слово «цвет» может писаться как «color», а в en_GB — «colour». Кроме того, разной будет валюта: в первом случае доллары, во втором фунты, — а также формат даты и времени.

Можно создать набор файлов для языка без указания кода страны. Например, можно задать набор файлов en для английского и добавить для локали en_US дополнительные ресурсы, относящиеся только к английскому (США).

Локализация — это не просто перевод файлов приложения на другой язык. Она также охватывает и другие типы ресурсов: аудиофайлы, изображения и видео.

## Выбор локали

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Определить локаль, используемую в вашем содержимом, можно одним из следующих способов:

- Пакет flash.globalization — использование классов для локали, доступных в пакете flash.globalization, для получения локали по умолчанию на основе настроек операционной системы и пользовательских настроек. Это предпочтительный подход в приложениях, которые будут выполняться в средах выполнения Flash Player 10.1 и более поздних версий или AIR 2.0 и более поздних версий. Дополнительные сведения см. в разделе ««Определение локали» на странице 1015».
- Запрос пользователя. Можно запустить приложение на локали по умолчанию, а затем узнать у пользователя его предпочтения.
- `(AIR only) Capabilities.languages`. Свойство `Capabilities.languages` содержит список массива языков, перечисленных на предпочитаемых языках пользователя, согласно настройкам операционной системы. Строки, содержащие языковые теги (а также информацию о сценариях и регионах, если имеется), определены стандартом RFC4646 (http://www.ietf.org/rfc/rfc4646.txt). В строках в качестве разделителя используется дефис (например, `"en-US"` или `"ja-JP"`). Первая запись в возвращенном массиве будет иметь тот ID основного языка, который задан в свойстве language. Например, если `languages[0]` равно `"en-US"`, то свойство `language` будет `"en"`. Тем не менее, если свойство language равно `"xu"` (неизвестный язык), первый элемент в массиве `languages` будет другим.
- `Capabilities.language`. Свойство `Capabilities.language` содержит код языка пользовательского интерфейса операционной системы. Тем не менее значение этого свойства ограничивается 20 известными языками. В англоязычных системах это свойство возвращает только код языка, без кода страны. Поэтому лучше использовать первый элемент в массиве `Capabilities.languages`.

# Локализация содержимого Flex

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Adobe Flex предлагает инфраструктуру для локализации содержимого Flex. Эта инфраструктура включает классы Locale, ResourceBundle и ResourceManagerImpl, а также интерфейсы IResourceBundle и IResourceManagerImpl.

Библиотека для локализации Adobe Flex, содержащая классы утилит для сортировки локалей приложения, предоставляется на странице Google Code (http://code.google.com/p/as3localelib/).

### Дополнительные разделы справки

http://code.google.com/p/as3localelib/

# Локализация содержимого Flash

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Adobe Flash Professional содержит класс Locale в компонентах ActionScript 3.0. Класс Locale позволяет управлять тем, как SWF-файл отображает текст на разных языках. Панель Flash «Строки» позволяет использовать в динамических полях ID строк вместо литералов. Благодаря этому можно создать SWF-файл, отображающий текст, загруженный из XML-файла соответствующего языка. Сведения об использовании класса Locale см. в справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform.

# Локализация приложений AIR

**Adobe AIR 1.0 и более поздних версий**

В комплект AIR SDK включена инфраструктура локализации HTML (в файле AIRLocalizer.js). Эта инфраструктура включает API-интерфейсы, которые помогают в работе с несколькими локалями в приложениях на базе HTML. Библиотеку ActionScript для сортировки локалей можно найти на странице http://code.google.com/p/as3localelib/.

### Дополнительные разделы справки

http://code.google.com/p/as3localelib/

# Локализация даты, времени и валюты

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В разных локалях отображения даты, времени и валюты могут значительно отличаться. Например, в США даты принято записывать в формате «месяц/день/год», в то время как в Европе обычно пишут «день/месяц/год».

Можно написать код для форматирования даты, времени и валюты. Например, в коде ниже объект Date преобразуется между форматами «месяц/день/год» и «день/месяц/год». Если переменная `locale` (представляющая локаль) равна `"en_US"`, функция возвращает дату в формате «месяц/день/год». В этом примере для всех остальных локалей объект Date преобразуется в формат «день/месяц/год»:

```
function convertDate(date)
{
    if (locale == "en_US")
    {
        return (date.getMonth() + 1) + "/" + date.getDate() + "/" + date.getFullYear();
    }
    else
    {
        return date.getDate() + "/" + (date.getMonth() + 1) + "/" + date.getFullYear();
    }
}
```

**ADOBE FLEX**

Инфраструктура Flex содержит элементы управления для форматирования дат, времени и валюты. К ним относятся элементы DateFormatter и CurrencyFormatter.

- mx:DateFormatter
- mx:CurrencyFormatter

# Глава 58. О среде HTML

**Adobe AIR 1.0 и более поздних версий**

Adobe® AIR® использует движок WebKit *(www.webkit.org)*, который также используется веб-браузером Safari, для синтаксического анализа, макетирования, визуализации содержимого HTML и сценариев JavaScript. Использовать API-интерфейс AIR в HTML-содержимом необязательно. Код может быть написан непосредственно в объекте HTMLLoader или окне HTML на языке HTML или JavaScript. Большинство существующих HTML-страниц и приложений будет работать с некоторыми изменениями (здесь мы исходим из того, что используются функции HTML, CSS, DOM и JavaScript, совместимые с WebKit).

**Важно:** новые версии среды Adobe AIR могут включать в себя обновленные версии движка WebKit. Обновление WebKit в новой версии AIR *может* привести к неожиданным изменениям в развернутой программе AIR. Эти изменения способны повлиять на поведение и внешний вид содержимого HTML в программе. Например, усовершенствование или корректировка визуализации движка WebKit может привести к изменению макета элементов в интерфейсе пользователя программы. По этой причине настоятельно рекомендуется предусмотреть в вашей программе механизм обновления. Если вам необходимо обновить свою программу вследствие изменения используемой версии движка WebKit, включенного в AIR, в механизме обновления AIR предусмотрено предложение пользователю установить новую версию вашей программы.

В следующей таблице указана версия веб-браузера Safari, которая использует версию WebKit, эквивалентную версии, используемой в AIR:

| Версия AIR | Версия Safari |
|---|---|
| 1.0 | 2.04 |
| 1.1 | 3.04 |
| 1.5 | 4.0 Beta |
| 2.0 | 4.03 |
| 2.5 | 4.03 |
| 2.6 | 4.03 |
| 2.7 | 4.03 |
| 3 | 5.0.3 |

Можно всегда определить установленную версию WebKit путем проверки строки агента пользователя по умолчанию, возвращаемой объектом HTMLLoader:

```
var htmlLoader:HTMLLoader = new HTMLLoader();
trace( htmlLoader.userAgent );
```

Помните, что версия WebKit, используемая в среде AIR, не совпадает с версией с открытым кодом. Некоторые функции не поддерживаются в среде AIR, и версия AIR может включать исправления безопасности и ошибок, которые еще не доступны в соответствующей версии WebKit. См. раздел ««Функции WebKit не поддерживаются в AIR» на странице 1049».

Приложения AIR выполняются непосредственно на рабочем столе и имеют полный доступ к файловой системе, поэтому их модель безопасности HTML-содержимого более строгая, чем модель типичных веб-обозревателей. В AIR в *изолированную программную среду приложения* помещается только содержимое, загруженное непосредственно из установочного каталога. Изолированная программная среда приложения имеет самый высокий приоритет и открывает доступ к API-интерфейсам AIR. AIR помещает остальное содержимое в изолированные программные среды на основании происхождения этого содержимого. Файлы,

загруженные из файловой системы, помещаются в локальную изолированную программную среду. Файлы, загруженные из сети по протоколу http или https, помещаются в изолированную программную среду в соответствии со своим доменом или удаленным сервером. Содержимое в этих изолированных программных средах вне приложения не имеет доступа ни к каким интерфейсам AIR и работает, как в обычном веб-обозревателе.

HTML-содержимое в среде AIR не отображает SWF- или PDF-содержимое, если применяются параметры альфа-канала, масштабирования или прозрачности. Дополнительные сведения см. в разделах ««Примечания по загрузке содержимого SWF или PDF на страницу HTML» на странице 1081» и ««Прозрачность окон» на странице 959».

### Дополнительные разделы справки

Справочник по DOM в Webkit

Справочник по HTML в Safari

Справочник по CSS в Safari

www.webkit.org

# Обзор среды HTML

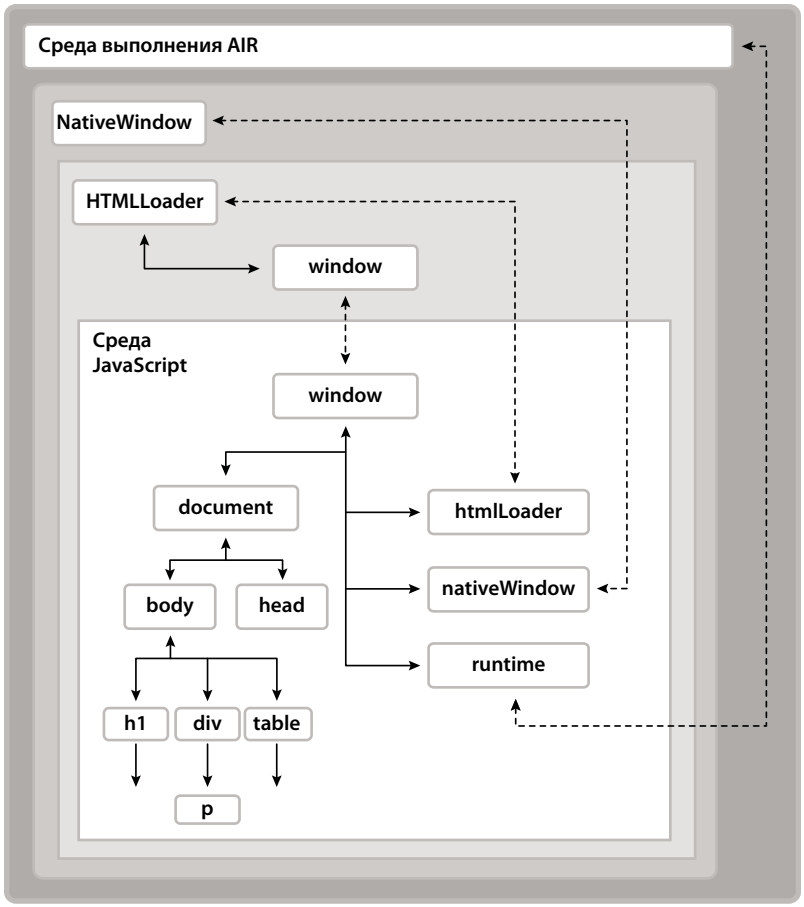
**Adobe AIR 1.0 и более поздних версий**

Adobe AIR обеспечивает полноценную среду JavaScript наподобие среды обозревателя, включающую средства визуализации HTML, объектную модель документа и интерпретатор JavaScript. Среда JavaScript представлена классом AIR HTMLLoader. В окнах HTML в объект HTMLLoader включено все HTML-содержимое, а он, в свою очередь, содержится в объекте NativeWindow. В SWF-содержимом класс HTMLLoader, являющийся расширением класса Sprite, может быть добавлен в список отображения рабочей области, как любой другой экранный объект. Свойства класса Adobe® ActionScript® 3.0 описываются в разделе ««Написание сценария для HTML-контейнера AIR» на странице 1079 и *справочнике ActionScript 3.0 для платформы Adobe Flash Platform*. В инфраструктуре Flex класс AIR HTMLLoader заключен в компонент mx:HTML. Компонент mx:HTML является расширением класса UIComponent, чтобы он мог использоваться с другими контейнерами Flex. В остальном среда JavaScript внутри компонента mx:HTML точно такая же.

## О среде разработки сценариев JavaScript и ее связи с хостом AIR

**Adobe AIR 1.0 и более поздних версий**

На схеме ниже показано, как связаны среда JavaScript и рабочая среда AIR. Хотя показано только одно исходное окно, в приложении AIR может содержаться несколько. (А в одном окне может содержаться несколько объектов HTMLLoader.)




*В среде JavaScript есть собственные объекты Document и Window. Код JavaScript может общаться со средой выполнения AIR с помощью свойств runtime, nativeWindow и htmlLoader. Код ActionScript может взаимодействовать со средой JavaScript с помощью свойства window объекта HTMLLoader, являющегося ссылкой на объект Window языка JavaScript. Кроме того, и объекты ActionScript, и объекты JavaScript могут прослушивать события, отправляемые объектами AIR и JavaScript.*

Свойство `runtime` предоставляет доступ к классам API-интерфейса AIR для создания новых объектов AIR и членов классов доступа (также называемых статическими). Для доступа к API-интерфейсу AIR необходимо добавить в свойство `runtime` имя класса (вместе с именем пакета). Например, для создания объекта File будет использоваться следующая инструкция:

```
var file = new window.runtime.filesystem.File();
```

***Примечание.*** *AIR SDK содержит файл JavaScript `AIRAliases.js`, в котором задаются удобные псевдонимы для большинства классов AIR. При импорте этого файла вместо window.runtime.package.Class можно использовать более короткую форму air.Class. Например, можно создать объект File с помощью метода `new air.File()`.*

Объект NativeWindow предоставляет свойства для управления окном рабочего стола. С HTML-страницы с помощью свойства `window.nativeWindow` можно получать доступ к объекту-контейнеру NativeWindow.

Объект HTMLLoader предлагает свойства, методы и события для управления загрузкой и визуализацией содержимого. С HTML-страницы с помощью свойства `window.htmlLoader` можно получать доступ к родительскому объекту HTMLLoader.

***Важная информация.*** *Свойства* `htmlLoader`, `nativeWindow` *и* `runtime` *могут быть только у страниц, установленных как часть приложения, и только когда они загружаются в качестве документов верхнего уровня. Эти свойства не добавляются, когда документ загружается в обычный или встроенный фрейм. (Дочерний документ также имеет доступ к этим свойствам в родительском документе, пока оба находятся в одной и той же изолированной программной среде.) Например, документ, загруженный во фрейм, может получать доступ к свойству* `runtime` *своего родителя через* `parent.runtime`*).*

## О безопасности

**Adobe AIR 1.0 и более поздних версий**

AIR исполняет весь код в изолированной программной среде на основании домена, с которого он получен. Содержимое приложения, ограниченное до загружаемого из установочного каталога приложения, помещается в изолированную программную среду *приложения*. Доступ к рабочей среде и API-интерфейсам AIR возможен только для кода HTML и JavaScript, исполняемого в рамках изолированной программной среды. В то же время большая часть динамической проверки и исполнения кода JavaScript в изолированной программной среде блокируется после возвращения всех обработчиков события `load` на странице.

Страницу приложения можно преобразовать в изолированную программную среду вне приложения путем загрузки страницы во фрейм или встроенный фрейм и задания для фрейма атрибутов AIR `sandboxRoot` и `documentRoot`. Если атрибуту `sandboxRoot` в качестве значения задать реальный удаленный домен, то содержимое в изолированной программной среде сможет выполнять междоменный сценарий с содержимым этого домена. Такое преобразование может пригодиться при загрузке и создании сценариев для удаленного содержимого, например в *мэшап*-приложениях.

Есть и другой способ перекрестного выполнения сценариев, и единственный способ дать содержимому вне приложения доступ к API-интерфейсу приложения AIR — это создать *мост изолированной программной среды*. Мост *от родительского к дочернему элементу* позволяет содержимому фрейма, встроенного фрейма или окна получать доступ к методам и свойствам, заданным в изолированной программной среде приложения. И наоборот, мост *от дочернего элемента к родительскому* позволяет приложению получать доступ к методам и свойствам, заданным в изолированной программной среде дочернего элемента. Мосты изолированных программных сред устанавливаются путем задания свойств `parentSandboxBridge` и `childSandboxBridge` объекта window. Дополнительные сведения см. в разделах ««Безопасность HTML в Adobe AIR» на странице 1165» и ««Элементы frame и iframe в HTML» на странице 1045».

## О подключаемых модулях и вложенных объектах

**Adobe AIR 1.0 и более поздних версий**

Среда AIR поддерживает подключаемый модуль Adobe® Acrobat®. Для отображения PDF-содержимого необходимо установить Acrobat или Adobe® Reader® версии 8.1 (или более поздней). Объект HTMLLoader предлагает свойство для проверки возможности отображения PDF-содержимого. Содержимое SWF-файла также может отображаться в среде HTML, но эта возможность является встроенной и не предполагает использования внешних подключаемых модулей.

Остальные подключаемые модули WebKit не поддерживаются средой AIR.

### Дополнительные разделы справки

«Безопасность HTML в Adobe AIR» на странице 1165

«Изолированные программные среды в HTML» на странице 1037

«Элементы frame и iframe в HTML» на странице 1045

«Объект Window в JavaScript» на странице 1043

«Объект XMLHttpRequest» на странице 1039

«Добавление PDF-содержимого в приложения AIR» на странице 592

# AIR и WebKit

**Adobe AIR 1.0 и более поздних версий**

Adobe AIR использует движок WebKit с открытым исходным кодом, который также применяется в веб-браузере Safari. AIR добавляет ряд расширений, разрешающих доступ к классам и объектам среды выполнения и повышающих безопасность. Кроме того, сам WebKit добавляет функции, не включенные в стандарты W3C для HTML, CSS и JavaScript.

Здесь рассматриваются только дополнения AIR и наиболее примечательные расширения WebKit. Дополнительную информацию о нестандартных функциях HTML, CSS и JavaScript см. по адресам www.webkit.org и developer.apple.com. Стандарты доступны на веб-сайте W3C. На сайте Mozilla также опубликованы полезные сведения о HTML, CSS и DOM (разумеется, механизмы WebKit и Mozilla различаются).

## JavaScript в AIR

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

AIR несколько меняет типичное поведение основных объектов JavaScript. Многие из этих изменений вносятся для того, чтобы повысить безопасность приложений AIR. В то же время эти отличия в поведении означают, что некоторые основные кодовые комбинации JavaScript и существующие веб-приложения, использующие их, порой могут работать в среде AIR непредсказуемо. Сведения об исправлении таких ошибок см. в разделе ««Избежание ошибок безопасности в JavaScript» на странице 1056».

### Изолированные программные среды в HTML

**Adobe AIR 1.0 и более поздних версий**

AIR помещает содержимое в изолированные программные среды согласно его происхождению. Правила изолированной программной среды соответствуют политике отбора по критерию происхождения, реализуемой большинством обозревателей, а также правилам изолированных программных сред проигрывателя Adobe Flash. Кроме того, AIR предлагает новый тип — изолированную программную среду *приложения* для хранения и защиты содержимого приложения. Дополнительные сведения о типах изолированных программных сред, встречающихся при разработке приложений AIR, см. в разделе ««Изолированные программные среды» на странице 1123».

Доступ к рабочей среде и API-интерфейсам AIR возможен только для кода HTML и JavaScript, исполняемого в рамках изолированной программной среды приложения. В то же время динамическая проверка и исполнение кода JavaScript в самых разных его формах подвергаются серьезным ограничениям внутри изолированной программной среды приложения из соображений безопасности. Эти ограничения применяются независимо от того, загружает ли приложение данные с сервера. (Даже файловое содержимое, вставленные строки и данные, введенные непосредственно пользователем, могут быть небезопасны.)

Изолированная программная среда, в которую помещается содержимое страницы, определяется его происхождением. В изолированную программную среду приложения помещаются только данные, загруженные непосредственно из установочного каталога приложения (установочный каталог приложения отражен в URL-схеме `app:`). Содержимое, загружаемое из файловой системы, помещается в *локальную изолированную программную среду файловой системы* или в *локальную доверенную изолированную программную среду*, которые разрешают взаимодействие с содержимым в рамках локальной файловой системы, но не с удаленным содержимым. Содержимое, загружаемое из сети, помещается в удаленную изолированную программную среду в соответствии с доменом, с которого оно получено.

Чтобы страница приложения могла свободно обращаться к содержимому в удаленной изолированной программной среде, ее можно сопоставить с тем же доменом, что и удаленное содержимое. Например, если в приложении отображается карта из Интернета, то страница приложения, загружающая и отображающая это содержимое, может сопоставляться с этим доменом. Атрибуты сопоставления страниц с удаленными изолированными программными средами и доменами — новые, они добавляются в элементы frame и iframe в HTML.

Чтобы содержимое изолированной программной среды вне приложения могло безопасно использовать функции AIR, можно установить мост родительской изолированной программной среды. Чтобы содержимое приложения могло безопасно вызывать методы и свойства доступа к другим изолированным программным средам, можно установить мост дочерней изолированной программной среды. Здесь безопасность обеспечивается тем, что удаленное содержимое не может случайным образом получать ссылки на объекты, свойства или методы, не предоставляемые явно. По мосту могут передаваться только данные простого типа, функции и анонимные объекты. Тем не менее следует избегать предоставления потенциально опасных функций. Например, если предоставляется интерфейс, позволяющий удаленному содержимому читать и записывать любые файлы в системе пользователя, то такое удаленное содержимое может серьезно повредить систему.

## Функция eval() в JavaScript

**Adobe AIR 1.0 и более поздних версий**

Использование функции `eval()` возможно в изолированной программной среде только после загрузки страницы. Некоторые варианты ее использования разрешаются, чтобы данные в формате JSON можно было безопасно разбирать, но любая проверка, итогом которой становится исполнимая инструкция, приведет к ошибке. В разделе ««Ограничения по использованию кода в содержимом из различных изолированных программных сред» на странице 1168» описываются разрешенные способы использования функции `eval()`.

## Конструкторы функций

**Adobe AIR 1.0 и более поздних версий**

В изолированной программной среде приложения конструкторы функций могут запускаться до окончания загрузки страницы. После окончания работы всех обработчиков события `load` на странице новые функции создавать нельзя.

## Загрузка внешних сценариев

**Adobe AIR 1.0 и более поздних версий**

HTML-страницы в изолированной программной среде приложения не могут использовать тег `script` для загрузки файлов JavaScript извне каталога приложения. Чтобы страница приложения могла загружать сценарий извне каталога приложения, она должна быть сопоставлена с изолированной программной средой приложения.

## Объект XMLHttpRequest

**Adobe AIR 1.0 и более поздних версий**

AIR предоставляет объект XMLHttpRequest (XHR), который приложения могут использовать для запроса данных. В примере ниже показан простой запрос данных.

```
xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", "http:/www.example.com/file.data", true);
xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4) {
        //do something with data...
    }
}
xmlhttp.send(null);
```

В отличие от обозревателей, среда AIR разрешает содержимому, работающему в изолированной программной среде, запрашивать данные с любого домена. Результат исполнения XHR, содержащий JSON-строку, может быть преобразован в данные объектов, если только он не содержит исполняемый код. Если в результате XHR получаются исполняемые инструкции, генерируется ошибка, а завершить проверку не удается.

Для предотвращения случайного внедрения кода из удаленных источников синхронные XHR возвращают пустые результаты, если они выполнены до окончания загрузки страницы. Асинхронные XHR всегда возвращают результат после окончания загрузки страницы.

По умолчанию AIR блокирует междоменные запросы XMLHttpRequest в изолированных программных средах вне приложения. Родительское окно в изолированной программной среде приложения может разрешать междоменные запросы в дочернем фрейме с содержимым из изолированной программной среды вне приложения. Для этого атрибут `allowCrossDomainXHR` в элементе frame или iframe, добавленный средой AIR, принимает значение `true`:

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    allowCrossDomainXHR="true"
</iframe>
```

***Примечание.*** *По возможности для загрузки данных можно также использовать класс AIR URLStream.*

При отправке запроса XMLHttpRequest удаленному серверу из фрейма или встроенного фрейма с содержимым, сопоставленным с удаленной изолированной программной средой, проверьте, чтобы сопоставляемый URL-адрес не маскировал адрес сервера в XHR. Например, рассмотрите следующее определение встроенного фрейма iframe, сопоставляющее содержимое приложения с удаленной изолированной программной средой для домена example.com:

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    documentRoot="app:/sandbox/"
    sandboxRoot="http://www.example.com/"
    allowCrossDomainXHR="true"
</iframe>
```

Так как атрибут `sandboxRoot` повторно сопоставляет корневой URL-адрес www.example.com, все запросы загружаются из каталога приложения, а не с удаленного сервера. Запросы сопоставляются повторно независимо от того, как они появились: в результате перехода по странице или из запроса XMLHttpRequest.

Чтобы нечаянно не заблокировать запросы данных с удаленного сервера, сопоставьте `sandboxRoot` с подкаталогом удаленного URL-адреса, а не с корневым каталогом. Необязательно, чтобы каталог существовал. Например, чтобы запросы на www.example.com загружались с удаленного сервера, а не из каталога приложения, измените предыдущий встроенный фрейм (iframe), как показано ниже.

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    documentRoot="app:/sandbox/"
    sandboxRoot="http://www.example.com/air/"
    allowCrossDomainXHR="true"
</iframe>
```

В этом случае локально будет загружено только содержимое в подкаталоге `air`.

Дополнительные сведения о сопоставлении изолированных программных сред см. в разделах ««Элементы frame и iframe в HTML» на странице 1045» и ««Безопасность HTML в Adobe AIR» на странице 1165».

## Файлы cookies

**Adobe AIR 1.0 и более поздних версий**

В приложениях AIR только содержимое удаленных изолированных программных сред (загруженное по протоколам http: и https:) может использовать файлы cookies (свойство `document.cookie`). В изолированной программной среде приложения имеются другие средства хранения постоянных данных, таких как классы EncryptedLocalStore, SharedObject и FileStream.

## Объект Clipboard

**Adobe AIR 1.0 и более поздних версий**

API-интерфейс Clipboard в WebKit управляется событиями `copy`, `cut` и `paste`. Объект события, передаваемый в ответ на эти события, обеспечивает доступ к буферу обмена через свойство `clipboardData`. Используйте для чтения и записи данных в буфер обмена методы объекта `clipboardData`.

| Метод | Описание |
|---|---|
| clearData(mimeType) | Очищает буфер обмена. Установите для параметра `mimeType` значение MIME-типа стираемых данных. |
| getData(mimeType) | Получает данные из буфера обмена. Этот метод может быть вызван только в обработчике события `paste`. Установите для параметра `mimeType` значение MIME-типа возвращаемых данных. |
| setData(mimeType, данные) | Копирует данные в буфер обмена. Установите для параметра `mimeType` значение MIME-типа данных. |

Код JavaScript вне изолированной программной среды приложения получает доступ к буферу обмена только через эти события. Тем не менее содержимое изолированной программной среды приложения может обращаться к системному буферу напрямую с помощью класса Clipboard в AIR. Например, с помощью показанной ниже инструкции можно получить данные о формате текста в буфере:

```
var clipping = air.Clipboard.generalClipboard.getData("text/plain",
                                air.ClipboardTransferMode.ORIGINAL_ONLY);
```

Ниже перечислены поддерживаемые MIME-типы данных:

| Тип MIME | Значение |
|---|---|
| Текст | «text/plain»; |
| HTML | «text/html»; |
| URL-адрес | «text/uri-list»; |
| Растровое изображение | «image/x-vnd.adobe.air.bitmap»; |
| Список файлов | «application/x-vnd.adobe.air.file-list». |

***Важная информация.*** *Доступ к файловым данным в буфере обмена может получать только содержимое в изолированной программной среде. Если к файловому объекту в буфере обмена пытается обратиться содержимое извне приложения, то генерируется ошибка безопасности.*

Дополнительные сведения об использовании буфера обмена см. в разделе ««Копирование и вставка» на странице 641» и статье «Использование Pasteboard в JavaScript (Центр разработки Apple)».

## Перетаскивание

**Adobe AIR 1.0 и более поздних версий**

Жесты перетаскивания в HTML-документ и обратно вызывают следующие события DOM: `dragstart`, `drag`, `dragend`, `dragenter`, `dragover`, `dragleave` и `drop`. Объект события, передаваемый в ответ на эти события, обеспечивает доступ к перетаскиваемым данным через свойство `dataTransfer`. Свойство `dataTransfer` указывает на объект, предоставляющий те же методы, что и объект `clipboardData`, связанный с событием clipboard. Например, следующая функция получает данные о формате текста из события `drop`:

```
function onDrop(dragEvent){
    return dragEvent.dataTransfer.getData("text/plain",
            air.ClipboardTransferMode.ORIGINAL_ONLY);
}
```

Ниже перечислены значимые члены объекта `dataTransfer`.

| Член | Описание |
|---|---|
| clearData(mimeType) | Стирает данные. Установите для параметра `mimeType` значение MIME-типа стираемого представления данных. |
| getData(mimeType) | Получает перетаскиваемые данные. Этот метод может быть вызван только в обработчике события `drop`. Установите для параметра `mimeType` значение MIME-типа получаемых данных. |
| setData(mimeType, данные) | Задайте перетаскиваемые данные. Установите для параметра `mimeType` значение MIME-типа данных. |
| types | Массив строк, содержащий MIME-типы всех представлений данных, доступных в настоящий момент в объекте `dataTransfer`. |
| effectsAllowed | Указывает, можно ли копировать, перемещать или создавать ссылки на перетаскиваемые данные. Задайте свойство `effectsAllowed` в обработчике события `dragstart`. |
| dropEffect | Указывает, какие из разрешенных эффектов отпускания разрешены целевым элементом перетаскивания. Задайте свойство `dropEffect` в обработчике события `dragEnter`. Во время перетаскивания курсор изменяет вид, чтобы показать, что произойдет, когда пользователь отпустит кнопку мыши. Если свойство `dropEffect` не указано, выбирается свойство `effectsAllowed`. Эффект copy имеет приоритет перед move, а тот, в свою очередь, имеет приоритет перед link. Пользователь может изменять приоритет по умолчанию с помощью клавиатуры. |

Дополнительные сведения о добавлении функции перетаскивания в приложение AIR см. в разделе ««Перетаскивание в AIR» на странице 654» и в статье «Использование перетаскивания в JavaScript (Центр разработки Apple)».

## Свойства innerHTML и outerHTML

**Adobe AIR 1.0 и более поздних версий**

В целях безопасности AIR накладывает ограничения на использование свойств `innerHTML` и `outerHTML` содержимым в изолированной программной среде. Перед событием load и во время исполнения любого обработчика события load свойства `innerHTML` и `outerHTML` могут использоваться без ограничений. Тем не менее после загрузки страницы свойства `innerHTML` и `outerHTML` могут использоваться только для добавления в документ статического содержимого. Все инструкции в строке, назначенные свойствам `innerHTML` и `outerHTML`, порождающие исполняемый код, игнорируются. Например, если включить атрибут обратного вызова события в определение элемента, прослушиватель события не будет добавлен. Точно так же вложенные теги `<script>` не проверяются. Дополнительные сведения см. в разделе ««Безопасность HTML в Adobe AIR» на странице 1165».

## Методы Document.write() и Document.writeln()

**Adobe AIR 1.0 и более поздних версий**

В изолированной программной среде методы `write()` и `writeln()` могут использоваться без ограничений, пока для страницы не будет выполнено событие `load`. Тем не менее когда страница загружена, вызов этих методов не очистит страницу и не создаст новую. В изолированной программной среде вне приложения, характерной для большинства веб-обозревателей, вызов метода `document.write()` или `writeln()` после завершения загрузки страницы очищает текущую страницу и открывает новую, пустую.

## Свойство Document.designMode

**Adobe AIR 1.0 и более поздних версий**

Чтобы все элементы документа стали редактируемыми, установите для свойства `document.designMode` значение `on`. Встроенный редактор позволяет изменять, копировать, вставлять и перетаскивать текст. Значение `on` свойства `designMode` идентично значению `true` свойства `contentEditable` элемента `body`. Свойство `contentEditable` может использоваться в большинстве HTML-элементов для определения редактируемых областей документа. Дополнительные сведения см. в разделе ««Атрибут contentEditable в HTML» на странице 1048».

## События unload (для объектов body и frameset)

**Adobe AIR 1.0 и более поздних версий**

В теге верхнего уровня `frameset` или `body` окна (в т. ч. и главного окна приложения) нельзя использовать событие `unload` для ответа на закрытие окна (или приложения). Вместо этого используйте событие `exiting` объекта NativeApplication (для определения момента закрытия приложения). Также может использоваться событие `closing` объекта NativeWindow (для определения закрытия окна). Например, в следующем коде JavaScript, когда пользователь закрывает приложение, выводится сообщение (`Goodbye.` ):

```
var app = air.NativeApplication.nativeApplication;
app.addEventListener(air.Event.EXITING, closeHandler);
function closeHandler(event)
{
    alert("Goodbye.");
}
```

Тем не менее сценарии *могут* отвечать на событие `unload`, вызванное в результате навигации по фрейму, встроенному фрейму или содержимому окна верхнего уровня.

***Примечание.*** *Возможно, этих ограничений не будет в следующей версии Adobe AIR.*

## Объект Window в JavaScript

**Adobe AIR 1.0 и более поздних версий**

Объект Window остается глобальным объектом в контексте выполнения JavaScript. В изолированной программной среде приложения AIR добавляет новые свойства в объект JavaScript Window для обеспечения доступа к встроенным классам AIR, а также для важных домашних объектов. Кроме того, некоторые методы и свойства ведут себя по-разному в зависимости от того, находятся ли они внутри изолированной программной среды.

**свойство Window.runtime** Свойство `runtime` позволяет создавать экземпляры и использовать встроенные классы среды выполнения в изолированной программной среде приложения. Эти классы включают API-интерфейсы AIR и проигрывателя Flash Player (но не инфраструктуру Flex). Например, следующая инструкция создает объект файла AIR:

```
var preferencesFile = new window.runtime.flash.filesystem.File();
```

Файл `AIRAliases.js`, включенный в комплект AIR SDK, содержит определения псевдонимов, позволяющих сокращать такие ссылки. Например, при импортировании на страницу файла `AIRAliases.js` объект File может быть создан с помощью следующей инструкции:

```
var preferencesFile = new air.File();
```

Свойство `window.runtime` определяется только для содержимого внутри изолированной программной среды и только для родительского документа страницы с фреймами, в т. ч. встроенными.

См. «Использование файла AIRAliases.js» на странице 1062.

**Свойство Window.nativeWindow** Свойство `nativeWindow` содержит ссылку на основной объект исходного окна. С помощью этого свойства можно программировать функции и свойства окон: положение, размер, видимость экрана и др., — а также обрабатывать события окон: закрытие, изменение размера и перемещение. Например, следующая инструкция закрывает окно:

```
window.nativeWindow.close();
```

***Примечание.*** *Функции управления окнами объекта NativeWindow пересекаются с функциями объекта Window в JavaScript. В этих случаях можно выбрать тот метод, который вам наиболее удобен.*

Свойство `window.nativeWindow` определяется только для содержимого внутри изолированной программной среды и только для родительского документа страницы с фреймами, в т. ч. встроенными.

**Свойство Window.htmlLoader** Свойство `htmlLoader` содержит ссылку на объект AIR HTMLLoader с HTML-содержимым. С помощью этого свойства можно программировать поведение и внешний вид HTML-документа. Например, с помощью свойства `htmlLoader.paintsDefaultBackground` можно определить, будет ли фон белым (по умолчанию):

```
window.htmlLoader.paintsDefaultBackground = false;
```

***Примечание.*** *Сам объект HTMLLoader содержит свойство* `window`*, ссылающееся на объект JavaScript Window своего HTML-содержимого. Это свойство позволяет получать доступ к среде JavaScript с помощью ссылки на содержащий его HTMLLoader.*

Свойство `window.htmlLoader` определяется только для содержимого внутри изолированной программной среды и только для родительского документа страницы с фреймами, в т. ч. встроенными.

**Свойства Window.parentSandboxBridge и Window.childSandboxBridge** Свойства `parentSandboxBridge` и `childSandboxBridge` определяют интерфейс взаимодействия родительского и дочернего окон. Дополнительные сведения см. в разделе ««Перекрестное выполнение сценариев в содержимом из различных изолированных программных сред безопасности» на странице 1074».

**Функции Window.setTimeout() и Window.setInterval()** Среда AIR накладывает ограничения безопасности на использование функций `setTimeout()` и `setInterval()` в изолированной программной среде. Код, исполняемый в строке при вызове функций `setTimeout()` и `setInterval()`, определить нельзя. Следует использовать обращение к функции. Дополнительную информацию см. в описании функций «Функции setTimeout() и setInterval()» на странице 1059.

**Функция Window.open()** Когда метод `open()` вызывается кодом, исполняемым в изолированной программной среде вне приложения, он открывает окно, только если он вызван в результате действий пользователя (например, щелчка мыши или нажатия клавиши). Кроме того, перед заголовком окна стоит заголовок приложения (это помогает отличать окна, открытые удаленным содержимым, от окон, открытых приложением). Дополнительные сведения см. в разделе ««Ограничения по вызову метода JavaScript window.open()» на странице 1171».

## Объект air.NativeApplication

**Adobe AIR 1.0 и более поздних версий**

Объект NativeApplication содержит информацию о состоянии приложения, отправляет некоторые важные события (уровня приложения) и содержит ряд функций по управлению поведением приложения. Единственный экземпляр объекта NativeApplication создается автоматически, и доступ к нему осуществляется с помощью свойства `NativeApplication.nativeApplication`.

Доступ к объекту из кода JavaScript может быть получен следующим образом:

```
var app = window.runtime.flash.desktop.NativeApplication.nativeApplication;
```

Или, если импортирован сценарий `AIRAliases.js`, можно воспользоваться сокращенным вариантом:

```
var app = air.NativeApplication.nativeApplication;
```

Доступ к объекту NativeApplication может быть получен только в пределах изолированной программной среды приложения. Дополнительные сведения об объекте NativeApplication см. в разделе ««Работа с информацией среды выполнения AIR и операционной системы» на странице 952».

## Схема URL-адресов в JavaScript

**Adobe AIR 1.0 и более поздних версий**

Исполнение кода, заданного схемой URL-адресов JavaScript (например, `href="javascript:alert('Test')"`), блокируется в изолированной программной среде приложения. Ошибка при этом не генерируется.

# HTML в AIR

**Adobe AIR 1.0 и более поздних версий**

AIR и WebKit задают несколько нестандартных элементов и атрибутов HTML, включая:

«Элементы frame и iframe в HTML» на странице 1045

«Обработчики событий элементов HTML» на странице 1047

## Элементы frame и iframe в HTML

**Adobe AIR 1.0 и более поздних версий**

AIR добавляет новые атрибуты в элементы frame и iframe содержимого в изолированной программной среде приложения.

**Атрибут sandboxRoot** Атрибут `sandboxRoot` задает альтернативный домен происхождения файла в атрибуте фрейма `src` (вне приложения). Файл загружается в изолированную программную среду вне приложения, соответствующую определенному домену. Содержимое файла и содержимое, загруженное с заданного домена, поддерживают перекрестное выполнение сценариев.

***Важная информация.*** *Если в качестве значения `sandboxRoot` указан основной URL-адрес домена, все запросы содержимого с этого домена загружаются из каталога приложения, а не с удаленного сервера (вне зависимости от того, как формируется запрос: в результате навигации пользователя, интерфейсом XMLHttpRequest или иным способом загрузки содержимого).*

**атрибут documentRoot** Атрибут `documentRoot` задает локальный каталог, из которого загружаются URL-адреса, ведущие к файлам в расположении, указанном в `sandboxRoot`.

При обработке URL-адресов в атрибуте фрейма `src` или в содержимом, загружаемом во фрейм, часть URL, совпадающая со значением `sandboxRoot`, замещается значением `documentRoot`. Следовательно, в следующем теге фрейма:

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"/>
```

`child.html` загружается из подкаталога `sandbox` установочной папки приложения. При обработке относительных URL-адресов в `child.html` за основу берется каталог `sandbox`. Обратите внимание, что файлы на удаленном сервере `www.example.com/air` недоступны из фрейма, так как AIR будет пытаться загрузить их из каталога app:/sandbox/.

**Атрибут allowCrossDomainXHR** Чтобы содержимое фрейма могло отправлять запросы XMLHttpRequest любому удаленному домену, включите в открывающий фрейм код `allowCrossDomainXHR="allowCrossDomainXHR"`. По умолчанию содержимое вне приложения может отправлять такие запросы только в пределах собственного домена. С междоменным API XHR связаны серьезные риски безопасности. Код на этой странице может обмениваться данными с любым доменом. Если на страницу каким-то образом проникло вредоносное содержимое, то опасность угрожает всем данным, которые доступны коду в текущей изолированной программной среде. Междоменный API XHR следует разрешать только тем страницам, которые вы лично создаете и контролируете, и только если загрузка данных с другого домена действительно необходима. Также для предотвращения внедрения вредоносного кода и иных атак рекомендуется проверять все внешние данные, загружаемые на страницу.

***Важная информация.*** *Если атрибут `allowCrossDomainXHR` включен в элемент frame или iframe, междоменный API XHR разрешен (кроме случаев, когда назначенное значение равно 0 или начинается с буквы «f» или «n»). Например, если `allowCrossDomainXHR` равно «`deny`», междоменный API XHR будет разрешен. Чтобы отключить междоменное выполнение сценариев, удалите атрибут из объявления элемента.*

**атрибут ondominitialize** Задает обработчик события `dominitialize` для фрейма. Это событие является характерным для среды AIR. Оно выполняется при создании объектов окна и документа фрейма, но до разбора какого-либо кода и до создания элементов документа.

Фрейм отправляет событие `dominitialize` на достаточно раннем этапе загрузки, чтобы любой код дочерней страницы мог ссылаться на объекты, переменные и функции, добавленные в дочерний документ обработчиком `dominitialize`. Родительская страница должна находиться в той же изолированной программной среде, что и дочерняя, чтобы напрямую добавлять объекты в дочернем документе или обращаться к ним. Тем не менее родитель в изолированной программной среде приложения может установить мост для обращения к содержимому в изолированной программной среде вне приложения.

В примерах ниже показано, как использовать тег iframe в AIR.

Документ `child.html` помещается в удаленную изолированную программную среду без сопоставления с реальным доменом или удаленным сервером.

```
<iframe src="http://localhost/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://localhost/air/"/>
```

Документ `child.html` помещается в удаленную изолированную программную среду с разрешением запросов XMLHttpRequest к домену `www.example.com`:

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"/>
```

Документ `child.html` помещается в удаленную изолированную программную среду с разрешением запросов XMLHttpRequest к любому удаленному домену.

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"
            allowCrossDomainXHR="allowCrossDomainXHR"/>
```

Документ `child.html` помещается в изолированную программную среду с доступом к локальной файловой системе.

```
<iframe     src="file:///templates/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="app-storage:/templates/"/>
```

Документ `child.html` помещается в удаленную изолированную программную среду с помощью события `dominitialize`, которое устанавливает мост.

```
<html>
<head>
<script>
var bridgeInterface = {};
bridgeInterface.testProperty = "Bridge engaged";
function engageBridge(){
    document.getElementById("sandbox").parentSandboxBridge = bridgeInterface;
}
</script>
</head>
<body>
<iframe id="sandbox"
            src="http://www.example.com/air/child.html"
            documentRoot="app:/"
            sandboxRoot="http://www.example.com/air/"
            ondominitialize="engageBridge()"/>
</body>
</html>
```

В следующем документе `child.html` показано, как содержимое дочерней изолированной программной среды может получать доступ к мосту родительской:

```
<html>
    <head>
        <script>
            document.write(window.parentSandboxBridge.testProperty);
        </script>
    </head>
    <body></body>
</html>
```

Дополнительные сведения см. в разделах ««Перекрестное выполнение сценариев в содержимом из различных изолированных программных сред безопасности» на странице 1074» и ««Безопасность HTML в Adobe AIR» на странице 1165».

## Обработчики событий элементов HTML

**Adobe AIR 1.0 и более поздних версий**

Объекты модели DOM в AIR и WebKit отправляют некоторые события, которых нет в стандартной модели событий DOM. В таблице ниже перечислены связанные атрибуты событий, которые могут использоваться для указания обработчиков этих событий.

| Имя атрибута обратного вызова | Описание |
|---|---|
| oncontextmenu | Вызывается при активации контекстного меню с помощью правой кнопки мыши или щелчка по выделенному тексту с удержанием командной клавиши. |
| oncopy | Вызывается при копировании выделенного фрагмента элемента. |
| oncut | Вызывается при вырезании выделенного фрагмента элемента. |
| ondominitialize | Вызывается при создании модели DOM документа, загружаемого во фрейм или встроенный фрейм, но до создания элементов DOM или разбора кода сценариев. |

| Имя атрибута обратного вызова | Описание |
|---|---|
| ondrag | Вызывается при перетаскивании элемента. |
| ondragend | Вызывается при отпускании после перетаскивания. |
| ondragenter | Вызывается, когда курсор при перетаскивании входит в границы элемента. |
| ondragleave | Вызывается, когда курсор при перетаскивании выходит за границы элемента. |
| ondragover | Вызывается непрерывно, пока курсор при перетаскивании находится в границах элемента. |
| ondragstart | Вызывается при начале жеста перетаскивания. |
| ondrop | Вызывается, когда жест перетаскивания прекращается на элементе. |
| onerror | Вызывается, если при загрузке элемента возникает ошибка. |
| oninput | Вызывается, когда в элемент form вводится текст. |
| onpaste | Вызывается, когда в элемент вставляются данные. |
| onscroll | Вызывается при прокрутке содержимого в элементе, поддерживающем возможность прокрутки. |
| onselectstart | Вызывается, когда начинается выделение. |

## Атрибут contentEditable в HTML

**Adobe AIR 1.0 и более поздних версий**

Атрибут `contentEditable` может быть добавлен к любому HTML-элементу. После этого пользователи смогут изменять содержимое элемента. В примере ниже в HTML-коде указано, что весь документ является редактируемым, кроме первого элемента `p`.

```
<html>
<head/>
<body contentEditable="true">
    <h1>de Finibus Bonorum et Malorum</h1>
    <p contentEditable="false">Sed ut perspiciatis unde omnis iste natus error.</p>
    <p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis.</p>
</body>
</html>
```

***Примечание.*** *Если свойство `document.designMode` имеет значение `on`, то все элементы документа можно редактировать вне зависимости от того, задано ли свойство `contentEditable` для отдельного элемента. Однако если задать для свойства `designMode` значение `off`, то элементы, для которых `contentEditable` равно `true`, по-прежнему будут редактируемыми. Дополнительные сведения см. в разделе ««Свойство Document.designMode» на странице 1042».*

## Класс Data: URL-адреса

**Adobe AIR 2 и более поздних версий**

AIR поддерживает URL-адреса класса `data:` для следующих элементов:

- img

- input type="image"
- Правила CSS, допускающие использование изображений (например, background-image)

URL-адреса класса Data позволяют вставлять двоичные данные изображения непосредственно в документ CSS или HTML в качестве строки, закодированной в формате base64. В следующем примере URL-адрес класса data: используется в качестве повторяющегося фона:

```
<html>
<head>
<style>
body {
background-
image:url('
0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAAAZQTFRF%2F6cA%2F%2F%2F%2Fgxp3lwAAAAJ0Uk5T%2FwDltzBKAAA
BF0lEQVR42uzZQQ7CMAxE0e%2F7X5oNCyRocWzPiJbMBZ6qpIljE%2BnwklgKG7kwUjc2IkIaxkY0CPdEsCCasws6ShX
BgmBBmEagpXQQLAgWBAuSY2gaKaWPYEGwIEwg0FRmECwIFoQeQjJlhJWUEFazjFDJCkI5WYRWMgjtfEGYyQnCXD4jTCd
m1zmngFpBFznwVNi5RPSbwbWnpYr%2BBHi%2FtCTfgPLEPL7jBctAKBRptXJ8M%2BprIuZKu%2BUKcg4YK1PLz7kx4bS
qHyPaT4d%2B28OCJJiRBo4FCQsSA0bziT3XubMgYUG6fc5fatmGBQkL0hoJ1IaZMiQsSFiQ8vRscTjlQOI2iHZwtpHuf
%2BJAYiOiJSkj8Z%2FIQ4ABANvXGLd3%2BZMrAAAAAElFTkSuQmCC');
background-repeat:repeat;
}
</style>
</head>
<body>
</body>
</html>
```

При использовании URL-адресов класса data: помните, что дополнительный пробел играет важную роль. Например, строка данных должна вводиться как одна неразрывная строка. В противном случае разрывы строки будут считаться частью данных и декодирование изображения будет невозможно.

## CSS в AIR

**Adobe AIR 1.0 и более поздних версий**

WebKit поддерживает некоторые расширенные свойства CSS. В большинстве из следующих расширений используется префикс: `-webkit`. Обратите внимание, что некоторые из этих расширений являются экспериментальными по сути и могут быть удалены из будущих версий WebKit. Дополнительные сведения о поддержке Webkit для CSS и соответствующих расширений для CSS см. в справочнике по CSS в Safari.

## Функции WebKit не поддерживаются в AIR

**Adobe AIR 1.0 и более поздних версий**

AIR не поддерживает следующие функции, доступные в WebKit или Safari 4.

- Междоменный обмен сообщениями с помощью метода window.postMessage (AIR обеспечивает собственные API-интерфейсы для обмена данными между доменами)
- Переменные CSS
- Web Open Font Format (WOFF) и шрифты SVG.
- Теги для видео- и аудиоданных HTML
- Запросы устройств мультимедиа

- Автономный кэш приложений
- Печать (AIR обеспечивает собственный API-интерфейс PrintJob)
- Средства проверки орфографии и грамматики
- SVG
- WAI-ARIA
- Веб-сокеты (AIR обеспечивает собственные API-интерфейсы сокетов)
- Потоки worker Web
- API-интерфейс WebKit для SQL (AIR предоставляет собственный API-интерфейс)
- API-интерфейс геопозиционирования WebKit (AIR предоставляет собственный API-интерфейс геопозиционирования в поддерживаемых устройствах)
- API-интерфейс WebKit для пакетной отправки файлов
- События прикосновения WebKit (AIR предоставляет собственные события прикосновения)
- Язык WML (Wireless Markup Language, язык разметки для беспроводных систем)

В следующем списке содержатся определенные API-интерфейсы JavaScript, элементы HTML, а также свойства и значения CSS, которые не поддерживаются в среде AIR:

**Неподдерживаемые элементы объекта Window в JavaScript:**

- applicationCache();
- console;
- openDatabase();
- postMessage();
- document.print();

**Неподдерживаемые теги HTML:**

- audio;
- video.

**Неподдерживаемые атрибуты HTML:**

- aria-*
- draggable
- formnovalidate
- list
- novalidate
- onbeforeload
- onhashchange
- onorientationchange
- onpagehide
- onpageshow
- onpopstate

- ontouchstart
- ontouchmove
- ontouchend
- ontouchcancel
- onwebkitbeginfullscreen
- onwebkitendfullscreen
- pattern
- required
- sandbox

**Неподдерживаемые события JavaScript:**

- beforeload
- hashchange
- orientationchange
- pagehide
- pageshow
- popstate
- touchstart
- touchmove
- touchend
- touchcancel
- webkitbeginfullscreen
- webkitendfullscreen

**Неподдерживаемые свойства CSS:**

- background-clip
- background-origin (используйте -webkit-background-origin)
- background-repeat-x
- background-repeat-y
- background-size (используйте -webkit-background-size)
- border-bottom-left-radius
- border-bottom-right-radius
- border-radius
- border-top-left-radius
- border-top-right-radius
- text-rendering
- -webkit-animation-play-state
- -webkit-background-clip

- -webkit-color-correction
- -webkit-font-smoothing

**Неподдерживаемые значения CSS:**

- значения свойства appearance:
  - media-volume-slider-container
  - media-volume-slider
  - media-volume-sliderthumb
  - outer-spin-button
- border-box (background-clip и background-origin)
- contain (background-size)
- content-box (background-clip и background-origin)
- cover (background-size)
- значения свойства list:
  - afar
  - amharic
  - amharic-abegede
  - cjk-earthly-branch
  - cjk-heavenly-stem
  - ethiopic
  - ethiopic-abegede
  - ethiopic-abegede-am-et
  - ethiopic-abegede-gez
  - ethiopic-abegede-ti-er
  - ethiopic-abegede-ti-et
  - ethiopic-halehame-aa-er
  - ethiopic-halehame-aa-et
  - ethiopic-halehame-am-et
  - ethiopic-halehame-gez
  - ethiopic-halehame-om-et
  - ethiopic-halehame-sid-et
  - ethiopic-halehame-so-et
  - ethiopic-halehame-ti-er
  - ethiopic-halehame-ti-et
  - ethiopic-halehame-tig
  - hangul
  - hangul-consonant

  - lower-norwegian
  - oromo
  - sidama
  - somali
  - tigre
  - tigrinya-er
  - tigrinya-er-abegede
  - tigrinya-et
  - tigrinya-et-abegede
  - upper-greek
  - upper-norwegian
- -wap-marquee (свойство отображения)

# Глава 59. Программирование HTML и JavaScript в AIR

**Adobe AIR 1.0 и более поздних версий**

Некоторые вопросы программирования касаются только разработки приложений Adobe® AIR® с помощью HTML и JavaScript. Информация в этой главе обязательна для ознакомления вне зависимости от того, разрабатываете ли вы для AIR HTML-приложение или SWF-приложение с использованием HTML и JavaScript и класса HTMLLoader (или компонента mx:HTML Flex™).

## О классе HTMLLoader

**Adobe AIR 1.0 и более поздних версий**

Класс HTMLLoader в Adobe AIR определяет экранный объект, который выводит HTML-содержимое в приложении AIR. В SWF-приложениях элемент управления HTMLLoader может быть добавлен в существующее окно, либо с помощью метода `HTMLLoader.createRootWindow()` может быть создано окно HTML, которое уже содержит объект HTMLLoader. Объект HTMLLoader доступен через свойство JavaScript `window.htmlLoader` из загруженной HTML-страницы.

### Загрузка HTML-содержимого из URL

**Adobe AIR 1.0 и более поздних версий**

Следующий программный код загружает URL в объект HTMLLoader (добавьте HTMLLoader в качестве дочернего объекта к объекту Stage или другому контейнеру отображаемых объектов для вывода HTML-содержимого в приложении):

```
import flash.html.HTMLLoader;

var html:HTMLLoader = new HTMLLoader;
html.width = 400;
html.height = 600;
var urlReq:URLRequest = new URLRequest("http://www.adobe.com/");
html.load(urlReq);
```

Свойства `width` и `height` объекта HTMLLoader по умолчанию равны 0. Эти значения можно установить при добавлении HTMLLoader в рабочую область. В процессе загрузки страницы объект HTMLLoader отправляет несколько событий. Они помогают определить, безопасно ли взаимодействие с загружаемой страницей. Эти события описаны в разделе «Обработка в AIR событий HTML» на странице 1098.

***Примечание.*** *В инфраструктуре Flex только классы-расширения класса UIComponent могут добавляться в качестве дочерних для компонентов Container. По этой причине объект HTMLLoader не может быть добавлен как дочерний для компонента Container, но элемент управления Flex mx:HTML может использоваться для создания заказного класса-расширения UIComponent, содержащего HTMLLoader в качестве дочернего объекта UIComponent. Также можно добавить объект HTMLLoader в качестве дочернего элемента UIComponent, а UIComponent добавить в контейнер Flex.*

Визуализацию HTML-текста можно осуществлять с помощью класса TextField, но его возможности ограничены. Класс TextField проигрывателя Adobe® Flash® поддерживает HTML-разметку, но из-за ограничения по размеру его возможности урезаны. (Класс HTMLLoader в среде Adobe AIR недоступен в проигрывателе Flash Player.)

## Загрузка HTML-содержимого из строки

**Adobe AIR 1.0 и более поздних версий**

Метод `loadString()` объекта HTMLLoader загружает строку HTML-содержимого в объект HTMLLoader:

```
var html:HTMLLoader = new HTMLLoader();
var htmlStr:String = "<html><body>Hello <b>world</b>.</body></html>";
html.loadString(htmlStr);
```

По умолчанию содержимое, загруженное с помощью метода `loadString()`, помещается в изолированную программную среду вне приложения, имеющую следующие характеристики.

- В ней предусмотрена возможность доступа для загрузки содержимого из сети (но не из файловой системы).
- Она не может загружать данные с помощью XMLHttpRequest.
- Для свойства `window.location` установлено значение `"about:blank"`.
- Для содержимого недоступно свойство `window.runtime` (как и для содержимого любых изолированных программных сред вне приложения).

В Adobe AIR 1.5 класс HTMLLoader включает свойство `placeLoadStringContentInApplicationSandbox`. Если этому свойству для объекта HTMLLoader установлено значение `true`, то содержимое, загружаемое методом `loadString()`, помещается в изолированную программную среду приложения. (Значение по умолчанию равно `false`.) Это обеспечивает для содержимого, загруженного методом `loadString()`, доступ к свойству `window.runtime` и ко всем прикладным интерфейсам программирования AIR. Если для этого свойства устанавливается значение `true`, убедитесь в надежности источника данных для строки, используемой при вызове метода `loadString()`. Инструкции программных кодов в строке HTML выполняются со всеми правами приложения, если для этого свойства задано значение `true`. Устанавливайте для этого свойства значение `true` только в том случае, если уверены, что данная строка содержит безопасный программный код.

В приложениях, компилируемых с помощью пакетов SDK из AIR 1.0 или AIR 1.1, содержимое, загруженное методом `loadString()`, помещается в изолированную программную среду приложения.

## Меры безопасности при использовании HTML в приложениях AIR

**Adobe AIR 1.0 и более поздних версий**

Файлы, установленные в приложение AIR, имеют доступ к API-интерфейсам AIR. Из соображений безопасности содержимое из других источников не имеет такого доступа. Например, это ограничение не позволяет содержимому с удаленного домена (например, http://example.com) читать содержимое каталога рабочего стола пользователя (и ниже по иерархии).

Из-за брешей в безопасности циклов, которые благодаря функции `eval()` (и связанным API-интерфейсам) могут быть использованы злоумышленником, содержимое, установленное вместе с приложением по умолчанию, не имеет доступа к этим методам. Однако в некоторых инфраструктурах Ajax используется вызов функции `eval()` и связанных API.

Для правильного структурирования содержимого приложения AIR следует учитывать правила безопасности и ограничения, накладываемые на каждый из типов содержимого. Содержимое из различных источников размещается отдельно в соответствии с требованиями безопасности в так называемые изолированные программные среды (см. раздел «Изолированные программные среды» на странице 1123). По умолчанию содержимое, устанавливаемое вместе с приложением, помещается в изолированную программную среду *приложения*, что дает ему доступ к API-интерфейсам AIR. Изолированная программная среда приложения является самой защищенной из всех, так как накладываемые на нее ограничения предотвращают внедрение небезопасного кода.

Среда выполнения позволяет загружать содержимое, установленное вместе с приложением, и в другие изолированные программные среды (вне приложения). Степень безопасности в изолированных программных средах вне приложения схожа со средой веб-обозревателя. Например, код в изолированной программной среде вне приложения может использовать функцию `eval()` и похожие методы (но не имеет доступа к API-интерфейсам AIR). Среда выполнения позволяет содержимому в разных изолированных программных средах взаимодействовать без открытия API-интерфейсов AIR содержимому извне приложения. Дополнительные сведения см. в разделе «Перекрестное выполнение сценариев в содержимом из различных изолированных программных сред безопасности» на странице 1074.

При вызове кода, который из соображений безопасности не может использоваться в изолированной программной среде, среда выполнения отправляет ошибку JavaScript: «Нарушение безопасности среды выполнения Adobe AIR для кода JavaScript в изолированной программной среде приложения».

Чтобы этого не происходило, следуйте рекомендациям в следующем разделе «Избежание ошибок безопасности в JavaScript» на странице 1056.

Дополнительные сведения см. в разделе ««Безопасность HTML в Adobe AIR» на странице 1165.

# Избежание ошибок безопасности в JavaScript

**Adobe AIR 1.0 и более поздних версий**

При вызове кода, который в силу ограничений безопасности не может использоваться в изолированной программной среде, среда выполнения отправляет ошибку JavaScript: «Нарушение безопасности среды выполнения Adobe AIR для кода JavaScript в изолированной программной среде приложения». Во избежание этой ошибки следуйте приведенным ниже рекомендациям.

## Причины ошибок безопасности JavaScript

**Adobe AIR 1.0 и более поздних версий**

Код, выполняемый в изолированной программной среде приложения, не имеет доступа к большинству операций, включая проверку и выполнение строк после выполнения события `load` и окончания обработки события `load`. Попытка использовать следующие инструкции JavaScript для проверки и выполнения потенциально опасных строк приведут к ошибкам JavaScript:

- функция eval();
- Функции setTimeout() и setInterval();
- Конструктор функции.

  Кроме того, следующие типы инструкций JavaScript не удастся выполнить, хотя ошибка безопасности JavaScript при этом не будет сгенерирована:

- javascript:URL
- Обратные вызовы событий, назначаемые атрибутом onevent в инструкциях innerHTML и outerHTML
- Загрузка файлов JavaScript извне приложения в установочный каталог приложения
- Методы document.write() и document.writeln()
- Синхронные запросы XMLHttpRequest до события load или во время обработки события load
- Динамически создаваемые элементы сценариев

  ***Примечание.*** *В редких случаях допускается проверка строк. Дополнительные сведения см. в разделе ««Ограничения по использованию кода в содержимом из различных изолированных программных сред» на странице 1168».*

  Список инфраструктур Ajax, которые поддерживают изолированную программную среду приложения, опубликован на сайте Adobe по адресу http://www.adobe.com/go/airappsandboxframeworks_ru.

  В последующих разделах рассказывается, как переписать сценарии, чтобы избежать ошибок безопасности JavaScript и «тихих отказов» при выполнении кода в изолированной программной среде приложения.

## Сопоставление содержимого приложения с другой изолированной программной средой

**Adobe AIR 1.0 и более поздних версий**

В большинстве случаев, чтобы избежать ошибок безопасности JavaScript, приложение можно переписать или изменить его структуру. Однако когда это невозможно, содержимое приложения может быть загружено в другую изолированную программную среду, как описано в разделе «Загрузка содержимого приложения в изолированную программную среду вне приложения» на странице 1075. Если этому содержимому также необходим доступ к API-интерфейсам AIR, можно создать мост, как описано в разделе «Настройка интерфейса моста изолированной программной среды» на странице 1076.

## функция eval()

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В изолированной программной среде приложения функция `eval()` может использоваться только до выполнения события `load` или во время обработки события `load`. После загрузки страницы вызов `eval()` не приведет к выполнению кода. Тем не менее в ряде случаев можно переписать код, чтобы не прибегать к функции `eval()`.

## Назначение свойств объекту

**Adobe AIR 1.0 и более поздних версий**

Вместо разбора строки для создания средства доступа к свойствам:

```
eval("obj." + propName + " = " + val);
```

доступ к свойствам можно получать с помощью обозначений квадратными скобками:

```
obj[propName] = val;
```

## Создание функции с переменными в контекстном меню

**Adobe AIR 1.0 и более поздних версий**

Замените инструкции:

```
function compile(var1, var2){
    eval("var fn = function(){ this."+var1+"(var2) }");
    return fn;
}
```

на:

```
function compile(var1, var2){
    var self = this;
    return function(){ self[var1](var2) };
}
```

## Создание объекта, использующего имя класса в качестве строкового параметра

**Adobe AIR 1.0 и более поздних версий**

Представим некий класс JavaScript, определенный следующим кодом:

```
var CustomClass =
    {
        Utils:
        {
            Parser: function(){ alert('constructor') }
        },
        Data:
        {

        }
    };
var constructorClassName = "CustomClass.Utils.Parser";
```

Самый простой способ создать его экземпляр — функция `eval()`:

```
var myObj;
eval('myObj=new ' + constructorClassName +'()')
```

Однако можно обойтись и без `eval()`, если разобрать каждый компонент имени класса и создать новый объект с использованием квадратных скобок:

```
function getter(str)
{
    var obj = window;
    var names = str.split('.');
    for(var i=0;i<names.length;i++){
        if(typeof obj[names[i]]=='undefined'){
            var undefstring = names[0];
            for(var j=1;j<=i;j++)
                undefstring+="."+names[j];
            throw new Error(undefstring+" is undefined");
        }
        obj = obj[names[i]];
    }
    return obj;
}
```

Экземпляр создается следующим образом:

```
try{
    var Parser = getter(constructorClassName);
    var a = new Parser();
    }catch(e){
        alert(e);
}
```

## Функции setTimeout() и setInterval()

**Adobe AIR 1.0 и более поздних версий**

Замените строку, передаваемую в виде функции обработчика, на ссылку или объект функции. Например, замените инструкцию:

```
setTimeout("alert('Timeout')", 100);
```

на:

```
setTimeout(function(){alert('Timeout')}, 100);
```

Или, когда требуется, чтобы в качестве вызывающего объекта функции использовался объект `this` замените инструкцию:

```
this.appTimer = setInterval("obj.customFunction();", 100);
```

на:

```
var _self = this;
this.appTimer = setInterval(function(){obj.customFunction.apply(_self);}, 100);
```

## Конструктор функции

**Adobe AIR 1.0 и более поздних версий**

Вместо вызова `new Function(param, body)` можно воспользоваться встроенным объявлением функции. Другой вариант — этот вызов можно осуществить до обработки события `load`.

## javascript:URL

**Adobe AIR 1.0 и более поздних версий**

Код, определенный в строке с помощью схемы URL-адресов «javascript:», игнорируется в изолированной программной среде. Ошибка безопасности JavaScript не генерируется. Ссылки можно замещать с использованием схемы «javascript: URL», например можно заменить код:

```
<a href="javascript:code()">Click Me</a>
```

на:

```
<a href="#" onclick="code()">Click Me</a>
```

## Обратные вызовы событий, назначаемые атрибутом onevent в инструкциях innerHTML и outerHTML

**Adobe AIR 1.0 и более поздних версий**

При использовании innerHTML или outerHTML для добавления элементов в модель DOM документа все обратные вызовы событий, назначенные в инструкции, например `onclick` или `onmouseover`, игнорируются. Ошибка безопасности не генерируется. Вместо этого можно назначить атрибут `id` новым элементам и задать функции обратного вызова обработчика события с помощью метода `addEventListener()`.

Например, если целевой элемент является документом, как показано ниже:

```
<div id="container"></div>
```

Можно заменить инструкции следующим образом:

```
document.getElementById('container').innerHTML =
    '<a href="#" onclick="code()">Click Me.</a>';
```

на:

```
document.getElementById('container').innerHTML = '<a href="#" id="smith">Click Me.</a>';
document.getElementById('smith').addEventListener("click", function() { code(); });
```

## Загрузка файлов JavaScript извне приложения в установочный каталог приложения

**Adobe AIR 1.0 и более поздних версий**

Загрузка файлов сценариев извне изолированной программной среды приложения запрещена. Ошибка безопасности не генерируется. Все файлы сценариев, работающие в изолированной программной среде приложения, необходимо установить в каталог приложения. Чтобы использовать на странице внешние сценарии, необходимо сопоставить ее с другой изолированной программной средой. См. раздел «Загрузка содержимого приложения в изолированную программную среду вне приложения» на странице 1075.

## Методы document.write() и document.writeln()

**Adobe AIR 1.0 и более поздних версий**

Вызовы методов `document.write()` и `document.writeln()` игнорируются после обработки события `load` для данной страницы. Ошибка безопасности не генерируется. В качестве альтернативы можно загрузить новый файл или заменить «тело» (элемент body) документа, пользуясь методиками работы с DOM.

## Синхронные запросы XMLHttpRequest до события load или во время обработки события load

**Adobe AIR 1.0 и более поздних версий**

Синхронные запросы XMLHttpRequest, инициированные до обработки события `load` страницы или во время обработки события `load`, не возвращают никакого содержимого. Асинхронные запросы XMLHttpRequest могут инициироваться, но ничего не возвращают, пока не будет завершено событие `load`. После обработки события `load` синхронные XMLHttpRequest работают как обычно.

## Динамически создаваемые элементы сценариев

**Adobe AIR 1.0 и более поздних версий**

Динамически создаваемые элементы сценариев (например, с помощью innerHTML или метода `document.createElement()`) игнорируются.

# Доступ к классам API-интерфейса AIR из JavaScript

**Adobe AIR 1.0 и более поздних версий**

Помимо стандартного и расширенного элементов Webkit, код HTML и JavaScript имеет доступ к домашним классам среды выполнения. Эти классы разрешают доступ к расширенным функциям AIR, некоторые из которых перечислены ниже:

- Доступ к файловой системе
- Использование локальных баз данных SQL
- Управление меню приложений и окон
- Доступ к сокетам для работы в сети
- Использование заданных пользователем классов и объектов
- Звуковые возможности

  Например, API-интерфейс файла AIR включает класс File, содержащийся в пакете flash.filesystem. Объект File создается в JavaScript следующим образом:

```
var myFile = new window.runtime.flash.filesystem.File();
```

Объект `runtime` является особым объектом JavaScript, который доступен HTML-содержимому в изолированной программной среде приложения AIR. С его помощью можно получать доступ из JavaScript к классам среды выполнения. Свойство `flash` объекта `runtime` предоставляет доступ к flash-пакету. В свою очередь, свойство `flash.filesystem` объекта `runtime` обеспечивает доступ к пакету flash.filesystem (а он содержит класс File). Пакеты помогают организовать классы ActionScript.

***Примечание.*** *Свойство `runtime` автоматически не добавляется к объектам window страниц, загружаемых в элементы frame или iframe. Тем не менее, пока дочерний документ находится в изолированной программной среде приложения, он имеет доступ к свойству `runtime` своего родителя.*

Так как структура пакета классов среды выполнения требует вводить длинные строки кода JavaScript для доступа к каждому из классов (например, `window.runtime.flash.desktop.NativeApplication`), в комплект AIR SDK включен файл AIRAliases.js, который открывает доступ к классам среды выполнения более простым способом (например, с помощью записи `air.NativeApplication`).

Классы API-интерфейса AIR рассматриваются в разных частях данного руководства. Другие классы API-интерфейса проигрывателя Flash Player, которые могут быть полезны разработчикам, описаны в *справочнике по API-интерфейсу Adobe AIR для HTML-разработчиков*. Язык ActionScript используется в SWF-содержимом (проигрывателе Flash Player). Синтаксисы JavaScript и ActionScript похожи. (И тот, и другой основаны на разных версиях языка ECMAScript.) Все встроенные классы доступны как для языка JavaScript (в HTML-содержимом), так и для языка ActionScript (в SWF-содержимом).

***Примечание.*** *В коде JavaScript нельзя использовать классы Dictionary, XML и XMLList, которые доступны в ActionScript.*

## Использование файла AIRAliases.js

**Adobe AIR 1.0 и более поздних версий**

Классы среды выполнения организованы в виде пакета, как показано ниже:

- `window.runtime.flash.desktop.NativeApplication`
- `window.runtime.flash.desktop.ClipboardManager`
- `window.runtime.flash.filesystem.FileStream`
- `window.runtime.flash.data.SQLDatabase`

В комплект AIR SDK входит файл AIRAliases.js, содержащий определения «псевдонимов», благодаря которым для доступа к классам среды выполнения не нужно вводить длинные строки. Например, к перечисленным выше классам можно обратиться с помощью следующих записей:

- `air.NativeApplication`
- `air.Clipboard`
- `air.FileStream`
- `air.SQLDatabase`

Здесь перечислены лишь некоторые классы в файле AIRAliases.js. Полный список классов и функций уровня пакета см. в *справочнике по API-интерфейсу Adobe AIR для HTML-разработчиков*.

Помимо широко используемых классов среды выполнения, файл AIRAliases.js содержит псевдонимы для часто используемых функций уровня пакета: `window.runtime.trace()`, `window.runtime.flash.net.navigateToURL()` и `window.runtime.flash.net.sendToURL()`, — их псевдонимами являются `air.trace()`, `air.navigateToURL()` и `air.sendToURL()`.

Чтобы задействовать файл AIRAliases.js, включите в код HTML-страницы следующую ссылку `script`:

```
<script src="AIRAliases.js"></script>
```

Проверьте правильность пути в ссылке `src`.

***Важная информация.*** *Примеры кода JavaScript в этом руководстве основаны на предположении, что файл AIRAliases.js включен в код HTML-страницы, если иное прямо не указано.*

# Об URL-адресах в AIR

**Adobe AIR 1.0 и более поздних версий**

В HTML-содержимом, работающем в среде AIR, можно использовать любую из перечисленных ниже схем URL-адресов для определения атрибутов `src` в тегах `img`, `frame`, `iframe` и `script`, атрибута `href` в теге `link` и других атрибутов, содержащих URL-адреса.

| Схема URL-адресов | Описание | Пример |
|---|---|---|
| file | Путь, заданный относительно корня файловой системы. | `file:///c:/AIR Test/test.txt` |
| app | Путь, заданный относительно корневого каталога установленного приложения. | `app:/images` |
| app-storage | Путь, заданный относительно каталога хранилища приложения. AIR определяет для каждого установленного приложения уникальный каталог хранилища, куда сохраняются все данные, относящиеся к этому приложению. | `app-storage:/settings/prefs.xml` |
| http | Стандартный HTTP-запрос. | `http://www.adobe.com` |
| https | Стандартный HTTPS-запрос. | `https://secure.example.com` |

Дополнительные сведения см. в разделе ««Схемы URI» на странице 874».

Многие API-интерфейсы AIR, включая интерфейсы классов File, Loader, URLStream и Sound, используют объект URLRequest, а не строки с URL. Для создания экземпляров объекта URLRequest используется строка, в которой могут содержаться такие схемы URL-адресов. Например, следующая инструкция создает объект URLRequest, который может использоваться для запроса домашней страницы сайта Adobe:

```
var urlReq = new air.URLRequest("http://www.adobe.com/");
```

Сведения об объектах URLRequest см. в разделе ««Связь HTTP» на странице 872».

# Открытие доступа к объектам ActionScript для JavaScript

**Adobe AIR 1.0 и более поздних версий**

JavaScript на HTML-странице, загружаемой объектом HTMLLoader, может вызывать классы, объекты и функции, определенные контекстом выполнения ActionScript с помощью свойств `window.runtime`, `window.htmlLoader` и `window.nativeWindow` HTML-страницы. Кроме того, объекты и функции ActionScript, могут быть доступны коду JavaScript, если создать на них ссылки к контексте выполнения JavaScript.

## Простой пример доступа к объектам JavaScript из ActionScript

**Adobe AIR 1.0 и более поздних версий**

В примере ниже показано, как добавлять свойства, указывающие на глобальный объект окна HTML-страницы:

```
 var html:HTMLLoader = new HTMLLoader();
var foo:String = "Hello from container SWF."
function helloFromJS(message:String):void {
    trace("JavaScript says:", message);
}
var urlReq:URLRequest = new URLRequest("test.html");
html.addEventListener(Event.COMPLETE, loaded);
html.load(urlReq);

function loaded(e:Event):void{
    html.window.foo = foo;
    html.window.helloFromJS = helloFromJS;
}
```

HTML-содержимое (в файле test.html), загруженное в объект HTMLLoader в предыдущем примере, имеет доступ к свойству `foo` и методу `helloFromJS()`, которые определены в родительском SWF-файле:

```
 <html>
    <script>
        function alertFoo() {
            alert(foo);
        }
    </script>
    <body>
        <button onClick="alertFoo()">
            What is foo?
        </button>
        <p><button onClick="helloFromJS('Hi.')">
            Call helloFromJS() function.
        </button></p>
    </body>
</html>
```

При доступе к контексту JavaScript загружаемого документа можно использовать событие `htmlDOMInitialize` для создания объектов на раннем этапе построения страницы, чтобы все заданные сценарии могли к ним обратиться. Если дождаться события `complete`, то только сценарии, запускаемые после события `load`, будут иметь доступ к добавляемым объектам.

## Открытия определениям классов доступа к JavaScript

**Adobe AIR 1.0 и более поздних версий**

Чтобы классы ActionScript были доступны для JavaScript, можно назначить загружаемое HTML-содержимое домену приложения, содержащему определения классов. Домен приложения в контексте выполнения JavaScript может быть задан с помощью свойства `runtimeApplicationDomain` объекта HTMLLoader. Чтобы сделать домен приложения основным, задайте для `runtimeApplicationDomain` значение `ApplicationDomain.currentDomain`, как показано ниже:

```
html.runtimeApplicationDomain = ApplicationDomain.currentDomain;
```

После определения `runtimeApplicationDomain` контекст JavaScript пользуется определениями классов совместно с назначенным доменом. Для создания экземпляра заказного класса в JavaScript задайте на него ссылку с помощью свойства `window.runtime` и оператора `new`:

```
var customClassObject = new window.runtime.CustomClass();
```

HTML-содержимое должно приходить с домена, подходящего по параметрам безопасности. Если HTML-содержимое приходит с домена, отличающегося от назначенного домена приложения, страница будет использовать домен приложения по умолчанию. Например, при загрузке удаленной страницы из Интернета ApplicationDomain.currentDomain нельзя назначить доменом приложения для страницы.

## Удаление прослушивателей событий

**Adobe AIR 1.0 и более поздних версий**

Если прослушиватели событий JavaScript добавлены к объектам вне текущей страницы, включая объекты среды выполнения, объекты загруженного SWF-содержимого и даже объекты JavaScript, выполняемые на других страницах, необходимо удалять эти прослушиватели при выгрузке страницы. В противном случае прослушиватель будет отправлять событие функции обработчика, которой более не существует. В этом случае будет выведено следующее сообщение об ошибке: «Приложение попыталось обратиться к объекту JavaScript на HTML-странице, которая уже выгружена». Удаление ненужных прослушивателей также позволяет среде AIR восстановить задействованную память. Дополнительные сведения см. в разделе ««Удаление прослушивателей событий на HTML-страницах с навигацией» на странице 1103».

# Доступ к объектам DOM и JavaScript из ActionScript

**Adobe AIR 1.0 и более поздних версий**

Когда объект HTMLLoader отправляет событие `complete`, все объекты модели объекта документа (DOM) в HTML становятся доступны. Доступные объекты включают отображаемые элементы (такие как объекты `div` и `p` на странице) и переменные и функции JavaScript. Событие `complete` соответствует событию `load` (загрузки страницы) в JavaScript. До отправки события `complete` элементы, переменные и функции DOM, скорее всего, нельзя будет разобрать или создать. По возможности следует дождаться события `complete` до того, как обращаться к модели DOM HTML.

Например, рассмотрим следующую HTML-страницу:

```
<html>
    <script>
        foo = 333;
        function test() {
            return "OK.";
        }
    </script>
    <body>
        <p id="p1">Hi.</p>
    </body>
</html>
```

На этой простой HTML-странице определяется переменная JavaScript *foo* и функция JavaScript *test()*. И то, и другое являются свойствами глобального объекта `window` страницы. Кроме того, объект `window.document` включает элемент P (с ID *p1*), доступ к которому осуществляется с помощью метода `getElementById()`. После загрузки страницы (когда объект HTMLLoader отправляет событие `complete`) каждый из этих объектов доступен из ActionScript, как показано ниже:

```
var html:HTMLLoader = new HTMLLoader();
html.width = 300;
html.height = 300;
html.addEventListener(Event.COMPLETE, completeHandler);
var xhtml:XML =
    <html>
        <script>
            foo = 333;
            function test() {
                return "OK.";
            }
        </script>
        <body>
            <p id="p1">Hi.</p>
        </body>
    </html>;
html.loadString(xhtml.toString());

function completeHandler(e:Event):void {
    trace(html.window.foo); // 333
    trace(html.window.document.getElementById("p1").innerHTML); // Hi.
    trace(html.window.test()); // OK.
}
```

Для доступа к содержимому HTML-элемента используйте свойство `innerHTML`. Например, в коде выше свойство `html.window.document.getElementById("p1").innerHTML` используется для получения содержимого HTML-элемента *p1*.

Свойства HTML-страницы можно задать через ActionScript. Например, в примере ниже содержимое элемента `p1` и значение переменной JavaScript `foo` задаются с помощью ссылки на объект-контейнер HTMLLoader:

```
html.window.document.getElementById("p1").innerHTML = "Goodbye";
html.window.foo = 66;
```

# Вложение SWF-содержимого в HTML

**Adobe AIR 1.0 и более поздних версий**

В приложении AIR SWF-содержимое можно встроить в HTML-содержимое точно так же, как в обозревателе. Встроить SWF-содержимое можно с помощью тега `object`, тега `embed` или обоих.

***Примечание.*** *Общие рекомендации по веб-разработке предполагают использование и тега `object`, и тега `embed` для отображения SWF-содержимого на HTML-странице. Эти рекомендации не помогут при работе с AIR. Тег `object` стандарта W3C может сам по себе использоваться в содержимом, отображаемом в AIR. В то же время теги `object` и `embed` могут при необходимости использоваться вместе — такая необходимость может возникнуть при отображении HTML-содержимого в обозревателе.*

Если в объекте NativeWindow, отображающим содержимое HTML и SWF, включена прозрачность, то AIR не отобразит содержимое SWF, если режим окна (`wmode`), в которое внедрено содержимое, принимает значение `window`. В данном случае для отображения содержимого SWF на прозрачной странице HTML задайте для параметра `wmode` значение `opaque` или `transparent`. По умолчанию параметр `wmode` использует значение `window`, поэтому содержимое может не отображаться, если не задать требуемое значение.

В примере ниже показано, как с помощью тега HTML `object` отобразить SWF-файл в содержимом HTML. Параметру `wmode` присваивается значение `opaque`, чтобы содержимое отображалось даже в том случае, когда под ним находится прозрачный объект NativeWindow. SWF-файл загружается из каталога приложения, но использоваться может любая схема URL-адресов, поддерживаемая AIR. (Расположение, из которого загружается SWF-файл, определяет защищенную изолированную программную среду, в которую AIR помещает содержимое.)

```
<object type="application/x-shockwave-flash" width="100%" height="100%">
    <param name="movie" value="app:/SWFFile.swf"></param>
    <param name="wmode" value="opaque"></param>
</object>
```

Можно также загрузить содержимое динамически при помощи сценария. В следующем примере узел `object` создается для отображения SWF-файла, заданного параметром `urlString`. В примере узел добавляется в качестве дочернего элемента страницы с ID, указанным в параметре `elementID`:

```
<script>
function showSWF(urlString, elementID){
    var displayContainer = document.getElementById(elementID);
    var flash = createSWFObject(urlString, 'opaque', 650, 650);
    displayContainer.appendChild(flash);
}
function createSWFObject(urlString, wmodeString, width, height){
    var SWFObject = document.createElement("object");
    SWFObject.setAttribute("type","application/x-shockwave-flash");
    SWFObject.setAttribute("width","100%");
    SWFObject.setAttribute("height","100%");
    var movieParam = document.createElement("param");
    movieParam.setAttribute("name","movie");
    movieParam.setAttribute("value",urlString);
    SWFObject.appendChild(movieParam);
    var wmodeParam = document.createElement("param");
    wmodeParam.setAttribute("name","wmode");
    wmodeParam.setAttribute("value",wmodeString);
    SWFObject.appendChild(wmodeParam);
    return SWFObject;
}
</script>
```

Содержимое SWF не отображается при изменении масштаба или повороте объекта HTMLLoader, а также если свойству `alpha` присвоено значение, отличное от 1.0. В версиях, предшествовавших AIR 1.5.2, содержимое SWF не отображалось в окне с прозрачностью при любых значениях `wmode`.

***Примечание.*** *Когда внедренный объект SWF пытается загрузить внешний ресурс, например видеофайл, SWF-содержимое может быть некорректно визуализировано, если в HTML-файле не указан абсолютный путь к видеофайлу. Тем не менее внедренный объект SWF может загрузить внешний файл изображения с использованием относительного пути.*

В этом примере демонстрируется, как внешние ресурсы можно загрузить через объект SWF, внедренный в HTML-содержимое:

```
var imageLoader;

function showSWF(urlString, elementID){
    var displayContainer = document.getElementById(elementID);
    imageLoader = createSWFObject(urlString,650,650);
    displayContainer.appendChild(imageLoader);
}

function createSWFObject(urlString, width, height){

    var SWFObject = document.createElement("object");
    SWFObject.setAttribute("type","application/x-shockwave-flash");
    SWFObject.setAttribute("width","100%");
    SWFObject.setAttribute("height","100%");

    var movieParam = document.createElement("param");
    movieParam.setAttribute("name","movie");
    movieParam.setAttribute("value",urlString);
    SWFObject.appendChild(movieParam);

    var flashVars = document.createElement("param");
    flashVars.setAttribute("name","FlashVars");

    //Load the asset inside the SWF content.
    flashVars.setAttribute("value","imgPath=air.jpg");
    SWFObject.appendChild(flashVars);

    return SWFObject;
}
function loadImage()
{
  showSWF("ImageLoader.swf", "imageSpot");

}
```

В следующем примере ActionScript считывается путь к изображению, передаваемый HTML-файлом, и изображение загружается в рабочую область:

```
package
{
      import flash.display.Sprite;
      import flash.display.LoaderInfo;
      import flash.display.StageScaleMode;
      import flash.display.StageAlign;
      import flash.display.Loader;
      import flash.net.URLRequest;

      public class ImageLoader extends Sprite
      {
            public function ImageLoader()
            {

                  var flashvars = LoaderInfo(this.loaderInfo).parameters;

                  if(flashvars.imgPath){
                        var imageLoader = new Loader();
                        var image = new URLRequest(flashvars.imgPath);
                        imageLoader.load(image);
                        addChild(imageLoader);
                        imageLoader.x = 0;
                        imageLoader.y = 0;
                        stage.scaleMode=StageScaleMode.NO_SCALE;
                        stage.align=StageAlign.TOP_LEFT;
                  }
            }
      }
}
```

# Использование библиотек ActionScript на HTML-странице

**Adobe AIR 1.0 и более поздних версий**

AIR расширяет элемент HTML-сценария, и страница может загружать классы ActionScript в скомпилированный SWF-файл. Например, для импорта библиотеки *myClasses.swf*, расположенной в подкаталоге `lib` в корневом каталоге приложения, включите в HTML-файл следующий код:

```
<script src="lib/myClasses.swf" type="application/x-shockwave-flash"></script>
```

***Важная информация.*** *Чтобы библиотека загрузилась нормально, значение атрибута должно быть* `type="application/x-shockwave-flash"`.

Если SWF-содержимое компилируется как SWF-файл для проигрывателя Adobe Flash 10 или AIR 1.5, необходимо установить в качестве пространства имен XML в файле дескриптора приложения пространство имен AIR 1.5.

Каталог `lib` и файл `myClasses.swf` также должны быть включены в пакет с файлом AIR.

Импортированные классы могут быть доступны через свойство `runtime` объекта Window в JavaScript:

```
var libraryObject = new window.runtime.LibraryClass();
```

Если классы SWF-файла организованы в виде пакетов, необходимо также включать имя пакета. Например, если определение LibraryClass содержится в пакете *utilities*, экземпляр класса создается с помощью следующей инструкции:

```
var libraryObject = new window.runtime.utilities.LibraryClass();
```

***Примечание.*** *Для компиляции SWF-библиотеки ActionScript и использования ее на HTML-странице в AIR пользуйтесь компилятором* `acompc`*. Утилита acompc входит в пакет Flex SDK и описана в документации по Flex SDK.*

## Доступ к объектам DOM и JavaScript в HTML из импортированного файла ActionScript

**Adobe AIR 1.0 и более поздних версий**

Для доступа к объектам на HTML-странице из ActionScript в SWF-файле, импортированном в страницу с помощью тега `<script>`, передайте ссылку на объект JavaScript, например `window` или `document`, функции, определенной в коде ActionScript. С помощью ссылки, включенной в функцию, можно обратиться к объекту JavaScript (или другим объектам, доступным по передаваемой ссылке).

Например, рассмотрим следующую HTML-страницу:

```
<html>
    <script src="ASLibrary.swf" type="application/x-shockwave-flash"></script>
    <script>
        num = 254;
        function getStatus() {
            return "OK.";
        }
        function runASFunction(window){
            var obj = new runtime.ASClass();
            obj.accessDOM(window);
        }
    </script>
    <body onload="runASFunction">
        <p id="p1">Body text.</p>
    </body>
</html>
```

На этой простой HTML-странице определяется переменная JavaScript *num* и функция JavaScript *getStatus()*. Обе они являются свойствами объекта `window` страницы. Кроме того, объект `window.document` включает названный элемент P (с ID *p1*).

Страница загружает файл ActionScript ASLibrary.swf, содержащий класс ASClass. ASClass определяет функцию с именем `accessDOM()`, которая просто отслеживает значения этих объектов JavaScript. Метод `accessDOM()` принимает в качестве аргумента объект JavaScript Window. С помощью ссылки Window он получает доступ к другим объектам на странице, включая переменные, функции и элементы DOM, как показано в определении ниже:

```
public class ASClass{
    public function accessDOM(window:*):void {
        trace(window.num); // 254
        trace(window.document.getElementById("p1").innerHTML); // Body text..
        trace(window.getStatus()); // OK.
    }
}
```

И получать, и задавать свойства HTML-страницы можно из импортированного класса ActionScript. Например, следующая функция задает содержимое элемента `p1` и значение переменной JavaScript `foo` на странице:

```
public function modifyDOM(window:*):void {
    window.document.getElementById("p1").innerHTML = "Bye";
    window.foo = 66;
```

# Преобразование объектов Date и RegExp

**Adobe AIR 1.0 и более поздних версий**

В обоих языках — JavaScript и ActionScript — определяются классы Date и RegExp, но объекты этих типов не преобразуются между их контекстами выполнения автоматически. Объекты Date и RegExp необходимо преобразовать, прежде чем задавать с их помощью параметры свойств и функций в другом контексте выполнения.

Например, код ActionScript преобразует объект JavaScript Date с именем `jsDate` в объект ActionScript Date:

```
var asDate:Date = new Date(jsDate.getMilliseconds());
```

Показанный ниже код ActionScript преобразует объект JavaScript RegExp с именем `jsRegExp` в объект RegExp языка ActionScript:

```
var flags:String = "";
if (jsRegExp.dotAll) flags += "s";
if (jsRegExp.extended) flags += "x";
if (jsRegExp.global) flags += "g";
if (jsRegExp.ignoreCase) flags += "i";
if (jsRegExp.multiline) flags += "m";
var asRegExp:RegExp = new RegExp(jsRegExp.source, flags);
```

# Управление таблицей стилей HTML из ActionScript

**Adobe AIR 1.0 и более поздних версий**

После отправки объектом HTMLLoader события `complete` CSS-стили страницы становятся доступны для изучения и управления.

Например, рассмотрим несложный HTML-документ:

```
<html>
<style>
    .style1A { font-family:Arial; font-size:12px }
    .style1B { font-family:Arial; font-size:24px }
</style>
<style>
    .style2 { font-family:Arial; font-size:12px }
</style>
<body>
    <p class="style1A">
        Style 1A
    </p>
    <p class="style1B">
        Style 1B
    </p>
    <p class="style2">
        Style 2
    </p>
</body>
</html>
```

После того как объект HTMLLoader загружает это содержимое, CSS-стилями на странице можно управлять с помощью массива `cssRules` массива `window.document.styleSheets`, как показано ниже:

```
var html:HTMLLoader = new HTMLLoader( );
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.addEventListener(Event.COMPLETE, completeHandler);
function completeHandler(event:Event):void {
    var styleSheet0:Object = html.window.document.styleSheets[0];
    styleSheet0.cssRules[0].style.fontSize = "32px";
    styleSheet0.cssRules[1].style.color = "#FF0000";
    var styleSheet1:Object = html.window.document.styleSheets[1];
    styleSheet1.cssRules[0].style.color = "blue";
    styleSheet1.cssRules[0].style.font-family = "Monaco";
}
```

Этот код подстраивает CSS-стили таким образом, чтобы получившийся HTML-документ выглядел следующим образом:

Style 1A

Style 1B

Style 2

Помните, что после того как объект HTMLLoader отправляет событие `complete`, можно добавлять новые стили для страницы.

# Перекрестное выполнение сценариев в содержимом из различных изолированных программных сред безопасности

**Adobe AIR 1.0 и более поздних версий**

Модель безопасности среды выполнения изолирует код из различных источников. Перекрестное выполнение сценариев содержимого в разных изолированных программных средах безопасности позволяет содержимому в одной из них обращаться к некоторым свойствам и методам в другой.

## Защищенные изолированные программные среды AIR и код JavaScript

**Adobe AIR 1.0 и более поздних версий**

Политика AIR также основывается на определении происхождения, что не позволяет коду с одного домена обращаться к коду с другого. Все файлы помещаются в изолированные программные среды на основании своего происхождения. Как правило, содержимое в изолированной программной среде приложения не может нарушать этот принцип и выполнять перекрестное выполнение сценариев содержимого, загруженного извне установочного каталога приложения. Однако AIR предлагает несколько методов перекрестного выполнения сценариев содержимого вне приложения.

Один из методов использует фреймы и встроенные фреймы для привязки содержимого к защищенной изолированной программной среде. Все страницы, загружаемые из изолированной программной среды приложения, ведут себя так, будто бы они загружены с удаленного домена. Например, сопоставление содержимого приложения с доменом *example.com* позволяет перекрестное выполнение сценариев для страниц, загруженных с example.com.

Так как этот метод помещает содержимое приложения в другую изолированную программную среду, код в этом содержимом более не подчиняется ограничениям по выполнению кода в проверяемых строках. Техника сопоставления для изолированных программных сред позволяет снимать эти ограничения, даже если нет необходимости в перекрестном выполнении сценариев удаленного содержимого. Это особенно полезно при работе с одной из многочисленных инфраструктур JavaScript или с существующим кодом, который основывается на проверке строк. Однако необходимо помнить о дополнительных рисках внедрения и выполнения вредоносного содержимого при запуске содержимого вне изолированной программной среды приложения.

В то же время содержимое приложения, которое сопоставлено с другой изолированной программной средой, теряет доступ к API-интерфейсам AIR, поэтому техника сопоставления с изолированной программной средой не подходит для представления функциональности AIR коду, выполняемому вне изолированной программной среды приложения.

Еще один способ перекрестного выполнения сценариев позволяет создать интерфейс *моста изолированной программной среды* между содержимым в изолированной программной среде вне приложения и его родительским документом в изолированной программной среде приложения. Мост предоставляет дочернему содержимому доступ к свойствам и методам, заданным родителем, а родителю — доступ к свойствам и методам, заданным дочерним элементом (или и то и другое сразу).

Наконец, междоменные запросы XMLHttpRequest могут выполняться из изолированной программной среды приложения и из других изолированных программных сред (необязательно).

Дополнительные сведения см. в разделах ««Элементы frame и iframe в HTML» на странице 1045», ««Безопасность HTML в Adobe AIR» на странице 1165» и ««Объект XMLHttpRequest» на странице 1039».

## Загрузка содержимого приложения в изолированную программную среду вне приложения

**Adobe AIR 1.0 и более поздних версий**

Чтобы содержимое приложения могло выполнять безопасное перекрестное выполнение сценариев содержимого, загруженного из установочного каталога приложения, для загрузки содержимого приложения в ту же изолированную программную среду, что и внешнего содержимого, используется элемент `frame` или `iframe`. Если перекрестное выполнение сценариев удаленного содержимого не требуется, но требуется загрузить страницу приложения вне его изолированной программной среды, можно воспользоваться тем же способом и задать в качестве домена по умолчанию `http://localhost/` или любое другое безопасное значение.

AIR добавляет в элемент frame новые атрибуты `sandboxRoot` и `documentRoot`, которые позволяют указать, нужно ли сопоставлять файл приложения, загруженный во фрейм, с изолированной программной средой вне приложения. Вместо этого файлы, указанные в URL-адресе `sandboxRoot`, загружаются в каталог `documentRoot`. Из соображений безопасности содержимое приложения, загружаемое таким способом, обрабатывается так, будто оно загружено из URL-адреса `sandboxRoot`.

Свойство `sandboxRoot` задает URL-адрес для определения изолированной программной среды и домена, куда следует поместить содержимое. Следует использовать схему URL-адресов `file:`, `http:` или `https:`. При указании относительного URL-адреса содержимое остается в изолированной программной среде приложения.

Свойство `documentRoot` указывает каталог, из которого загружается содержимое фрейма. Следует использовать схему URL-адресов `file:`, `app:` или `app-storage:`.

В примере ниже содержимое, установленное в подкаталог приложения `sandbox`, предназначено для запуска в удаленной изолированной программной среде и домене `www.example.com`:

```
<iframe
    src="http://www.example.com/local/ui.html"
    sandboxRoot="http://www.example.com/local/"
    documentRoot="app:/sandbox/">
</iframe>
```

Страница `ui.html` может загружать файл javascript из локальной папки `sandbox` с помощью следующего тега:

```
<script src="http://www.example.com/local/ui.js"></script>
```

Содержимое также может загружаться из каталога на удаленном сервере с помощью тега script, как показано ниже:

```
<script src="http://www.example.com/remote/remote.js"></script>
```

URL `sandboxRoot` маскирует все содержимое с этого же URL-адресу на удаленном сервере. В предыдущем примере доступ к удаленному содержимому на домене `www.example.com/local/` (или любом из его подкаталогов) запрещен, так как AIR повторно сопоставляет запросы с локальным каталогом приложения. Запросы сопоставляются повторно независимо от того, какой способ загрузки содержимого вызывает их: навигация по странице, запрос XMLHttpRequest или любой другой.

## Настройка интерфейса моста изолированной программной среды

**Adobe AIR 1.0 и более поздних версий**

Мост изолированной программной среды может использоваться, когда необходим доступ к свойствам или методам в изолированной программной среде вне приложения или когда содержимое вне приложения должно обращаться к свойствам и методам, определенным содержимым изолированной программной среды приложения. Используйте свойства `childSandboxBridge` и `parentSandboxBridge` для создания объекта `window` любого дочернего документа.

## Установка моста дочерней изолированной программной среды

**Adobe AIR 1.0 и более поздних версий**

Свойство `childSandboxBridge` позволяет дочернему документу предоставлять интерфейс содержимому родительского документа. Для этого задайте в качестве значения свойства `childSandbox` функцию или объект в дочернем документе. После этого объект или функция будут доступны из содержимого родительского документа. В примере ниже показан сценарий, выполняемый в дочернем документе, который предоставляет своему родителю объект, содержащий функцию и свойство:

```
var interface = {};
interface.calculatePrice = function(){
    return ".45 cents";
}
interface.storeID = "abc"
window.childSandboxBridge = interface;
```

Если дочернее содержимое загружается во встроенный фрейм, у которого есть id «child», то доступ к интерфейсу из родительского содержимого осуществляется путем чтения свойства фрейма `childSandboxBridge`:

```
var childInterface = document.getElementById("child").contentWindow.childSandboxBridge;
air.trace(childInterface.calculatePrice()); //traces ".45 cents"
air.trace(childInterface.storeID)); //traces "abc"
```

## Установка моста родительской изолированной программной среды

**Adobe AIR 1.0 и более поздних версий**

Свойство `parentSandboxBridge` позволяет родительскому документу предоставлять интерфейс содержимому дочернего документа. Для этого в качестве значения свойства `parentSandbox` дочернего документа назначается функция или объект в родительском документе. После этого объект или функция будут доступны из содержимого дочернего элемента. В примере ниже показан сценарий, выполняемый в родительском фрейме, который предоставляет своему дочернему элементу объект, содержащий функцию:

```
var interface = {};
interface.save = function(text){
    var saveFile = air.File("app-storage:/save.txt");
    //write text to file
}
document.getElementById("child").contentWindow.parentSandboxBridge = interface;
```

С помощью этого интерфейса содержимое в дочернем фрейме может сохранять текст в файл `save.txt`, но других способов доступа к файловой системе у него нет. Дочернее содержимое может вызывать функцию сохранения следующим образом:

```
var textToSave = "A string.";
window.parentSandboxBridge.save(textToSave);
```

Содержимое приложения должно предоставлять другим изолированным программным средам максимально строгий интерфейс. Содержимое вне приложения следует расценивать как небезопасное, потому что оно может подвергаться случайному или злонамеренному внедрению кода. Для защиты от злонамеренного использования интерфейса, который предоставляется через мост родительской изолированной программной среды следует пользоваться соответствующими мерами безопасности.

## Доступ к мосту родительской изолированной программной среды во время загрузки страницы

**Adobe AIR 1.0 и более поздних версий**

Чтобы код дочернего документа имел доступ к мосту родительской изолированной программной среды, необходимо установить мост до того, как начнет выполняться сценарий. Объекты window, frame и iframe отправляют событие `dominitialize` после создания новой модели DOM для страницы, но до разбора какого-либо сценария или добавления элементов DOM. Событие `dominitialize` позволяет установить мост для создания объектов на раннем этапе построения страницы, чтобы все сценарии дочернего документа могли к ним обратиться.

В примере ниже показано, как создать мост родительской изолированной программной среды в ответ на событие `dominitialize`, отправляемое из дочернего фрейма:

```
<html>
<head>
<script>
var bridgeInterface = {};
bridgeInterface.testProperty = "Bridge engaged";
function engageBridge(){
    document.getElementById("sandbox").contentWindow.parentSandboxBridge = bridgeInterface;
}
</script>
</head>
<body>
<iframe id="sandbox"
            src="http://www.example.com/air/child.html"
            documentRoot="app:/"
            sandboxRoot="http://www.example.com/air/"
            ondominitialize="engageBridge()"/>
</body>
</html>
```

В следующем документе `child.html` показано, как содержимое дочерней изолированной программной среды может получать доступ к мосту родительской:

```
<html>
    <head>
        <script>
            document.write(window.parentSandboxBridge.testProperty);
        </script>
    </head>
    <body></body>
</html>
```

Чтобы прослушивать событие `dominitialize` в дочернем окне, а не во фрейме, следует добавить прослушиватель новому дочернему объекту window, который создан функцией `window.open()`:

```
 var childWindow = window.open();
childWindow.addEventListener("dominitialize", engageBridge());
childWindow.document.location = "http://www.example.com/air/child.html";
```

В этом случае содержимое приложения не может быть сопоставлено с изолированной программной средой вне приложения. Этот метод подходит только для случаев, когда страница `child.html` загружается извне каталога приложения. Содержимое приложения в окне по-прежнему может быть сопоставлено с изолированной программной средой вне приложения, но для этого сначала нужно загрузить промежуточную страницу, которая также использует фреймы для загрузки дочернего документа, и сопоставить ее с нужной изолированной программной средой.

Если для создания окна используется функция `createRootWindow()` класса HTMLLoader, новое окно не становится дочерним элементом документа, из которого вызывается функция `createRootWindow()`. Таким образом, мост изолированной программной среды не может быть создан между вызывающим окном и содержимым вне приложения, загружаемым в новое окно. Вместо этого следует загрузить в новом окне промежуточную страницу, которая использует фреймы для загрузки дочернего содержимого. Можно установить мост от родительского документа нового окна к дочернему документу, загружаемому во фрейм.

# Глава 60. Написание сценария для HTML-контейнера AIR

**Adobe AIR 1.0 и более поздних версий**

Класс HTMLLoader служит контейнером для содержимого HTML в Adobe® AIR®. Этот класс содержит много свойств и методов, унаследованных от класса Sprite, для управления поведением и внешним видом объекта в списке отображения ActionScript® 3.0. Более того, этот класс определяет свойства и методы для таких задач, как загрузка содержимого HTML, взаимодействие с ним и управление историей.

Класс HTMLHost определяет набор вариантов поведения по умолчанию для HTMLLoader. При создании объекта HTMLLoader реализация HTMLHost не выполняется. Таким образом, когда содержимое HTML инициирует один из вариантов поведения по умолчанию, например изменение местоположения окна или заголовка окна, ничего не происходит. Класс HTMLHost можно расширить для определения вариантов поведения, необходимых для вашего приложения.

Реализация HTMLHost по умолчанию выполняется для окон HTML, созданных при помощи AIR. Можно назначить реализацию HTMLHost по умолчанию другому объекту HTMLLoader, задав свойство `htmlHost` объекта при помощи нового объекта HTMLHost, созданного с использованием параметра `defaultBehavior`, установленного на значение `true`.

***Примечание.*** *В Adobe® Flex™ Framework объект HTMLLoader заключен в оболочку компонента mx:HTML. При работе с Flex используйте компонент HTML.*

## Свойства отображения объектов HTMLLoader

**Adobe AIR 1.0 и более поздних версий**

Объект HTMLLoader наследует свойства отображения класса Sprite в Adobe® Flash® Player. Например, можно изменить размер объекта, переместить или скрыть его, а также изменить цвет фона объекта. Можно также применить дополнительные эффекты: фильтры, маски, масштабирование и поворот. В случае применения эффектов следует учитывать их влияние на удобочитаемость. При применении некоторых эффектов содержимое SWF и PDF, загруженное на страницу HTML, может не отображаться.

Окна HTML содержат объект HTMLLoader, отвечающий за визуализацию содержимого HTML. Этот объект ограничен площадью окна, поэтому изменение размеров, положения, угла поворота или масштаба не всегда приводит к желаемым результатам.

### Основные свойства отображения

**Adobe AIR 1.0 и более поздних версий**

Основные свойства отображения HTMLLoader позволяют расположить элемент управления внутри родительского экранного объекта, установить размер, отобразить или скрыть элемент управления. Не следует изменять эти свойства для объекта HTMLLoader окна HTML.

В число основных свойств входят следующие:

| Свойство | Примечания |
|---|---|
| `x`, `y` | Располагает объект внутри родительского контейнера. |
| `width`, `height` | Изменяет размеры области отображения. |
| `visible` | Управляет видимостью объекта и его содержимого. |

За пределами окна HTML свойства `width` и `height` объекта HTMLLoader по умолчанию установлены на 0. Ширину и высоту необходимо задать до того, как будет видимо содержимое HTML. Содержимое HTML подгоняется под размер HTMLLoader в соответствии со свойствами HTML и CSS содержимого. При изменении размера HTMLLoader содержимое загружается повторно.

При загрузке содержимого в новый объект HTMLLoader (со свойством `width`, установленным на 0) возникает соблазн задать свойства `width` и `height` объекта HTMLLoader при помощи свойств `contentWidth` и `contentHeight`. Такой метод работает для страниц с достаточной минимальной шириной, если они развернуты в соответствии с правилами HTML и спецификации CSS. Однако некоторые страницы имеют длинный и узкий макет из-за отсутствия достаточной ширины, обеспечиваемой объектом HTMLLoader.

***Примечание.*** *При изменении ширины и высоты объекта HTMLLoader значения scaleX и scaleY не изменяются, как в случае с большинством других типов экранных объектов.*

## Прозрачность содержимого HTMLLoader

**Adobe AIR 1.0 и более поздних версий**

Свойство `paintsDefaultBackground` объекта HTMLLoader, которое по умолчанию имеет значение `true`, задает непрозрачный фон объекта HTMLLoader. Если свойство `paintsDefaultBackground` имеет значение `false`, фон прозрачный. Контейнер объектов отображения или другие экранные объекты, расположенные под объектом HTMLLoader, видимы за элементами основного фона содержимого HTML.

Если для элемента основной части документа HTML или другого его элемента задан фоновый цвет (с использованием `style="background-color:gray"`, например), то фон этой части документа HTML непрозрачный и отображается с заданным фоновым цветом. Если задать свойство `opaqueBackground` объекта HTMLLoader при свойстве `paintsDefaultBackground` установленным на `false`, то цвет, заданный для свойства `opaqueBackground`, становится видимым.

***Примечание.*** *Можно использовать прозрачную графику формата PNG для применения к элементу документа HTML полупрозрачного фона. Установка стиля непрозрачности для элемента HTML не поддерживается.*

## Масштабирование содержимого HTMLLoader

**Adobe AIR 1.0 и более поздних версий**

Избегайте изменения масштаба объекта HTMLLoader выше коэффициента 1.0. Текст содержимого HTMLLoader отображается при определенном разрешении и при увеличении масштаба объекта HTMLLoader выглядит пикселизированным. Во избежание масштабирования объекта HTMLLoader, а также его содержимого, при изменении размера окна установите свойство `scaleMode` объекта Stage на значение `StageScaleMode.NO_SCALE`.

## Примечания по загрузке содержимого SWF или PDF на страницу HTML

**Adobe AIR 1.0 и более поздних версий**

Содержимое SWF и PDF, загруженное в объект HTMLLoader, исчезает при следующих условиях:

- При масштабировании объекта HTMLLoader выше или ниже коэффициента 1.0.
- При установке свойства alpha объекта HTMLLoader на значение, отличное от 1.0.
- При вращении содержимого HTMLLoader.

Содержимое снова отобразится при удалении проблемного значения свойства и активных фильтров.

К тому же, среда выполнения не может отображать содержимое PDF в прозрачных окнах. Среда выполнения отображает содержимое SWF, внедренное в страницу HTML, только когда параметр `wmode` объекта или тега внедрения принимает значение `opaque` или `transparent`. Так как для `wmode` по умолчанию используется значение `window`, то содержимое SWF не отображается в прозрачных окнах до принудительного изменения значения параметра wmode.

***Примечание.*** *В версиях до AIR 1.5.2 внедренное на страницу HTML содержимое SWF не отображалось, даже если параметр wmode принимал необходимое значение.*

Дополнительную информацию о загрузке подобного материала в HTMLLoader см. в разделах «Вложение SWF-содержимого в HTML» на странице 1067 и «Добавление PDF-содержимого в приложения AIR» на странице 592.

## Дополнительные свойства отображения

**Adobe AIR 1.0 и более поздних версий**

Класс HTMLLoader наследует несколько методов, которые можно использовать для специальных эффектов. В общем, использование этих эффектов для отображения HTMLLoader имеет некоторые ограничения, но они полезны для создания переходов и других временных эффектов. Например, при отображении диалогового окна для ввода данных пользователем можно применить размытие к отображению основного окна, пока пользователь не закроет диалоговое окно. Подобным образом можно применить к отображению затухание при закрытии окна.

В число дополнительных свойств входят следующие:

| Свойство | Ограничения |
|---|---|
| `alpha` | Может снизить удобочитаемость содержимого HTML |
| `filters` | В окне HTML внешние эффекты обрезаются краем окна |
| `graphics` | Фигуры, отрисованные при помощи графических команд, отображаются под содержимым HTML, включая фон по умолчанию. Свойство paintsDefaultBackground должно иметь значение false, чтобы отрисованные фигуры были видимы. |
| `opaqueBackground` | Не изменяет цвет фона по умолчанию. Свойство paintsDefaultBackground должно иметь значение false, чтобы этот слой цвета был видимым. |

| Свойство | Ограничения |
|---|---|
| `rotation` | Углы прямоугольной области HTMLLoader могут быть обрезаны краем окна. Содержимое SWF и PDF, загруженное в содержимое HTML, не отображается. |
| `scaleX`, `scaleY` | Изображение может выглядеть пикселизированным при масштабном коэффициенте выше 1. Содержимое SWF и PDF, загруженное в содержимое HTML, не отображается. |
| `transform` | Может снизить удобочитаемость содержимого HTML. Отображение HTML может быть обрезано краем окна. Содержимое SWF и PDF, загруженное в содержимое HTML, не отображается, если преобразование включает в себя поворот, масштабирование или наклон. |

Следующий пример иллюстрирует установку массива `filters` для размытия всего отображения HTML:

```
 var html:HTMLLoader = new HTMLLoader();
var urlReq:URLRequest = new URLRequest("http://www.adobe.com/");
html.load(urlReq);
html.width = 800;
html.height = 600;

var blur:BlurFilter = new BlurFilter(8);
var filters:Array = [blur];
html.filters = filters;
```

# Прокрутка содержимого HTML

**Adobe AIR 1.0 и более поздних версий**

Класс HTMLLoader включает в себя следующие свойства, которые помогают управлять прокруткой содержимого HTML:

| Свойство | Описание |
|---|---|
| `contentHeight` | Высота содержимого HTML в пикселах. |
| contentWidth | Ширина содержимого HTML в пикселах. |
| scrollH | Горизонтальное положение прокрутки содержимого HTML в объекте HTMLLoader. |
| scrollV | Вертикальное положение прокрутки содержимого HTML в объекте HTMLLoader. |

Следующий код устанавливает свойство `scrollV` так, что содержимое HTML прокручивается в нижнюю часть страницы:

```
 var html:HTMLLoader = new HTMLLoader();
html.addEventListener(Event.HTML_BOUNDS_CHANGE, scrollHTML);

const SIZE:Number = 600;
html.width = SIZE;
html.height = SIZE;

var urlReq:URLRequest = new URLRequest("http://www.adobe.com");
html.load(urlReq);
this.addChild(html);

function scrollHTML(event:Event):void
{
    html.scrollV = html.contentHeight - SIZE;
}
```

HTMLLoader не включает в себя горизонтальную и вертикальную полосы прокрутки. Полосы прокрутки можно выполнить в ActionScript или при помощи компонента Flex. Компонент Flex HTML автоматически включает в себя полосы прокрутки для содержимого HTML. Можно также использовать метод `HTMLLoader.createRootWindow()` для создания окна, содержащего объект HTMLLoader с полосами прокрутки (см. раздел «Создание окон с прокруткой содержимого HTML» на странице 1095).

# Доступ к истории HTML

**Adobe AIR 1.0 и более поздних версий**

По мере загрузки новых страниц в объект HTMLLoader среда выполнения составляет историю объекта. История соответствует объекту `window.history` на странице HTML. Класс HTMLLoader включает в себя следующие свойства и методы для работы с историей HTML:

| Член класса | Описание |
|---|---|
| `historyLength` | Общая длина истории, включая записи о предыдущих и последующих событиях. |
| `historyPosition` | Текущая позиция в списке истории. Элементы истории перед этой позицией представляют собой навигацию «назад», а элементы после этой позиции — навигацию «вперед». |
| `getHistoryAt()` | Возвращает объект URLRequest, соответствующий определенной записи истории в списке. |
| `historyBack()` | Если возможно, переходит назад по списку истории. |
| `historyForward()` | Если возможно, переходит вперед по хронологическому списку. |
| `historyGo()` | Выполняет переход на указанное количество шагов в истории обозревателя. Переходит вперед при положительном значении, и назад — при отрицательном. При значении 0 выполняется перезагрузка страницы. При указании позиции за пределами конца списка выполняется переход в конец списка. |

Элементы истории хранятся как объекты типа HTMLHistoryItem. Класс HTMLHistoryItem имеет следующие свойства:

| Свойство | Описание |
|---|---|
| `isPost` | Имеет значение `true`, если страница HTML содержит данные POST. |
| `originalUrl` | Исходный URL-адрес страницы HTML до перенаправлений. |
| `title` | Заголовок страницы HTML. |
| `url` | URL-адрес страницы HTML. |

# Задание агента пользователя для использования при загрузке содержимого HTML

**Adobe AIR 1.0 и более поздних версий**

Класс HTMLLoader имеет свойство `userAgent`, которое позволяет задать строку агента пользователя для использования объектом HTMLLoader. Свойство `userAgent` объекта HTMLLoader необходимо задать перед вызовом метода `load()`. Если это свойство задано для экземпляра HTMLLoader, то свойство `userAgent` объекта URLRequest, передаваемое методу `load()`, *не* используется.

Можно задать строку агента пользователя по умолчанию, используемую всеми объектами HTMLLoader в домене приложения, задав свойство `URLRequestDefaults.userAgent`. Статические свойства URLRequestDefaults используются по умолчанию для всех объектов URLRequest, а не только для объектов URLRequest, используемых в методе `load()` объектов HTMLLoader. Значение свойства `userAgent` объекта HTMLLoader переопределяет установку по умолчанию свойства `URLRequestDefaults.userAgent`.

Если значение агента пользователя не задано ни для свойства `userAgent` объекта HTMLLoader, ни для свойства `URLRequestDefaults.userAgent`, то используется значение агента пользователя, принятое в AIR по умолчанию. Это значение по умолчанию зависит от операционной системы (например, Mac OS или Windows), языка среды выполнения и ее версии, как в следующих двух примерах:

- `"Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/420+ (KHTML, like Gecko) AdobeAIR/1.0"`
- `"Mozilla/5.0 (Windows; U; en) AppleWebKit/420+ (KHTML, like Gecko) AdobeAIR/1.0"`

# Задание кодировки символов для содержимого HTML

**Adobe AIR 1.0 и более поздних версий**

Страница HTML может указывать на используемую ей кодировку символов при помощи тега `meta`, как в следующем примере:

```
meta http-equiv="content-type" content="text/html" charset="ISO-8859-1";
```

Измените настройку страницы так, чтобы использовалась конкретная кодировка символов, задав свойство `textEncodingOverride` объекта HTMLLoader:

```
var html:HTMLLoader = new HTMLLoader();
html.textEncodingOverride = "ISO-8859-1";
```

В случае, если страница HTML не содержит указание на кодировку, кодировку символов для содержимого объекта HTMLLoader можно задать при помощи свойства `textEncodingFallback` объекта HTMLLoader:

```
var html:HTMLLoader = new HTMLLoader();
html.textEncodingFallback = "ISO-8859-1";
```

Свойство `textEncodingOverride` переопределяет настройку страницы HTML. Свойство `textEncodingOverride` и настройка страницы HTML переопределяют значение свойства `textEncodingFallback`.

Свойство `textEncodingOverride` или свойство `textEncodingFallback` необходимо задать до загрузки содержимого HTML.

# Определение пользовательских интерфейсов, подобных интерфейсу обозревателя, для содержимого HTML

**Adobe AIR 1.0 и более поздних версий**

JavaScript содержит несколько API-интерфейсов для управления окном, в котором отображается содержимое HTML. В AIR эти API-интерфейсы можно переопределить путем реализации пользовательского класса HTMLHost.

## О расширении класса HTMLHost

**Adobe AIR 1.0 и более поздних версий**

Если, например, ваше приложение представляет несколько объектов HTMLLoader в интерфейсе с вкладками, можно сделать так, чтобы при загрузке страниц HTML изменялось название вкладки, а не заголовок основного окна. Более того, можно сделать так, чтобы код реагировал на вызов метода `window.moveTo()` изменением положения объекта HTMLLoader в контейнере родительского экранного объекта, перемещением окна, содержащего объект HTMLLoader, бездействием или выполнением какого-либо иного действия.

Класс HTMLHost в AIR управляет следующими свойствами и методами JavaScript:

- `window.status`
- `window.document.title`
- `window.location`
- `window.blur()`
- `window.close()`
- `window.focus()`
- `window.moveBy()`
- `window.moveTo()`
- `window.open()`

- `window.resizeBy()`
- `window.resizeTo()`

При создании объекта HTMLLoader с использованием `new HTMLLoader()` перечисленные свойства и методы JavaScript неактивны. Класс HTMLHost обеспечивает реализацию этих API-интерфейсов JavaScript подобно интерфейсу обозревателя по умолчанию. Для настройки вариантов поведения можно расширить класс HTMLHost. Для создания объекта HTMLHost, поддерживающего поведение по умолчанию, установите параметр `defaultBehaviors` на значение true в конструкторе HTMLHost:

```
var defaultHost:HTMLHost = new HTMLHost(true);
```

При создании окна HTML в AIR при помощи метода `createRootWindow()` класса HTMLLoader экземпляр HTMLHost, поддерживающий варианты поведения по умолчанию, назначается автоматически. Изменить поведение объекта узла можно путем присваивания другой реализации HTMLHost свойству `htmlHost` объекта HTMLLoader или путем присваивания значения `null` для отключения всех функций.

***Примечание.*** *AIR назначает объект HTMLHost по умолчанию исходному окну, созданному для приложения AIR на базе HTML, и всем окнам, созданным реализацией метода JavaScript `window.open()` по умолчанию.*

## Пример: расширение класса HTMLHost

**Adobe AIR 1.0 и более поздних версий**

Следующий пример показывает, как настроить способ влияния объекта HTMLLoader на пользовательский интерфейс путем расширения класса HTMLHost:

**Пример Flex:**

1. Создайте класс, расширяющий класс HTMLHost (подкласс).
2. Измените методы нового класса, при помощи которых будут обрабатываться изменения в настройках пользовательского интерфейса. Например, следующий класс, CustomHost, определяет варианты поведения при вызове метода `window.open()` и изменения `window.document.title`. Вызов метода `window.open()` открывает страницу HTML в новом окне, а изменения `window.document.title` (включая настройку элемента `<title>` страницы HTML) задают заголовок этого окна.

```
package
{
    import flash.html.*;
    import flash.display.StageScaleMode;
    import flash.display.NativeWindow;
    import flash.display.NativeWindowInitOptions;

    public class CustomHost extends HTMLHost
    {
        import flash.html.*;
        override public function
            createWindow(windowCreateOptions:HTMLWindowCreateOptions):HTMLLoader
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(windowCreateOptions.x,
                                                 windowCreateOptions.y,
                                                 windowCreateOptions.width,
                                                 windowCreateOptions.height);
            var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                         windowCreateOptions.scrollBarsVisible, bounds);
            htmlControl.htmlHost = new HTMLHostImplementation();
            if(windowCreateOptions.fullscreen){
                htmlControl.stage.displayState =
                    StageDisplayState.FULL_SCREEN_INTERACTIVE;
            }
            return htmlControl;
        }
        override public function updateTitle(title:String):void
        {
            htmlLoader.stage.nativeWindow.title = title;
        }
    }
}
```

**3** В коде, содержащем объект HTMLLoader (не в коде нового подкласса класса HTMLHost), создайте объект нового класса. Присвойте новый объект свойству `htmlHost` объекта HTMLLoader. В следующем коде Flex используется класс CustomHost, определенный в предыдущем шаге:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    applicationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.html.HTMLLoader;
            import CustomHost;
            private function init():void
            {
                var html:HTMLLoader = new HTMLLoader();
                html.width = container.width;
                html.height = container.height;
                var urlReq:URLRequest = new URLRequest("Test.html");
                html.htmlHost = new CustomHost();
                html.load(urlReq);
                container.addChild(html);
            }
        ]]>
    </mx:Script>
    <mx:UIComponent id="container" width="100%" height="100%"/>
</mx:WindowedApplication>
```

Для проверки описанного здесь кода вложите файл HTML со следующим содержимым в каталог приложения:

```
<html>
    <head>
        <title>Test</title>
    </head>
    <script>
        function openWindow()
        {
            window.runtime.trace("in");
            document.title = "foo"
            window.open('Test.html');
            window.runtime.trace("out");
        }
    </script>
    <body>
        <a href="#" onclick="openWindow()">window.open('Test.html')</a>
    </body>
</html>
```

**Пример Flash Professional.**

1 Создайте файл Flash для AIR. Задайте класс документа CustomHostExample и сохраните файл как CustomHostExample.fla.

2 Создайте файл ActionScript с именем CustomHost.as, содержащий класс, расширяющий класс HTMLHost (подкласс). Этот класс переопределяет некоторые методы нового класса, при помощи которых обрабатываются изменения настроек пользовательского интерфейса. Например, следующий класс, CustomHost, определяет варианты поведения при вызове метода `window.open()` и изменения `window.document.title`. Вызов метода `window.open()` открывает страницу HTML в новом окне, а изменения свойства `window.document.title` (включая настройку элемента `<title>` страницы HTML) задают заголовок этого окна.

```
package
{
    import flash.display.StageScaleMode;
    import flash.display.NativeWindow;
    import flash.display.NativeWindowInitOptions;
    import flash.events.Event;
    import flash.events.NativeWindowBoundsEvent;
    import flash.geom.Rectangle;
    import flash.html.HTMLLoader;
    import flash.html.HTMLHost;
    import flash.html.HTMLWindowCreateOptions;
    import flash.text.TextField;

    public class CustomHost extends HTMLHost
    {
        public var statusField:TextField;

        public function CustomHost(defaultBehaviors:Boolean=true)
        {
            super(defaultBehaviors);
        }
        override public function windowClose():void
        {
            htmlLoader.stage.nativeWindow.close();
        }
        override public function createWindow(
                                windowCreateOptions:HTMLWindowCreateOptions ):HTMLLoader
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(windowCreateOptions.x,
                                        windowCreateOptions.y,
                                        windowCreateOptions.width,
                                        windowCreateOptions.height);
            var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                    windowCreateOptions.scrollBarsVisible, bounds);
            htmlControl.htmlHost = new HTMLHostImplementation();
            if(windowCreateOptions.fullscreen){
                htmlControl.stage.displayState =
                    StageDisplayState.FULL_SCREEN_INTERACTIVE;
            }
            return htmlControl;
        }
        override public function updateLocation(locationURL:String):void
        {
            trace(locationURL);
        }
        override public function set windowRect(value:Rectangle):void
        {
            htmlLoader.stage.nativeWindow.bounds = value;
```

```
        }
        override public function updateStatus(status:String):void
        {
            statusField.text = status;
            trace(status);
        }
        override public function updateTitle(title:String):void
        {
            htmlLoader.stage.nativeWindow.title = title + "- Example Application";
        }
        override public function windowBlur():void
        {
            htmlLoader.alpha = 0.5;
        }
        override public function windowFocus():void
        {
            htmlLoader.alpha = 1;
        }
    }
}
```

**3** Создайте другой файл ActionScript с именем CustomHostExample.as, который будет содержать класс документа приложения. Этот класс создает объект HTMLLoader и устанавливает его свойство host на экземпляр класса CustomHost, определенный в предыдущем шаге:

```
package
{
    import flash.display.Sprite;
    import flash.html.HTMLLoader;
    import flash.net.URLRequest;
    import flash.text.TextField;

    public class CustomHostExample extends Sprite
    {
        function CustomHostExample():void
        {
            var html:HTMLLoader = new HTMLLoader();
            html.width = 550;
            html.height = 380;
            var host:CustomHost = new CustomHost();
            html.htmlHost = host;

            var urlReq:URLRequest = new URLRequest("Test.html");
            html.load(urlReq);

            addChild(html);

            var statusTxt:TextField = new TextField();
            statusTxt.y = 380;
            statusTxt.height = 20;
            statusTxt.width = 550;
            statusTxt.background = true;
            statusTxt.backgroundColor = 0xEEEEEEEE;
            addChild(statusTxt);

            host.statusField = statusTxt;
        }
    }
}
```

Для проверки описанного здесь кода вложите файл HTML со следующим содержимым в каталог приложения:

```
<html>
    <head>
    <title>Test</title>
    <script>
    function openWindow()
    {
    document.title = "Test"
    window.open('Test.html');
    }
    </script>
    </head>
    <body bgColor="#EEEEEE">
    <a href="#" onclick="window.open('Test.html')">window.open('Test.html')</a>
    <br/><a href="#" onclick="window.document.location='http://www.adobe.com'">
    window.document.location = 'http://www.adobe.com'</a>
    <br/><a href="#" onclick="window.moveBy(6, 12)">moveBy(6, 12)</a>
    <br/><a href="#" onclick="window.close()">window.close()</a>
    <br/><a href="#" onclick="window.blur()">window.blur()</a>
    <br/><a href="#" onclick="window.focus()">window.focus()</a>
    <br/><a href="#" onclick="window.status = new Date().toString()">window.status=new
Date().toString()</a>
    </body>
</html>
```

## Обработка изменений свойства window.location

**Adobe AIR 1.0 и более поздних версий**

Измените метод `locationChange()` для обработки изменений URL-адреса страницы HTML. Метод `locationChange()` вызывается, когда сценарий JavaScript страницы изменяет значение свойства `window.location`. В следующем примере загружается запрошенный URL-адрес:

```
override public function updateLocation(locationURL:String):void
{
    htmlLoader.load(new URLRequest(locationURL));
}
```

***Примечание.*** *При помощи свойства htmlLoader объекта HTMLHost можно создать ссылку на текущий объект HTMLLoader.*

## Обработка вызовов сценарием JavaScript методов window.moveBy(), window.moveTo(), window.resizeTo(), window.resizeBy()

**Adobe AIR 1.0 и более поздних версий**

Измените метод `set windowRect()` для обработки изменений границ содержимого HTML. Метод `set windowRect()` вызывается, когда сценарий JavaScript страницы вызывает метод `window.moveBy()`, `window.moveTo()`, `window.resizeTo()` или `window.resizeBy()`. В следующем примере обновляются границы окна рабочего стола:

```
override public function set windowRect(value:Rectangle):void
{
    htmlLoader.stage.nativeWindow.bounds = value;
}
```

## Обработка вызовов сценарием JavaScript метода window.open()

**Adobe AIR 1.0 и более поздних версий**

Измените метод `createWindow()` для обработки вызовов сценарием JavaScript метода `window.open()`. Реализация метода `createWindow()` отвечает за создание и возвращение нового объекта HTMLLoader. Как правило, объект HTMLLoader отображается в новом окне, но создавать окно необязательно.

Следующий пример иллюстрирует реализацию функции `createWindow()` с использованием `HTMLLoader.createRootWindow()` для создания как окна, так и объекта HTMLLoader. Также можно отдельно создать объект NativeWindow и добавить объект HTMLLoader в рабочую область окна.

```
override public function createWindow(windowCreateOptions:HTMLWindowCreateOptions):HTMLLoader{
    var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
    var bounds:Rectangle = new Rectangle(windowCreateOptions.x, windowCreateOptions.y,
                                 windowCreateOptions.width, windowCreateOptions.height);
    var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                     windowCreateOptions.scrollBarsVisible, bounds);
    htmlControl.htmlHost = new HTMLHostImplementation();
    if(windowCreateOptions.fullscreen){
        htmlControl.stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
    }
    return htmlControl;
}
```

***Примечание.*** *В данном примере пользовательская реализация HTMLHost назначается всем новым окнам, созданным при помощи метода `window.open()`. Также можно использовать другую реализацию или установить свойство `htmlHost` на значение null для новых окон, если необходимо.*

Объект, передаваемый в качестве параметра методу `createWindow()` является объектом HTMLWindowCreateOptions. Класс HTMLWindowCreateOptions содержит свойства, передающие значения, заданные строкой параметра `features` в вызове метода `window.open()`:

| свойство HTMLWindowCreateOptions | Соответствующие настройки строки features в вызове сценарием JavaScript метода window.open() |
|---|---|
| `fullscreen` | `fullscreen` |
| `height` | `height` |
| `locationBarVisible` | `location` |
| `menuBarVisible` | `menubar` |
| `resizeable` | `resizable` |
| `scrollBarsVisible` | `scrollbars` |
| `statusBarVisible` | `status` |
| `toolBarVisible` | `toolbar` |
| `width` | `width` |
| `x` | `left` или `screenX` |
| `y` | `top` или `screenY` |

Класс HTMLLoader не реализует все функции, которые могут быть указаны в строке feature. При необходимости приложение должно обеспечивать наличие полос прокрутки, строк размещения, строк меню, строк состояния и панелей инструментов.

Другие аргументы метода `window.open()` в JavaScript обрабатываются системой. Реализация метода `createWindow()` не должна загружать содержимое в объект HTMLLoader или задавать заголовок окна.

## Обработка вызовов сценарием JavaScript метода window.close()

**Adobe AIR 1.0 и более поздних версий**

Измените метод `windowClose()` для обработки вызовов сценарием JavaScript метода `window.close()`. Следующий пример закрывает окно рабочего стола при вызове метода `window.close()`:

```
override public function windowClose():void
{
    htmlLoader.stage.nativeWindow.close();
}
```

Вызовы сценарием JavaScript метода `window.close()` не должны закрывать окно содержимого. Можно, например, удалить объект HTMLLoader из списка отображения, оставив окно (возможно, с другим содержимым) открытым, как в следующем коде:

```
override public function windowClose():void
{
    htmlLoader.parent.removeChild(htmlLoader);
}
```

## Обработка изменений свойства window.status

**Adobe AIR 1.0 и более поздних версий**

Измените метод `updateStatus()` для обработки изменений сценарием JavaScript свойства `window.status`. В следующем примере отслеживается значение состояния:

```
 override public function updateStatus(status:String):void
{
    trace(status);
}
```

Запрошенное состояние передается в виде строки методу `updateStatus()`.

Объект HTMLLoader не предоставляет строку состояния.

## Обработка изменений свойства window.document.title

**Adobe AIR 1.0 и более поздних версий**

Измените метод `updateTitle()` для обработки изменений сценарием JavaScript значения свойства `window.document.title`. В следующем примере изменяется заголовок окна с добавлением в его конец строки «Sample»:

```
 override public function updateTitle(title:String):void
{
    htmlLoader.stage.nativeWindow.title = title + " - Sample";
}
```

Если свойство `document.title` задано для страницы HTML, запрошенный заголовок передается в виде строки методу `updateTitle()`.

Изменение свойства `document.title` не должно приводить к изменению заголовка окна, содержащего объект HTMLLoader. Можно изменить другой элемент интерфейса, например текстовое поле.

## Обработка вызовов сценарием JavaScript методов window.blur() и window.focus()

**Adobe AIR 1.0 и более поздних версий**

Измените методы `windowBlur()` и `windowFocus()` для обработки вызовов сценарием JavaScript методов `window.blur()` и `window.focus()`, как показано в следующем примере:

```
 override public function windowBlur():void
{
    htmlLoader.alpha = 0.5;
}
override public function windowFocus():void
{
    htmlLoader.alpha = 1.0;
    NativeApplication.nativeApplication.activate(htmlLoader.stage.nativeWindow);
}
```

***Примечание.*** *AIR не предоставляет API-интерфейс для деактивации окна или приложения.*

## Создание окон с прокруткой содержимого HTML

**Adobe AIR 1.0 и более поздних версий**

Класс HTMLLoader содержит статический метод, `HTMLLoader.createRootWindow()`, который позволяет открыть новое окно (представленное объектом NativeWindow), содержащее объект HTMLLoader, а также задать некоторые настройки пользовательского интерфейса для этого окна. Метод использует четыре параметра, определяющие пользовательский интерфейс:

| Параметр | Описание |
|---|---|
| `visible` | Логическое значение, указывающее на то, является окно исходно видимым (`true`) или нет (`false`). |
| `windowInitOptions` | Объект NativeWindowInitOptions. Класс NativeWindowInitOptions определяет параметры инициализации объекта NativeWindow, включая следующие: можно ли сворачивать, разворачивать окно или изменять его размер, имеет окно системную или пользовательскую рамку, является ли окно прозрачным (для окон, которые не используют системную рамку), а также тип окна. |
| `scrollBarsVisible` | Имеются в наличии полосы прокрутки (`true`) или нет (`false`). |
| `bounds` | Объект Rectangle, определяющий положение и размер нового окна. |

Например, в следующем коде используется метод `HTMLLoader.createRootWindow()` для создания окна с содержимым HTMLLoader и полосами прокрутки:

```
 var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
var bounds:Rectangle = new Rectangle(10, 10, 600, 400);
var html2:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions, true, bounds);
var urlReq2:URLRequest = new URLRequest("http://www.example.com");
html2.load(urlReq2);
html2.stage.nativeWindow.activate();
```

***Примечание.*** *Окна, созданные путем вызова метода* `createRootWindow()` *непосредственно в JavaScript, не зависят от открывающего окна HTML. Свойства JavaScript Window* `opener` *и* `parent`*, например, имеют значение* `null`*. Однако при косвенном вызове метода* `createRootWindow()` *путем изменения метода HTMLHost* `createWindow()` *на вызов метода* `createRootWindow()` *свойства* `opener` *и* `parent` *содержат ссылку на открывающее окно HTML.*

# Создание подклассов класса HTMLLoader

**Adobe AIR 1.0 и более поздних версий**

Можно создать подкласс класса HTMLLoader для создания новых вариантов поведения. Например, можно создать подкласс, определяющий прослушивателей событий HTMLLoader по умолчанию (событий, которые отправляются при визуализации содержимого HTML или при нажатии пользователем на ссылку).

В следующем примере класс HTMLHost расширяется для обеспечения поведения *normal* при вызове метода JavaScript `window.open()`. В этом примере затем определяется подкласс класса HTMLLoader, использующий класс пользовательской реализации HTMLHost:

```
 package
{
       import flash.html.HTMLLoader;
    public class MyHTMLHost extends HTMLHost
       {
           public function MyHTMLHost()
           {
               super(false);
        }
        override public function createWindow(opts:HTMLWindowCreateOptions):void
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(opts.x, opts.y, opts.width, opts.height);
            var html:HTMLLoader = HTMLLoader.createRootWindow(true,
                                                  initOptions,
                                                  opts.scrollBarsVisible,
                                                  bounds);
            html.stage.nativeWindow.orderToFront();
            return html
       }
}
```

Следующий код определяет подкласс класса HTMLLoader, который присваивает объект MyHTMLHost его свойству `htmlHost`:

```
package
{
       import flash.html.HTMLLoader;
    import MyHTMLHost;
    import HTMLLoader;
       public class MyHTML extends HTMLLoader
       {
           public function MyHTML()
           {
               super();
               htmlHost = new MyHTMLHost();
           }
       }
}
```

Подробные сведения о классе HTMLHost и методе `HTMLLoader.createRootWindow()`, использованных в данном примере, см. в разделе «Определение пользовательских интерфейсов, подобных интерфейсу обозревателя, для содержимого HTML» на странице 1085.

# Глава 61. Обработка в AIR событий HTML

**Adobe AIR 1.0 и более поздних версий**

Система обработки событий облегчает для программистов процедуру ответа на действия пользователя и события системы. Модель событий Adobe® AIR® не только удобна, но и соответствует принятым стандартам. Модель событий, основанная на Спецификации событий DOM 3-го уровня — архитектуре обработки событий, принятой в отрасли, — предоставляет эффективный и несложный инструмент обработки событий для программистов.

## События HTMLLoader

**Adobe AIR 1.0 и более поздних версий**

Объект HTMLLoader отправляет следующие события Adobe® ActionScript® 3.0:

| Событие | Описание |
|---|---|
| `htmlDOMInitialize` | Отправляется после создания HTML-документа, но до разбора каких-либо сценариев или добавления к странице узлов DOM. |
| `complete` | Отправляется при создании модели DOM HTML в ответ на загрузку, сразу после выполнения события `onload` на HTML-странице. |
| `htmlBoundsChanged` | Отправляется при изменении одного из свойств `contentWidth` и `contentHeight` или обоих этих свойств. |
| `locationChange` | Отправляется при изменении свойства location объекта HTMLLoader. |
| `locationChanging` | Отправляется перед изменением расположения HTMLLoader в связи с навигацией пользователя, вызовом сценария JavaScript или перенаправлением. Событие `locationChanging` не отправляется при вызове методов `load()`, `loadString()`,`reload()`,`historyGo()`,`historyForward()` или `historyBack()`.<br><br>Вызов метода `preventDefault()` отправленного объекта события отменяет навигацию.<br><br>Если ссылка открыта в системном браузере, событие locationChanging не отправляется, поскольку HTMLLoader не изменяет расположение. |
| `scroll` | Отправляется при любом изменении положения прокрутки в HTML. События scroll могут быть вызваны переходами по якорным ссылкам (содержащим символ #) на странице или в результате вызова метода `window.scrollTo()`. Ввод текста в текстовое поле или область также может вызывать событие scroll. |
| `uncaughtScriptException` | Отправляется при генерации в объекте HTMLLoader исключения JavaScript, которое не обнаруживается кодом JavaScript. |

Также можно зарегистрировать функцию ActionScript для события JavaScript (например, `onClick`). Дополнительные сведения см. в разделе «Обработка событий DOM с помощью ActionScript» на странице 1099.

# Обработка событий DOM с помощью ActionScript

**Adobe AIR 1.0 и более поздних версий**

Функции ActionScript могут быть зарегистрированы для ответа на события JavaScript. Например, рассмотрим следующее HTML-содержимое:

```
<html>
<body>
    <a href="#" id="testLink">Click me.</a>
</html>
```

Функция ActionScript может быть зарегистрирована в качестве обработчика любого события на этой странице. Например, в коде ниже функция `clickHandler()` добавляется на HTML-страницу в качестве прослушивателя события `onclick` элемента `testLink`:

```
var html:HTMLLoader = new HTMLLoader( );
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.addEventListener(Event.COMPLETE, completeHandler);

function completeHandler(event:Event):void {
    html.window.document.getElementById("testLink").onclick = clickHandler;
}

function clickHandler( event:Object ):void {
    trace("Event of type: " + event.type );
}
```

Отправляемый объект события не является типом flash.events.Event или одним из подклассов Event. Используйте класс Object для объявления типа аргумента функции обработчика событий.

Для регистрации прослушивателей этих событий также может использоваться метод `addEventListener()`. Например, метод `completeHandler()` из предыдущего примера может быть заменен:

```
function completeHandler(event:Event):void {
    var testLink:Object = html.window.document.getElementById("testLink");
    testLink.addEventListener("click", clickHandler);
}
```

Когда прослушиватель обращается к какому-либо элементу DOM, рекомендуется дождаться, пока родительский объект HTMLLoader отправит событие `complete`, и только после этого добавлять прослушиватели. HTML-страницы часто загружают несколько файлов, а модель DOM в HTML не может быть полностью построена, пока все файлы не будут загружены и разобраны. Объект HTMLLoader отправляет событие `complete`, когда все элементы уже созданы.

# Ответ на необрабатываемые исключения JavaScript

**Adobe AIR 1.0 и более поздних версий**

Рассмотрим следующий HTML-документ:

```
<html>
<head>
    <script>
        function throwError() {
            var x = 400 * melbaToast;
        }
    </script>
</head>
<body>
    <a href="#" onclick="throwError()">Click me.</a>
</html>
```

Он содержит функцию JavaScript `throwError()`, обращающуюся к неизвестной переменной `melbaToast`:

```
var x = 400 * melbaToast;
```

Когда операция JavaScript сталкивается с недопустимой операцией, которую код JavaScript не обрабатывает с помощью структуры `try/catch`, объект HTMLLoader, в котором содержится страница, отправляет событие HTMLUncaughtScriptExceptionEvent. Для этого события можно зарегистрировать прослушиватель, как показано ниже:

```
var html:HTMLLoader = new HTMLLoader();
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.width = container.width;
html.height = container.height;
container.addChild(html);
html.addEventListener(HTMLUncaughtScriptExceptionEvent.UNCAUGHT_SCRIPT_EXCEPTION,
                          htmlErrorHandler);
function htmlErrorHandler(event:HTMLUncaughtJavaScriptExceptionEvent):void
{
    event.preventDefault();
    trace("exceptionValue:", event.exceptionValue)
    for (var i:int = 0; i < event.stackTrace.length; i++)
    {
        trace("sourceURL:", event.stackTrace[i].sourceURL);
        trace("line:", event.stackTrace[i].line);
        trace("function:", event.stackTrace[i].functionName);
    }
}
```

В JavaScript можно обрабатывать одно и то же событие с помощью свойства window.htmlLoader:

```
<html>
<head>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>

    <script>
        function throwError() {
            var x = 400 * melbaToast;
        }

        function htmlErrorHandler(event) {
            event.preventDefault();
            var message = "exceptionValue:" + event.exceptionValue + "\n";
            for (var i = 0; i < event.stackTrace.length; i++){
                message += "sourceURL:" + event.stackTrace[i].sourceURL +"\n";
                message += "line:" + event.stackTrace[i].line +"\n";
                message += "function:" + event.stackTrace[i].functionName + "\n";
            }
            alert(message);
        }

        window.htmlLoader.addEventListener("uncaughtScriptException", htmlErrorHandler);
    </script>
</head>
<body>
    <a href="#" onclick="throwError()">Click me.</a>
</html>
```

Обработчик события `htmlErrorHandler()` отменяет поведение события по умолчанию (т. е. отправку сообщения об ошибке JavaScript в выводе трассировочных данных AIR) и генерирует собственное сообщение об ошибке вывода. Выводится значение `exceptionValue` объекта HTMLUncaughtScriptExceptionEvent. Выводятся свойства каждого объекта в массиве `stackTrace`:

```
exceptionValue: ReferenceError: Can't find variable: melbaToast
sourceURL: app:/test.html
line: 5
function: throwError
sourceURL: app:/test.html
line: 10
function: onclick
```

# Обработка событий среды выполнения с помощью JavaScript

**Adobe AIR 1.0 и более поздних версий**

Классы среды выполнения позволяют добавлять обработчики событий методом `addEventListener()`. Для добавления функции обработчика события вызовите метод `addEventListener()` объекта, который отправляет это событие, указав тип события и функцию обработчика. Например, чтобы прослушивать событие `closing`, отправляемое по щелчку мыши на значке закрытия окна, воспользуйтесь следующей инструкцией:

```
window.nativeWindow.addEventListener(air.NativeWindow.CLOSING, handleWindowClosing);
```

## Создание функции обработчика события

**Adobe AIR 1.0 и более поздних версий**

Ниже показан код, в котором создается простой HTML-файл, отображающий информацию о положении главного окна. Функция обработчика `moveHandler()` прослушивает событие move (определено в классе NativeWindowBoundsEvent) главного окна.

```
<html>
    <script src="AIRAliases.js" />
    <script>
        function init() {
            writeValues();
            window.nativeWindow.addEventListener(air.NativeWindowBoundsEvent.MOVE,
                                                 moveHandler);
        }
        function writeValues() {
            document.getElementById("xText").value = window.nativeWindow.x;
            document.getElementById("yText").value = window.nativeWindow.y;
        }
        function moveHandler(event) {
            air.trace(event.type); // move
            writeValues();
        }
    </script>
    <body onload="init()" />
        <table>
            <tr>
                <td>Window X:</td>
                <td><textarea id="xText"></textarea></td>
            </tr>
            <tr>
                <td>Window Y:</td>
                <td><textarea id="yText"></textarea></td>
            </tr>
        </table>
    </body>
</html>
```

Когда пользователь смещает окно, элементы textarea отображают новые положения по осям X и Y для этого окна:

Обратите внимание, что объект события передается методу `moveHandler()` в качестве аргумента. Параметр события позволяет функции обработчика исследовать объект события. В этом примере свойство `type` объекта события указывает, что событие относится к типу `move`.

## Удаление прослушивателей событий

**Adobe AIR 1.0 и более поздних версий**

Метод `removeEventListener()` можно использовать для удаления неиспользуемых прослушивателей событий. Ненужные прослушиватели рекомендуется удалять. Обязательными являются параметры `eventName` и `listener` — как и для метода `addEventListener()`.

## Удаление прослушивателей событий на HTML-страницах с навигацией

**Adobe AIR 1.0 и более поздних версий**

При навигации по HTML-содержимому или его сбрасывании из-за закрытия содержащего его окна прослушиватели событий, указывающие на объекты на незагруженной странице, не удаляются автоматически. Когда объект отправляет событие уже выгруженному обработчику, появляется следующее сообщение об ошибке: «Приложение ссылается на объект JavaScript на HTML-странице, которая более не загружена».

Чтобы эта ошибка не возникала, удалите прослушиватели событий JavaScript с HTML-страницы, прежде чем она будет закрыта. При навигации по странице (в объекте HTMLLoader) удалите прослушиватель во время события `unload` объекта `window`.

Например, показанный ниже код JavaScript удаляет прослушиватель события `uncaughtScriptException`:

```
window.onunload = cleanup;
window.htmlLoader.addEventListener('uncaughtScriptException', uncaughtScriptException);
function cleanup()
{
    window.htmlLoader.removeEventListener('uncaughtScriptException',
                            uncaughtScriptExceptionHandler);
}
```

Чтобы эта ошибка не возникала при закрытии окон с HTML-содержимым, вызовите функцию cleanup в ответ на событие `closing` объекта NativeWindow (`window.nativeWindow`). Например, показанный ниже код JavaScript удаляет прослушиватель события `uncaughtScriptException`:

```
 window.nativeWindow.addEventListener(air.Event.CLOSING, cleanup);
function cleanup()
{
    window.htmlLoader.removeEventListener('uncaughtScriptException',
                            uncaughtScriptExceptionHandler);
}
```

Вы также можете предупредить эту ошибку удалением прослушивателя события, как только он запустится (если событие требует лишь разовой обработки). Например, в следующем коде JavaScript окно HTML создается в результате вызова метода `createRootWindow()` класса HTMLLoader и добавления прослушивателя события `complete`. При вызове обработчика события `complete` он удаляет собственный прослушиватель с помощью функции `removeEventListener()`:

```
 var html = runtime.flash.html.HTMLLoader.createRootWindow(true);
html.addEventListener('complete', htmlCompleteListener);
function htmlCompleteListener()
{
    html.removeEventListener(complete, arguments.callee)
    // handler code..
}
html.load(new runtime.flash.net.URLRequest("second.html"));
```

Удаление ненужных прослушивателей событий также позволяет системному сборщику мусора восстанавливать память, занятую этими прослушивателями.

## Проверка существующих прослушивателей событий

**Adobe AIR 1.0 и более поздних версий**

Метод `hasEventListener()` позволяет проверить, зарегистрирован ли для объекта прослушиватель.

# Глава 62. Отображение HTML-содержимого в мобильных приложениях

**Adobe AIR 2.5 и более новых версий**

Класс StageWebView при отображении HTML-содержимого использует механизм управления системного браузера на мобильных устройствах и стандартный механизм управления Adobe® AIR® HTMLLoader в настольных системах. Проверьте свойство `StageWebView.isSupported`, чтобы определить, поддерживается ли класс на определенном устройстве. Поддержка не гарантирована для всех устройств в мобильном профиле.

Во всех профилях класс StageWebView поддерживает только ограниченный набор взаимодействий между HTML-содержимым и остальной частью приложения. Вы можете управлять навигацией, но взаимодействие между сценариями и прямой обмен данными не разрешены. Содержимое может быть загружено по локальному или удаленному URL, а также передано в строку HTML.

## Объекты StageWebView

Объект StageWebView не является экранным объектом и не может быть добавлен в список отображения. Вместо этого он выполняет функции окна просмотра, напрямую связанного с рабочей областью. Содержимое StageWebView отрисовывается поверх любого содержимого списка отображения. Способ управления порядком отрисовки нескольких объектов StageWebView не предусмотрен.

Чтобы отобразить объект StageWebView, используя свойство `stage` объекта StageWebView, задайте рабочую область, в которой появится объект. С помощью свойства `viewPort` установите размер отображения.

В свойстве `viewPort` задайте координаты x и y в диапазоне от -8192 до 8191. Максимальное значение ширины и высоты рабочей области составляет 8191. Если размер превышает максимальное значение, будет сгенерировано исключение.

Следующий пример создает объект StageWebView, задает свойства `stage` и `viewPort` и отображает строку HTML:

```
var webView:StageWebView = new StageWebView();
webView.viewPort = new Rectangle( 0, 0, this.stage.stageWidth, this .stage.stageHeight);
webView.stage = this.stage;
var htmlString:String = "<!DOCTYPE HTML>" +
                        "<html><body>" +
                        "<p>King Philip could order five good steaks.</p>" +
                        "</body></html>";
webView.loadString( htmlString );
```

Чтобы скрыть объект StageWebView, для свойства `stage` установите значение `null`. Для полного удаления объекта вызовите метод `dispose()`. Вызов метода `dispose()` не является обязательным, но он помогает сборщику мусора быстрее восстановить память, использовавшуюся объектом.

# Содержимое

Содержимое в объект StageWebView можно загрузить с помощью двух методов: `loadURL()` и `loadString()`.

Метод `loadURL()` загружает ресурс по указанному URL. Можно использовать любую схему URI, поддерживаемую механизмом управления системного браузера, включая: data:, file:, http:, https:, и javascript:. Схемы app: и app-storage: не поддерживаются. AIR не проверяет строку URL.

Метод `loadString()` загружает строковый литерал, содержащий HTML-содержимое. Расположение страницы, загруженной с помощью этого метода, задается следующим образом:

- На персональном компьютере: about:blank
- В iOS: *htmlString*
- В Android: формат данных URI закодированного свойства *htmlString*

Схема URI определяет правила загрузки встроенного содержимого или данных.

| Схема URI | Загрузка локального ресурса | Загрузка удаленного ресурса | Local XMLHttpRequest | Remote XMLHttpRequest |
|---|---|---|---|---|
| data: | Нет | Да | Нет | Нет |
| file: | Да | Да | Да | Да |
| http:, https: | Нет | Да | Нет | Один и тот же домен |
| about: (loadString() method) | Нет | Да | Нет | Нет |

***Примечание.*** *Если для свойства рабочей области `displayState` установлено значение `FULL_SCREEN`, на персональном компьютере вы не сможете выполнять ввод в текстовое поле, которое отображается в объекте StageWebView. Однако в iOS и Android ввод в текстовое поле в StageWebView можно выполнять, даже если для события рабочей области `displayState` установлено значение `FULL_SCREEN`.*

В следующем примере объект StageWebView используется для отображения веб-сайта Adobe.

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;

    public class StageWebViewExample extends MovieClip{

        var webView:StageWebView = new StageWebView();

        public function StageWebViewExample() {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadURL( "http://www.adobe.com" );
        }
    }
}
```

На устройствах Android необходимо задать разрешение Android INTERNET, чтобы приложения могли загружать удаленные ресурсы.

В ОС Android 3.0 и более поздних версий приложение должно включать аппаратное ускорение в элементе manifestAdditions Android дескриптора приложения AIR для отображения содержимого подключаемого модуля в объекте StageWebView. См. раздел «Включение Flash Player и других подключаемых модулей в объекте StageWebView».

## JavaScript URI

Можно использовать JavaScript URI для вызова функции, определенной на странице HTML, которая загружается объектом StageWebView. Функция, вызываемая с помощью JavaScript URI, выполняется в контексте загружаемой веб-страницы. В следующем примере объект StageWebView используется для вызова функции JavaScript:

```
package {
    import flash.display.*;
    import flash.geom.Rectangle;
    import flash.media.StageWebView;
    public class WebView extends Sprite
    {
        public var webView:StageWebView = new StageWebView();
        public function WebView()
        {
            var htmlString:String = "<!DOCTYPE HTML>" +
            "<html><script type=text/javascript>" +
            "function callURI(){" +
            "alert(\"You clicked me!!\");"+
            "}</script><body>" +
            "<p><a href=javascript:callURI()>Click Me</a></p>" +
            "</body></html>";
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadString( htmlString );
        }
    }
}
```

### Дополнительные разделы справки

Шон Войзен (Sean Voisen): Достижение максимальных результатов при использовании StageWebView

# События навигации

Когда пользователь нажимает на ссылку в HTML, объект StageWebView отправляет событие `locationChanging`. Можно вызывать метод `preventDefault()` объекта события, чтобы остановить навигацию. В противном случае объект StageWebView переходит на новую страницу и отправляет событие `locationChange`. После завершения загрузки страницы StageWebView отправляет событие `complete`.

Событие `locationChanging` отправляется каждый раз при перенаправлении HTML. События `locationChange` и `complete` отправляются в соответствующий момент времени.

В iOS событие `locationChanging` отправляется перед событием `locationChange`, за исключением методов `loadURL()` и `loadString()`. Событие `locationChange` также отправляется при изменении навигации через фреймы и встроенные фреймы.

Следующий пример показывает, как можно предотвратить изменение местоположения и вместо этого открыть новую страницу в системном браузере.

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.events.LocationChangeEvent;
    import flash.geom.Rectangle;
    import flash.net.navigateToURL;
    import flash.net.URLRequest;

    public class StageWebViewNavEvents extends MovieClip{
        var webView:StageWebView = new StageWebView();

        public function StageWebViewNavEvents() {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
        webView.addEventListener( LocationChangeEvent.LOCATION_CHANGING, onLocationChanging );
            webView.loadURL( "http://www.adobe.com" );
        }
        private function onLocationChanging( event:LocationChangeEvent ):void
        {
            event.preventDefault();
            navigateToURL( new URLRequest( event.location ) );
        }
    }
}
```

# История

Когда пользователь нажимает на ссылки в содержимом, отображаемые в объекте StageWebView, механизм управления сохраняет стеки истории переходов назад и вперед. Следующий пример демонстрирует навигацию по двум стекам истории переходов. В примере используются программные клавиши «Назад» и «Поиск».

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;

    public class StageWebViewExample extends MovieClip{

        var webView:StageWebView = new StageWebView();

        public function StageWebViewExample()
        {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadURL( "http://www.adobe.com" );

            stage.addEventListener( KeyboardEvent.KEY_DOWN, onKey );
        }

        private function onKey( event:KeyboardEvent ):void
        {
            if( event.keyCode == Keyboard.BACK && webView.isHistoryBackEnabled )
            {
                trace("back");
                webView.historyBack();
                event.preventDefault();
            }
            if( event.keyCode == Keyboard.SEARCH && webView.isHistoryForwardEnabled )
            {
                trace("forward");
                webView.historyForward();
            }
        }
    }
}
```

# Фокус

Несмотря на то что класс StageWebView не является отображаемым объектом, он содержит членов, которые позволят управлять контролем над переходами фокуса.

Когда объект StageWebView получает фокус, он отправляет событие `focusIn`. Используйте это событие для управления элементами фокуса в приложении, если это необходимо.

Когда объект StageWebView теряет фокус, он отправляет событие `focusOut`. Экземпляр StageWebView может потерять фокус, когда пользователь переключается за пределы первого или последнего элемента управления на веб-страницы с помощью трекбола или клавиш со стрелками. Свойство `direction` объекта события позволяет узнать, поднимается ли фокус выше верхней части страницы или опускается ниже нижней части страницы. Используя эту информацию, назначьте фокус соответствующему отображаемому объекту выше или ниже StageWebView.

В ОС iOS программная установка фокуса невозможна. StageWebView отправляет события `focusIn` и `focusOut` со свойствам направления `FocusEvent`, для которого установлено значение `none`. Если пользователь нажимает внутри объекта StageWebView, отправляется событие `focusIn`. Если пользователь нажимает за пределами объекта StageWebView, отправляется событие `focusOut`.

Следующий пример иллюстрирует переход фокуса от объекта StageWebView к отображаемым объектам Flash:

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;
    import flash.text.TextField;
    import flash.text.TextFieldType;
    import flash.events.FocusEvent;
    import flash.display.FocusDirection;
    import flash.events.LocationChangeEvent;

    public class StageWebViewFocusEvents extends MovieClip{
        var webView:StageWebView = new StageWebView();
        var topControl:TextField = new TextField();
        var bottomControl:TextField = new TextField();

        public function StageWebViewFocusEvents()
        {
            trace("Starting");
            topControl.type = TextFieldType.INPUT;
            addChild( topControl );
            topControl.height = 60;
            topControl.width = stage.stageWidth;
            topControl.background = true;
            topControl.text = "One control on top.";
            topControl.addEventListener( FocusEvent.FOCUS_IN, flashFocusIn );
            topControl.addEventListener( FocusEvent.FOCUS_OUT, flashFocusOut );

            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 60, stage.stageWidth, stage.stageHeight
- 120 );
            webView.addEventListener( FocusEvent.FOCUS_IN, webFocusIn );
            webView.addEventListener(FocusEvent.FOCUS_OUT, webFocusOut );
            webView.addEventListener(LocationChangeEvent.LOCATION_CHANGING,
                                     function( event:LocationChangeEvent ):void
                                     {
                                        event.preventDefault();
                                     } );
            webView.loadString("<form action='#'><input/><input/><input/></form>");
            webView.assignFocus();

            bottomControl.type = TextFieldType.INPUT;
            addChild( bottomControl );
            bottomControl.y = stage.stageHeight - 60;
            bottomControl.height = 60;
            bottomControl.width = stage.stageWidth;
            bottomControl.background = true;
            bottomControl.text = "One control on the bottom.";
            bottomControl.addEventListener( FocusEvent.FOCUS_IN, flashFocusIn );
```

```
            bottomControl.addEventListener( FocusEvent.FOCUS_OUT, flashFocusOut );}

        private function webFocusIn( event:FocusEvent ):void
        {
            trace("Web focus in");
        }

        private function webFocusOut( event:FocusEvent ):void
        {
            trace("Web focus out: " + event.direction);
            if( event.direction == FocusDirection.TOP )
            {
                stage.focus = topControl;
            }
            else
            {
                stage.focus = bottomControl;
            }
        }

        private function flashFocusIn( event:FocusEvent ):void
        {
            trace("Flash focus in");
            var textfield:TextField = event.target as TextField;
            textfield.backgroundColor = 0xff5566;
        }

        private function flashFocusOut( event:FocusEvent ):void
        {
            trace("Flash focus out");
            var textfield:TextField = event.target as TextField;
            textfield.backgroundColor = 0xffffff;
        }

    }
}
```

# Захват растрового изображения

Объект StageWebView визуализируется над всем содержимым списка отображения. Невозможно добавить содержимое поверх объекта StageWebView. Например, нельзя развернуть раскрывающийся список над содержимым StageWebView. Для решения этой проблемы получите снимок объекта StageWebView. Затем скройте объект StageWebView и вместо него добавьте растровое изображение со снимком.

Следующий пример иллюстрирует получение снимка объекта StageWebView с использованием метода `drawViewPortToBitmapData`. Он скрывает объект StageWebView, устанавливая для рабочей области значение «null». После того как веб-страница полностью загрузится, он вызывает функцию, которая создает и отображает растровое изображение. При выполнении кода отображается две метки: Google и Facebook. По нажатию на метку соответствующая веб-страница захватывается и отображается как снимок на рабочей области.

```
package
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.Sprite;
    import flash.events.*;
    import flash.geom.Rectangle;
    import flash.media.StageWebView;
    import flash.net.*;
    import flash.text.TextField;
    public class stagewebview extends Sprite
    {
        public var webView:StageWebView=new StageWebView();
        public var textGoogle:TextField=new TextField();
        public var textFacebook:TextField=new TextField();
        public function stagewebview()
        {
            textGoogle.htmlText="<b>Google</b>";
            textGoogle.x=300;
            textGoogle.y=-80;
            addChild(textGoogle);
            textFacebook.htmlText="<b>Facebook</b>";
            textFacebook.x=0;
            textFacebook.y=-80;
            addChild(textFacebook);
            textGoogle.addEventListener(MouseEvent.CLICK,goGoogle);
            textFacebook.addEventListener(MouseEvent.CLICK,goFaceBook);
            webView.stage = this.stage;
            webView.viewPort = new Rectangle(0, 0, stage.stageWidth, stage.stageHeight);
        }
    public function goGoogle(e:Event):void
        {
            webView.loadURL("http://www.google.com");
            webView.stage = null;
            webView.addEventListener(Event.COMPLETE,handleLoad);
        }

    public function goFaceBook(e:Event):void
        {
            webView.loadURL("http://www.facebook.com");
            webView.stage = null;
            webView.addEventListener(Event.COMPLETE,handleLoad);
        }
    public function handleLoad(e:Event):void
        {
            var bitmapData:BitmapData = new BitmapData(webView.viewPort.width,
webView.viewPort.height);
            webView.drawViewPortToBitmapData(bitmapData);
            var webViewBitmap:Bitmap=new Bitmap(bitmapData);
            addChild(webViewBitmap);
        }
    }
}
```

# Глава 63. Использование потоков worker для параллелизма

**Flash Player 11.4 и более поздних версий, Adobe AIR 13.4 и более поздних версий для ПК**

Потоки worker ActionScript позволяют выполнять код параллельно или, другими словами, в фоновом режиме, не прерывая выполнения главного кода.

API-интерфейсы параллелизма ActionScript доступны для ПК только в Flash Player 11.4 и более поздних версий и в AIR 3.4 и более поздних версий. Параллелизм не поддерживается на платформах AIR для мобильных устройств.

## Общие сведения о потоках worker и параллелизме

**Flash Player 11.4 и более поздних версий, Adobe AIR 13.4 и более поздних версий для ПК**

Когда приложение не использует потоки worker, его код выполняется как один линейный блок шагов, также называемый *потоком* выполнения. Поток выполняет код, написанный разработчиком. Кроме того, он выполняет большую часть кода, который является частью среды выполнения, в особенности кода, который обновляет экран, когда изменяются свойства экранных объектов. Хотя код записан порциями в виде методов и классов, при запуске код выполняется по одной строке так, как будто он написан в виде одной длинной последовательности шагов. Рассмотрим в качестве гипотетического примера эти действия, выполняемые приложением.

1. Вход в кадр: среда выполнения вызывает любые обработчики события `enterFrame` и по очереди выполняет их код.
2. Перемещение мыши: пользователь перемещает мышь, а среда выполнения вызывает обработчики событий мыши по мере того, как происходят различные события входа в область объекта и выхода из нее
3. Событие завершения загрузки: запрос на загрузку файла XML из URL возвращает данные загруженного файла. Вызывается обработчик события, который выполняет свои шаги, считывая содержимое XML и создавая набор объектов из данных XML.
4. Перемещение мыши: мышь снова перемещается, поэтому среда выполнения вызывает соответствующие обработчики событий мыши
5. Рендеринг: больше нет событий, ожидающих обработки, поэтому среда выполнения обновляет экран на основе изменений, внесенных в экранные объекты
6. Вход в кадр: цикл повторяется с начала

Как описано в примере, гипотетические шаги 1-5 выполняются последовательно в одном блоке времени, который называется кадром. Так как эти они выполняются последовательно одним потоком, среда выполнения не может прервать один шаг процесса для выполнения другого. При частоте 30 кадров в секунду среде выполнения отводится менее одной тридцатой доли секунды на выполнение всех этих операций. Во многих случаях этого времени достаточно для выполнения кода, и среда выполнения просто ждет до начала следующего кадра. Однако предположим, что данные XML, загружаемые на шаге 3, имеют обширную

структуру с многочисленными вложенными уровнями. Поскольку код циклически обрабатывает XML и создает объекты, на это может уйти больше времени, чем одна тридцатая доля секунды. В этом случае последующие шаги (обработка перемещений мыши и перерисовка экрана) выполняются с задержкой. В результате этого экран зависает и «притормаживает», так как перерисовка в ответ на перемещение мыши происходит недостаточно быстро.

Если весь код выполняется в одном потоке, есть только один способ предотвращения таких случайных зависаний. Необходимо избегать длительных операций, таких как циклическая обработка большого набора данных. Потоки worker ActionScript обеспечивают другое решение проблемы. Можно использовать отдельный поток worker для выполнения длительного кода. Каждый поток worker выполняется в отдельном потоке, поэтому фоновый поток worker выполняет длительную операцию в собственном потоке. Это освобождает поток выполнения главного потока worker для перерисовки экрана при входе в каждый кадр: он не блокируется выполнением других шагов.

Такая возможность выполнять несколько операций кода одновременно называется *параллелизмом*. Когда фоновый поток worker завершает свою операцию или промежуточные выполнения операции, можно отправлять уведомления и данные главному потоку worker. Таким образом можно написать код, который выполняет сложные и длительные операции и вместе с тем предотвращает такое неприятное явление, как зависание экрана.

Потоки worker полезны тем, что уменьшают вероятность падения частоты кадров по причине блокирования главного потока рендеринга другим кодом. Однако использование потоков worker предъявляет дополнительные требования к системной памяти и мощности ЦП, что может оказать значительное влияние на общее быстродействие приложения. Так как каждый поток worker использует собственный экземпляр виртуальной машины среды выполнения, даже самый простой поток worker может значительно увеличить потребление ресурсов. При использовании потоков worker проверяйте код на всех целевых платформах, чтобы убедиться, что его требования к системе не слишком велики. Adobe рекомендует не использовать больше одного или двух фоновых потоков worker в обычных сценариях.

# Создание потоков worker и управление ими

**Flash Player 11.4 и более поздних версий, Adobe AIR 13.4 и более поздних версий для ПК**

Первым шагом на пути к использованию потока worker для параллелизма является создание фонового потока worker. Для создания потока worker используются объекты двух типов. Во-первых, это экземпляр Worker, который представляет создаваемый поток. Во-вторых, это объект WorkerDomain, который создает экземпляр Worker и управляет действующими объектами Worker в приложении.

Когда среда выполнения загружается, он автоматически создает объект WorkerDomain. Среда выполнения также автоматически создает поток worker для главного SWF-файла приложения. Первый поток worker называется *первичным потоком worker*.

Так как для приложения создается только один объект WorkerDomain, доступ к экземпляру WorkerDomain обеспечивается с помощью статического свойства `WorkerDomain.current`.

В любое время можно получить доступ к текущему экземпляру Worker (то есть к потоку worker, в котором выполняется текущий код), используя статическое свойство `Worker.current`.

## Создание объекта Worker из SWF-файла

Тогда как главный SWF выполняется в первичном потоке worker, фоновый поток worker выполняет код одного SWF-файла. Чтобы использовать фоновый поток worker, необходимо написать и скомпилировать код потока worker в виде SWF-файла. Чтобы создать фоновый поток worker, родительскому потоку worker требуется доступ к байтам этого SWF-файла в виде объекта ByteArray. Этот объект ByteArray передается методу `createWorker()` объекта WorkerDomain с целью фактического создания потока worker.

Получить SWF фонового потока worker в виде объекта ByteArray можно тремя основными способами.

### Внедрение SWF потока worker

Используйте метатег [Embed] для внедрения SWF потока worker в главный SWF-файл в виде объекта ByteArray.

```
[Embed(source="../swfs/BgWorker.swf", mimeType="application/octet-stream")]
private static var BgWorker_ByteClass:Class;
private function createWorker():void
{
    var workerBytes:ByteArray = new BgWorker_ByteClass();
    var bgWorker:Worker = WorkerDomain.current.createWorker(workerBytes);

    // ... set up worker communication and start the worker
}
```

SWF потока worker компилируется в главный SWF как подкласс ByteArray с именем BgWorker_ByteClass. В результате создания экземпляра этого класса мы получаем объект ByteArray, предварительно заполненный байтами SWF-файла потока worker.

### Загрузка внешнего SWF-файла потока worker

Используйте объект URLLoader для загрузки внешнего SWF файла. SWF-файл должен находиться в том же домене безопасности, например SWF-файл, загруженный из того же интернет-домена, что и главный SWF-файл, или включенный в пакет приложения AIR.

```
var workerLoader:URLLoader = new URLLoader();
workerLoader.dataFormat = URLLoaderDataFormat.BINARY;
workerLoader.addEventListener(Event.COMPLETE, loadComplete);
workerLoader.load(new URLRequest("BgWorker.swf"));

private function loadComplete(event:Event):void
{
    // create the background worker
    var workerBytes:ByteArray = event.target.data as ByteArray;
    var bgWorker:Worker = WorkerDomain.current.createWorker(workerBytes);

    // ... set up worker communication and start the worker
}
```

Когда объект URLLoader завершает загрузку SWF-файла, байты SWF доступны в свойстве `data` этого объекта URLLoader (`event.target.data` в нашем примере).

### Использование главного SWF в качестве SWF потока worker

Можно использовать один SWF-файл и для главного SWF, и для SWF потока worker. Используйте свойство `loaderInfo.bytes` главного класса отображения для доступа к байтам SWF.

```
// The primordial worker's main class constructor
public function PrimordialWorkerClass()
{
    init();
}

private function init():void
{
    var swfBytes:ByteArray = this.loaderInfo.bytes;

    // Check to see if this is the primordial worker or the background worker
    if (Worker.current.isPrimordial)
    {
        // create a background worker
        var bgWorker:Worker = WorkerDomain.current.createWorker(swfBytes);

        // ... set up worker communication and start the worker
    }
    else // entry point for the background worker
    {
        // set up communication between workers using getSharedProperty()
        // ... (not shown)

        // start the background work
    }
}
```

Если применен этот прием, используйте инструкцию `if`, чтобы разветвить код SWF-файла в конструкторе главного класса или в методе, который он вызывает. Чтобы определить, выполняется код в главном потоке worker или в фоновом, проверьте свойство `isPrimordial` текущего объекта Worker, как показано в примере.

## Начало выполнения потока worker

После создания потока worker выполнение его кода запускается посредством вызова метода `start()` объекта Worker. Операция `start()` происходит не сразу. Чтобы узнать, когда выполняется поток worker, зарегистрируйте прослушиватель для события `workerState` объекта Worker. Это событие отправляется, когда объект Worker переходит от одного состояния жизненного цикла к другому, например, когда он начинает выполнение кода. В обработчике событий `workerState` проверьте, что свойство `state` объекта Worker имеет значение `WorkerState.RUNNING`. На этом этапе поток worker выполняется и конструктор его главного класса уже выполнил свои операции. Следующий код демонстрирует регистрацию события `workerState` и вызов метода `start()`.

```
// listen for worker state changes to know when the worker is running
bgWorker.addEventListener(Event.WORKER_STATE, workerStateHandler);
// set up communication between workers using
// setSharedProperty(), createMessageChannel(), etc.
// ... (not shown)
bgWorker.start();
private function workerStateHandler(event:Event):void
{
    if (bgWorker.state == WorkerState.RUNNING)
    {
        // The worker is running.
        // Send it a message or wait for a response.
    }
}
```

## Управление выполнением потока worker

Можно в любое время получить доступ к набору выполняемых потоков worker в приложении с помощью метода `listWorkers()` класса WorkerDomain. Этот метод возвращает набор потоков worker, у которых свойство `state` имеет значение `WorkerState.RUNNING`, в том числе первичный поток worker. Если поток worker не запущен или его выполнение уже остановлено, он не включается в набор.

Если поток worker больше не нужен, можно вызвать метод `terminate()` объекта Worker, чтобы завершить работу потока worker и освободить используемую им память и другие системные ресурсы.

# Обмен данными между потоками worker

**Flash Player 11.4 и более поздних версий, Adobe AIR 13.4 и более поздних версий для ПК**

Хотя потоки worker выполняют свой код в отдельных потоках выполнения, это было бы бесполезно, если бы они были полностью изолированы друг от друга. Связь между потоками worker, в конечном счете, означает обмен данными между ними. Предусмотрено три основных механизма передачи данных от одного потока worker к другому.

Выбирая метод обмена данными, подходящий для конкретной задачи, следует принять во внимание два основных пункта, в которых заключено различие этих методов. Одно из таких различий — наличие или отсутствие события для уведомления принимающего потока о доступности новых данных. При отсутствии такового поток-получатель должен сам выполнять проверку обновлений. Другое различие между этими методами обмена данными заключается в самом способе передачи данных. В одних случаях поток worker получает отдельную копию общих данных, что ведет к созданию большего числа объектов и, соответственно, к потреблению большего количества памяти и вычислительных циклов. В других случаях потоки worker работают с объектами, ссылающимися на общую системную память, что ведет к созданию меньшего числа объектов и к потреблению меньшего объема памяти в целом. Эти различия описаны в следующей таблице:

| **Метод передачи данных** | **Отправляется ли событие при получении данных?** | **Используется ли потоками worker общая память?** |
|---|---|---|
| Общие свойства потоков worker | Нет | Нет, объекты копируются, не передаются по ссылкам |
| Объект MessageChannel | Да | Нет, объекты копируются, не передаются по ссылкам |
| Общий ByteArray | Нет | Да, память общая |

## Передача данных с помощью общего свойства

Самым элементарным способом обмена данными между потоками worker является использование общего свойства. Каждый поток worker имеет внутренний словарь значений общих свойств. Свойства хранятся со строковыми именами ключей, которые позволяют отличать одно свойство от другого. Чтобы сохранить объект в потоке worker как общее свойство, вызовите метод `setSharedProperty()` объекта Worker с двумя аргументами (имя ключа и значение).

```
// code running in the parent worker
bgWorker.setSharedProperty("sharedPropertyName", someObject);
```

Когда общее свойство задано, значение можно прочитать с помощью метода `getSharedProperty()` объекта Worker, передав имя ключа.

```
// code running in the background worker
receivedProperty = Worker.current.getSharedProperty("sharedPropertyName");
```

Нет ограничений относительно того, какой поток worker считывает или задает значение свойства. Например, код в фоновом потоке worker может вызвать его метод `setSharedProperty()` для сохранения значения. Затем код, выполняемый в родительском потоке worker, может использовать `getSharedProperty()` для получения данных.

Значение, передаваемое методу `setSharedProperty()`, может быть объектом практически любого типа. Когда вызывается метод `getSharedProperty()`, возвращаемый объект является копией объекта, переданного методу `setSharedProperty()`, а не ссылкой на сам объект, за исключением нескольких особых случаев. Обмен данными подробно рассматривается в разделе «Общие ссылки и скопированные значения» на странице 1119.

Самым значительным преимуществом использования общего свойства для обмена данными между потоками worker является то, что оно доступно еще до запуска потока worker. Можно вызвать метод `setSharedProperty()` фонового объекта Worker, чтобы задать общее свойство еще до запуска потока worker. Когда родительский поток worker вызывает метод `start()` объекта Worker, среда выполнения вызывает конструктор главного класса дочернего потока worker. Все общие свойства, заданные до вызова метода `start()`, становятся доступными для чтения кодом дочернего потока worker.

## Передача данных с помощью объекта MessageChannel

Канал сообщений обеспечивает одностороннюю связь для передачи данных между двумя потоками worker. Использование объекта MessageChannel для передачи данных между объектами обеспечивает одно ключевое преимущество. Когда отправляется сообщение (объект) с помощью канала сообщений, объект MessageChannel отправляет событие `channelMessage`. Код в потоке-получателе worker может прослушивать это событие для определения доступности данных. Таким образом потоку-получателю worker не требуется постоянно проверять обновления данных.

Канал сообщений связан только с двумя потоками worker: отправителем и получателем. Чтобы создать объект MessageChannel, вызовите метод `createMessageChannel()` объекта-отправителя Worker, передав в качестве аргумента поток-получатель worker.

```
// In the sending worker swf
var sendChannel:MessageChannel;
sendChannel = Worker.current.createMessageChannel(receivingWorker);
```

Оба потока worker должны иметь доступ к объекту MessageChannel. Проще всего для этого передать объект MessageChannel с помощью метода `setSharedProperty()`.

```
receivingWorker.setSharedProperty("incomingChannel", sendChannel);
```

В потоке-получателе worker зарегистрируйте прослушиватель для события `channelMessage` объекта MessageChannel. Это событие отправляется, когда поток-отправитель worker передает данные через канал сообщений.

```
// In the receiving worker swf
var incomingChannel:MessageChannel;
incomingChannel = Worker.current.getSharedProperty("incomingChannel");
incomingChannel.addEventListener(Event.CHANNEL_MESSAGE, handleIncomingMessage);
```

Чтобы передать данные, в потоке-отправителе worker вызовите метод `send()` объекта MessageChannel.

```
// In the sending worker swf
sendChannel.send("This is a message");
```

В потоке-получателе worker объект MessageChannel вызывает обработчик событий `channelMessage`. Затем поток-получатель worker может получить данные, вызвав метод `receive()` объекта.

```
private function handleIncomingMessage(event:Event):void
{
    var message:String = incomingChannel.receive() as String;
}
```

Объект, возвращенный методом получения, имеет тот же тип данных, что и объект, переданный методу `send()`. Полученный объект является копией объекта, переданного отправителем, а не ссылкой на объект в потоке-отправителе worker, если он не относится к одному из немногих типов данных, описанных в разделе «Общие ссылки и скопированные значения» на странице 1119.

## Передача данных при помощи общего массива ByteArray

При передаче объекта от одного потока worker к другому, последний получает новый объект, являющийся копией исходного. Эти два объекта хранятся в различных ячейках системной памяти. Как следствие, каждая передача объекта приводит к созданию его копии, что увеличивает общую память, потребляемую при выполнении приложения. В дополнение к этому никакие изменения, произведенные над объектом в одном потоке, не отразятся в другом потоке. Дополнительные сведения о методике копирования данных приведены в разделе «Общие ссылки и скопированные значения» на странице 1119.

По умолчанию объект ByteArray обладает таким же поведением. При передаче экземпляра ByteArray при помощи метода `setSharedProperty()` объекта Worker или метода `send()` объекта MessageChannel в памяти компьютера создается новый объект ByteArray, и принимающий поток получает ссылку на только что созданный объект. Однако поведение объекта ByteArray можно поменять, задав свойству `shareable` этого объекта значение `true`.

При передаче объекта ByteArray от одного потока worker к другому, последний получает экземпляр ByteArray, ссылающийся на ту же оперативную системную память, что и экземпляр ByteArray первого потока worker. Как только программа в одном потоке worker изменит содержимое общего байтового массива, все изменения сразу же станут доступны всем потокам, имеющим доступ к этому массиву.

Поскольку объекты worker выполняют программный код одновременно, два объекта worker потенциально могут получить доступ к одному и тому же байтовому массиву в один и тот же момент. Это может привести к потере или повреждению данных. Для предупреждения таких проблем можно использовать несколько различных средств, позволяющих управлять доступом к общим ресурсам.

Класс ByteArray имеет методы для проверки и изменения содержимого байтового массива за одну операцию:

- метод atomicCompareAndSwapIntAt();
- метод atomicCompareAndSwapLength().

Кроме того, пакет flash.concurrent содержит классы, предоставляющие контроль доступа при работе с общими ресурсами:

- класс Mutex;
- класс Condition.

## Общие ссылки и скопированные значения

В обычных случаях при вызове метода `Worker.setSharedProperty()` или `MessageChannel.send()` передача объекта потоку-получателю worker осуществляется путем его сериализации в формате AMF. Это влечет за собой несколько последствий.

- Объект, который создается в потоке-получателе worker в результате вызова его метода `getSharedProperty()`, десериализируется из байт AMF. Это копия исходного объекта, а не ссылка на объект. Любые изменения, внесенные в объект в одном потоке worker, не отражаются в копии с другом потоке worker.
- Объекты, которые нельзя сериализировать в формате AMF, например экранные объекты, нельзя передать потоку worker с помощью `Worker.setSharedProperty()` или `MessageChannel.send()`.
- Чтобы можно было правильно десериализировать пользовательский класс, определение класса должно быть зарегистрировано с помощью функции `flash.net.registerClassAlias()` или метаданных `[RemoteClass]`. Для версий класса обоих потоков worker должен использоваться один и тот же псевдоним.

Имеется пять особых объектов, которые предоставляют общий доступ к своим ресурсам, а не копируются между потоками worker:

- объекты Worker;
- объекты MessageChannel;
- общий байтовый массив (объект ByteArray, у которого свойство `shareable` имеет значение `true`);
- объекты Mutex;
- объекты Condition.

Когда экземпляр одного из этих объектов передается с помощью метода `Worker.setSharedProperty()` или `MessageChannel.send()`, каждый поток worker имеет ссылку на один и тот же базовый объект. Изменения, внесенные в экземпляр в одном потоке worker, сразу отражаются в других потоках worker. Кроме того, если один и тот же экземпляр одного из таких объектов передается потоку worker несколько раз, среда выполнения не создает новую копию объекта в потоке-получателе worker. Вместо этого повторно используется та же ссылка.

## Дополнительные приемы обмена данными

В дополнение к механизмам передачи данных, предназначенным специально для потоков worker, обмен данными между потоками worker также может осуществляться с помощью любого существующего API-интерфейса, который поддерживает обмен данными между двумя SWF-приложениями, например:

- локальные общие объекты,
- запись данных в файл в одном потоке worker и чтение данных из файла в другом потоке worker,
- сохранение и чтение данных в базе данных SQLite.

Когда один ресурс используется двумя и более потоками worker, обычно рекомендуется избегать одновременного обращения нескольких потоков worker к ресурсу. Так, например, когда несколько потоков worker одновременно обращаются к файлу в локальной файловой системе, может произойти потеря или повреждение данных. Одновременный доступ может не поддерживаться операционной системой.

Во избежание проблем по причине одновременного доступа используйте классы Mutex и Condition из пакета flash.concurrent, предоставляющие контроль доступа при работе с общими ресурсами.

В отличие от других механизмов совместного использования данных, ядро базы данных SQLite предназначено для параллельного доступа и имеет собственную встроенную поддержку транзакций. Несколько потоков worker могут одновременно обращаться к базе данных SQLite без риска повреждения данных. Так как потоки worker используют разные экземпляры SQLConnection, каждый из них обращается к базе данных в рамках отдельной транзакции. Одновременные операции с данными не влияют на целостность данных.

**См. также**

«Работа с локальными базами данных SQL в AIR» на странице 768
Пакет flash.concurrent

# Глава 64. Безопасность

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Безопасность имеет огромное значение для компании Adobe, пользователей, владельцев веб-сайтов и разработчиков содержимого. По этой причине в Adobe® Flash® Player и Adobe® AIR™ предусмотрен набор правил безопасности и элементов управления, обеспечивающих защиту пользователей, владельцев веб-сайтов и разработчиков содержимого. В этой главе обсуждается модель безопасности для SWF-файлов, опубликованных с использованием ActionScript 3.0 и выполняемых в среде Flash Player 9.0.124.0 или более поздней версии, а также для файлов SWF, HTML и JavaScript, выполняемых в среде AIR 1.0 или более поздней версии, если не указано иное.

В данной теме приводятся общие сведения о безопасности. Здесь не дается исчерпывающее объяснение реализации, сценариев применения или подробностей использования тех или иных API-интерфейсов. Более подробное обсуждение безопасности проигрывателя Flash Player см. в Центре разработчиков Flash Player Developer Center в разделе «Безопасность» по адресу www.adobe.com/go/devnet_security_ru.

## Общие сведения о безопасности платформы Flash

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Модель безопасности, используемая в средах выполнения Flash Player и AIR, главным образом основана на исходном домене для загружаемых SWF-файлов, HTML, медиасодержимого и других ресурсов. Исполняемый код в файле с определенного интернет-домена, например www.example.com, может в любой момент получить доступ ко всем данным в этом домене. Эти ресурсы помещаются в одну группу безопасности, которая называется *изолированной программной средой*. (Дополнительные сведения см. в разделе ««Изолированные программные среды» на странице 1123».)

Например, код ActionScript в SWF-файле может загружать SWF-файлы, растровые изображения, аудио, текстовые файлы и другие ресурсы из своего домена. Кроме того, разрешается перекрестное выполнение сценариев между двумя SWF-файлами из одного домена, если они созданы с использованием ActionScript 3.0. *Перекрестное выполнение сценариев* — это способность кода в одном файле получать доступ к свойствам, методам и объектам, определенным кодом другого файла.

Перекрестное выполнение сценариев не поддерживается между файлами, созданными с помощью ActionScript 3.0, и файлами, использующими предыдущие версии ActionScript. Однако эти файлы могут взаимодействовать с помощью класса LocalConnection. Кроме того, по умолчанию запрещено перекрестное выполнение сценариев между SWF-файлами версии ActionScript 3.0, находящимися в разных доменах. Однако такой доступ можно разрешить, вызвав метод `Security.allowDomain()` в загруженном SWF-файле. Дополнительные сведения см. в разделе ««Перекрестное выполнение сценариев» на странице 1145».

Ниже перечислены основные правила безопасности, которые всегда применяются по умолчанию.

- Ресурсы, находящиеся в одной изолированной программной среде, всегда могут обращаться друг к другу.
- Исполняемые файлы из удаленной изолированной программной среды никогда не могут обращаться к локальным файлам и данным.

Среды выполнения Flash Player и AIR рассматривают следующие узлы как отдельные домены и создают для каждого из них отдельные изолированные среды безопасности:

- `http://example.com`
- `http://www.example.com`
- `http://store.example.com`
- `https://www.example.com`
- `http://192.0.34.166`

Даже если домен с именем, например http://example.com, соответствует определенному IP-адресу, например http://192.0.34.166, среда выполнения создает для них отдельные изолированные программные среды.

Существует два основных способа, которые разработчик может использовать для предоставления SWF-файлу доступа к ресурсам из других изолированных программных сред:

- метод `Security.allowDomain()` (см. раздел ««Элементы управления автора (разработчика)» на странице 1136»);
- файл политики URL-адресов (см. раздел «Элементы управления веб-сайта (файлы политики)» на странице 1132).

В модели безопасности среды выполнения Flash Player или AIR проводится различие между загрузкой содержимого и извлечением или получением данных. *Содержимое* — это мультимедийное содержимое, включая визуальное содержимое, которое можно отобразить в среде выполнения, аудио, видео, а также SWF-файл или HTML с отображаемым мультимедийным содержимым. *Данные* — это то, что доступно только для кода Содержимое и данные загружаются по-разному.

- Загрузка содержимого осуществляется с помощью таких классов, как Loader, Sound и NetStream; с помощью тегов MXML при использовании Flex; или с помощью тегов HTML в приложении AIR.
- Извлечение данных осуществляется из загруженного мультимедийного содержимого с помощью объектов Bitmap, методов `BitmapData.draw()` и `BitmapData.drawWithQuality()`, свойства `Sound.id3` или метода `SoundMixer.computeSpectrum()`. Метод `drawWithQuality` доступен в средах Flash Player 11.3 и AIR 3.3 и более поздних версий.
- Получение данных осуществляется напрямую посредством загрузки из внешнего файла (например, XML-файла) с помощью таких классов, как URLStream, URLLoader, FileReference, Socket и XMLSocket. В AIR предусмотрены дополнительные классы для загрузки данных, такие как FileStream и XMLHttpRequest.

Модель безопасности проигрывателя Flash Player определяет разные правила для загрузки содержимого и получения данных. В общем, для загрузки содержимого предусмотрено меньше ограничений, чем для получения данных.

Как правило, содержимое (SWF-файлы, растровые изображения, mp3-файлы и видео) можно загружать из любого местоположения, однако если оно находится не в том домене, что загружающий его код или содержимое, оно будет помещено в другую изолированную программную среду.

Существует лишь несколько препятствий для загрузки содержимого.

- По умолчанию локальные SWF-файлы (то есть загруженные с несетевого адреса, например с локального жесткого диска) помещаются в изолированную программную среду, локальную для файловой системы. Эти файлы не могут загружать содержимое из сети. Дополнительные сведения см. в разделе ««Локальные изолированные программные среды» на странице 1124».

- Серверы RTMP (Real-Time Messaging Protocol) могут ограничивать доступ к содержимому. Дополнительные сведения см. в разделе ««Содержимое, передаваемое с помощью серверов RTMP» на странице 1145».

Если загруженное мультимедийное содержимое является изображением, аудио- или видеофайлом, его данные, такие как пикселные и звуковые данные, будут доступными для SWF-файла из другой изолированной программной среды, только если домен этого SWF-файла добавлен в файл политики URL-адресов в исходном домене содержимого. Дополнительные сведения см. в разделе ««Получение данных загруженного мультимедийного содержимого» на странице 1149».

К числу других форм загруженных данных относятся текстовые файлы или XML-файлы, загруженные с помощью объекта URLLoader. В этом случае для доступа к данным из другой изолированной программной среды также необходимо предоставить разрешение через файл политики URL-адресов в исходном домене. Дополнительные сведения см. в разделе ««Использование классов URLLoader и URLStream» на странице 1152».

***Примечание.*** *Файлы политики никогда не требуются для загрузки удаленного содержимого или данных кодом, который выполняется внутри изолированной среды приложения AIR.*

# Изолированные программные среды

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Клиентские компьютеры могут получать отдельные файлы, содержащие код, содержимое и данные, из ряда источников, включая внешние веб-сайты, локальную файловую систему или установленное приложение AIR. Среда выполнения Flash Player или AIR в индивидуальном порядке распределяет файлы с кодами и другие ресурсы, такие как общие объекты, растровые изображения, звуки, видео и данные, по изолированным программным средам безопасности на основе того, из каких источников они загружены. В следующих разделах описываются правила, предусмотренные в средах выполнения для управления тем, какие ресурсы доступны коду или содержимому в определенной изолированной программной среде.

Дополнительные сведения о безопасности проигрывателя Flash Player см. в центре разработчиков Flash Player Developer Center в разделе «Безопасность» по адресу www.adobe.com/go/devnet_security_ru.

## Удаленные изолированные программные среды

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Среды выполнения Flash Player и AIR классифицируют ресурсы, получаемые из Интернета (включая SWF-файлы), и помещают их в разные изолированные программные среды в соответствии с их исходными доменами. Например, ресурсы, загруженные с домена *example.com* и с домена *foo.org* будут помещены в разные изолированные программные среды безопасности. По умолчанию этим файлам разрешен доступ к любым ресурсам, находящимся в их домене. Удаленным SWF-файлам можно разрешить доступ к данным в других доменах путем предоставления разрешения от веб-сайта или автора, например, с помощью файлов политики URL-адресов и метода `Security.allowDomain()`. Дополнительные сведения см. в разделах ««Элементы управления веб-сайта (файлы политики)» на странице 1132» и ««Элементы управления автора (разработчика)» на странице 1136».

Удаленные SWF-файлы не могут загружать локальные файлы или ресурсы.

Дополнительные сведения о безопасности проигрывателя Flash Player см. в центре разработчиков Flash Player Developer Center в разделе «Безопасность» по адресу www.adobe.com/go/devnet_security_ru.

## Локальные изолированные программные среды

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

*Локальным* называется любой файл, который указывается с использованием ссылки по протоколу `file:` или пути UNC (Universal Naming Convention — универсальные правила именования). Локальные SWF-файлы помещаются в одну из четырех локальных изолированных программных сред.

- Локальная изолированная программная среда файловой системы: в целях безопасности среды выполнения Flash Player и AIR по умолчанию помещают в нее все локальные файлы. Исполняемый код из этой изолированной программной среды может считывать локальные файлы (например, с помощью класса URLLoader), но не имеет никакой возможности взаимодействовать с сетью. Это предотвращает утечку локальных данных в сеть и другие виды несанкционированного доступа.
- Локальная изолированная программная среда с сетевым подключением: во время компиляции SWF-файла можно указать, что он имеет доступ к сети, когда выполняется в качестве локального файла (см. раздел «Настройка типа изолированной программной среды для локальных SWF-файлов» на странице 1127). Такие файлы помещаются в локальную изолированную программную среду с сетевым подключением. SWF-файлы, помещенные в локальную изолированную программную среду с сетевым подключением, теряют доступ к локальным файлам. Зато они получают доступ к сетевым данным. Однако SWF-файлы из локальной изолированной программной среды с сетевым подключением все равно не могут считывать данные из сети, если им не предоставлено соответствующее разрешение посредством файла политики URL-адресов или вызова метода `Security.allowDomain()`. Чтобы предоставить такое разрешение, файл политики URL-адресов должен разрешать доступ *всем* доменам с использованием оператора `<allow-access-from domain="*"/>` или метода `Security.allowDomain("*")`. Дополнительные сведения см. в разделах ««Элементы управления веб-сайта (файлы политики)» на странице 1132» и ««Элементы управления автора (разработчика)» на странице 1136».
- Локальная доверенная изолированная программная среда: в эту среду помещаются локальные SWF-файлы, зарегистрированные в качестве доверенных (пользователями или программами установки). Системные администраторы и пользователи также имеют возможность перераспределять (перемещать) локальные SWF-файлы в локальную доверенную изолированную программную среду или из нее в соответствии с требованиями безопасности (см. разделы ««Элементы управления администратора» на странице 1129» и ««Элементы управления пользователя» на странице 1130)». SWF-файлы, распределенные в локальную доверенную изолированную программную среду, могут взаимодействовать со всеми остальными SWF-файлами и загружать данные из любого местоположения (удаленного или локального).
- Изолированная программная среда приложения AIR: эта изолированная программная среда включает содержимое, установленное вместе с выполняемым приложением AIR. По умолчанию код, выполняемый в изолированной программной среде приложения AIR может выполнять перекрестные сценарии для кода из любого домена. Однако файлы, находящиеся за пределами изолированной программной среды приложения AIR, не могут выполнять перекрестные сценарии для находящегося внутри нее кода. По умолчанию код и содержимое в изолированной программной среде приложения AIR могут загружать содержимое и данные из любого домена.

Взаимодействие между локальной изолированной программной средой с сетевым подключением и локальной изолированной программной средой файловой системы, а также между локальной изолированной программной средой файловой системы и удаленной изолированной программной средой строго запрещено. Разрешение на такое взаимодействие не может предоставить ни приложение, выполняемое в проигрывателе Flash Player, ни пользователь, ни администратор.

Для выполнения сценариев в любом направлении между локальными HTML-файлами и локальными SWF-файлами, например, с помощью класса ExternalInterface, требуется, чтобы оба файла (HTML и SWF) находились в локальной доверенной изолированной программной среде. Это требование объясняется тем, что локальные модели безопасности обозревателей отличаются от модели проигрывателя Flash Player.

SWF-файлы из локальной изолированной программной среды с сетевым подключением не могут загружать SWF-файлы из локальной изолированной программной среды файловой системы. SWF-файлы из локальной изолированной программной среды файловой системы не могут загружать SWF-файлы из локальной изолированной программной среды с сетевым подключением.

## Изолированная программная среда приложения AIR

**Adobe AIR 1.0 и более поздних версий**

Среда выполнения Adobe AIR добавляет дополнительную изолированную среду *приложения*, которая расширяет модель изолированных программных сред безопасности Flash Player. Файлы, устанавливаемые в составе приложения AIR, загружаются в изолированную программную среду приложения. Все остальные файлы, загружаемые приложением, имеют те же ограничения по безопасности, которые определены в обычной модели безопасности Flash Player.

При установке приложения все файлы, включенные в пакет AIR, устанавливаются на компьютер пользователя в каталог приложения. Разработчики могут обратиться к этому каталогу в коде с использованием схемы URL `app:/` (см. раздел ««Схемы URI» на странице 874»). Все файлы дерева каталога приложения помещаются в изолированную программную среду приложения при выполнении приложения. Содержимое изолированной программной среды приложения наделяется полным набором привилегий, доступных приложению AIR, в том числе возможностью взаимодействия с локальной файловой системой.

Многие приложения AIR используют только такие локально установленные файлы для выполнения приложения. Однако приложения AIR не ограничены файлами в каталоге приложения — они могут загружать файлы любого типа из любых источников. К ним относятся файлы, локально установленные на компьютере пользователя, а также файлы из доступных внешних источников, например из локальной сети или Интернета. Тип файла не влияет на ограничения безопасности; загруженные HTML-файлы обладают теми же привилегиями безопасности, что и загруженные SWF-файлы из того же источника.

Содержимое изолированной программной среды безопасности приложения имеет доступ к API-интерфейсам AIR, к которым содержимое из других изолированных программных сред не имеет доступа. Например, свойство `air.NativeApplication.nativeApplication.applicationDescriptor`, которое возвращает содержимое файла дескриптора приложения, ограничено доступом к содержимому в изолированной программной среде безопасности приложения. Другим примером API-интерфейса с ограниченным доступом является класс FileStream, который содержит методы для чтения и записи в локальную файловую систему.

API-интерфейсы ActionScript, которые доступны только для содержимого в изолированной программной среде безопасности приложения, отмечены логотипом AIR в *справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform*. При использовании этих API-интерфейсов в других изолированных программных средах среда выполнения выдает исключение SecurityError.

Для содержимого HTML (в объекте HTMLLoader) все JavaScript API-интерфейсы AIR (которые доступны через свойство `window.runtime` или объект `air` при использовании файла AIRAliases.js) доступны для содержимого в изолированной программной среде безопасности приложения. Содержимое HTML в другой изолированной программной среде не имеет доступа к свойству `window.runtime`, поэтому такое содержимое не имеет доступа к API-интерфейсам Flash Player.

Содержимое, выполняемое в изолированной программной среде приложения AIR, имеет следующие дополнительные ограничения.

- Содержимое HTML в изолированной программной среде безопасности приложения имеет ограничения по использованию API-интерфейсов, которые могут динамически преобразовывать строки в исполняемый код после загрузки кода. Это предотвращает случайный ввод (и исполнение) приложением кода из источников, не относящихся к приложению (например, из потенциально небезопасных сетевых доменов). Примером может служить использование функции `eval()`. Дополнительные сведения см. в разделе ««Ограничения по использованию кода в содержимом из различных изолированных программных сред» на странице 1168».
- Для предотвращения возможных информационных атак теги `img` в содержимом HTML объектов TextField в ActionScript игнорируются содержимым SWF, находящимся в изолированной программной среде безопасности приложения.
- Содержимое в изолированной программной среде приложения не может использовать протокол `asfunction` в содержимом HTML текстовых полей ActionScript 2.0.
- Содержимое SWF в изолированной программной среде приложения не может использовать междоменный кэш — функцию, добавленную в Flash Player 9 Пакет обновлений 3. Эта функция позволяет проигрывателю Flash Player постоянно кэшировать содержимое компонента платформы Adobe и использовать его в содержимом SWF по запросу (исключая необходимость многократной повторной загрузки содержимого).

## Ограничения для JavaScript внутри AIR

**Adobe AIR 1.0 и более поздних версий**

В отличие от содержимого в изолированной программной среде безопасности приложения содержимое JavaScript из других изолированных программных сред безопасности *может* вызвать функцию `eval()` для исполнения динамически генерируемого кода в любое время. Однако существуют ограничения для сценариев JavaScript, выполняемых за пределами изолированной программной среды приложения AIR. Вот некоторые из них:

- Код JavaScript в изолированной программной среде безопасности, отличной от изолированной программной среды приложения, не имеет доступа к объекту `window.runtime`, поэтому не может исполнять API-интерфейсы AIR.
- По умолчанию содержимое в изолированной программной среде, отличной от изолированной программной среды приложения, не может вызывать XMLHttpRequest для загрузки данных из доменов, отличных от домена, вызывающего запрос. Однако код приложения может предоставить содержимому, не имеющему отношения к приложению, такую возможность, задав атрибут `allowCrossdomainXHR` в кадре-контейнере или во встроенном кадре. Дополнительные сведения см. в разделе ««Ограничения по использованию кода в содержимом из различных изолированных программных сред» на странице 1168».
- Существуют ограничения по вызову метода JavaScript `window.open()`. Дополнительные сведения см. в разделе ««Ограничения по вызову метода JavaScript window.open()» на странице 1171».
- Содержимое HTML в изолированных программных средах безопасности remote (сетевых) может загружать содержимое CSS, `frame`, `iframe` и `img` только из удаленных доменов (с сетевых URL-адресов).
- Содержимое HTML в локальной изолированной программной среде файловой системы, локальной изолированной программной среде с сетевым подключением или в локальной доверенной изолированной программной средае может загружать содержимое CSS, `frame`, `iframe` и `img` только из локальных изолированных программных сред (не с сетевых и программных URL-адресов).

Дополнительные сведения см. в разделе ««Ограничения по использованию кода в содержимом из различных изолированных программных сред» на странице 1168».

## Настройка типа изолированной программной среды для локальных SWF-файлов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Конечный пользователь или администратор компьютера может указать локальный SWF-файл как доверенный, чтобы он мог загружать данные из всех доменов, локальных и сетевых. Это указывается в каталоге глобальной доверенной зоны Global Flash Player Trust и в каталоге пользовательской доверенной зоны User Flash Player Trust. Дополнительные сведения см. в разделах ««Элементы управления администратора» на странице 1129» и ««Элементы управления пользователя» на странице 1130».

Дополнительные сведения о локальных изолированных программных средах см. в разделе ««Локальные изолированные программные среды» на странице 1124».

**Adobe Flash Professional**

Чтобы настроить SWF-файл для локальной изолированной программной среды файловой системы или для локальной изолированной программной среды с сетевым подключением, нужно настроить параметры публикации документа в инструменте разработки.

---

**Adobe Flex**

Чтобы настроить SWF-файл для локальной изолированной программной среды файловой системы или для локальной изолированной программной среды с сетевым подключением, нужно установить флаг `use-network` в компиляторе Adobe Flex. Дополнительные сведения см. в разделе «О параметрах компилятора приложений» руководства *Создание и развертывание приложений Adobe Flex 3*.

---

## Свойство Security.sandboxType

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Автор SWF-файла может использовать статическое свойство `Security.sandboxType`, предназначенное только для чтения, чтобы определить тип изолированной программной среды, в которую среда выполнения Flash Player или AIR поместила SWF-файл. Класс Security включает константы, представляющие возможные значения свойства `Security.sandboxType`, перечисленные ниже.

- `Security.REMOTE`: SWF-файл находится в удаленном местоположении в Интернете и подчиняется правилам изолированной программной среды на основе домена.
- `Security.LOCAL_WITH_FILE`: SWF-файл является локальным файлом, но не является доверенным для пользователя и не публиковался с возможностью сетевого подключения. SWF-файл может считывать информацию из локальных источников данных, но не может обмениваться данными через Интернет.
- `Security.LOCAL_WITH_NETWORK`: SWF-файл является локальным файлом, не является доверенным для пользователя, но публиковался с возможностью сетевого подключения. SWF-файл может обмениваться данными через Интернет, но не может считывать информацию из локальных источников данных.

- `Security.LOCAL_TRUSTED`: SWF-файл является локальным файлом, пользователь сделал его доверенным с помощью диспетчера настроек проигрывателя или файла конфигурации доверия Flash Player. SWF-файл может считывать информацию из локальных источников данных и обмениваться данными через Интернет.
- `Security.APPLICATION`: SWF-файл работает в приложении AIR и был установлен с пакетом (файлом AIR) для этого приложения. По умолчанию файлы в изолированной программной среде приложения AIR могут выполнять перекрестные сценарии для любого файла из любого домена. Однако файлы, находящиеся за пределами изолированной программной среды приложения AIR, не могут выполнять перекрестные сценарии для файла AIR. По умолчанию файлы в изолированной программной среде приложения AIR могут загружать содержимое и данные из любого домена.

# Элементы управления разрешениями

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Модель безопасности клиента Flash Player во время выполнения построена вокруг ресурсов, которые являются объектами, такими как SWF-файлы, локальные данные и URL-адреса в Интернете. *Заинтересованные лица* — это стороны, которые являются владельцами или пользователями этих ресурсов. Заинтересованные лица могут использовать элементы управления (параметры безопасности) для ограничения доступа к своим ресурсам. Для каждого ресурса назначается четыре заинтересованных лица. Проигрыватель Flash Player строго соблюдает иерархию полномочий для этих элементов управления, приведенную в следующей схеме.

*Иерархия элементов управления безопасностью*

Это означает, к примеру, что если администратор запрещает доступ к ресурсу, это ограничение не могут переопределить другие заинтересованные лица.

Для приложений AIR эти элементы управления доступом применяются только к содержимому, выполняемому за пределами изолированной программной среды приложения AIR.

## Элементы управления администратора

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Администратор компьютера (то есть пользователь, выполнивший вход с учетными данными администратора) может применить параметры безопасности проигрывателя Flash Player, затрагивающие всех пользователей компьютера. В некорпоративной среде, например на домашнем компьютере, административный доступ обычно есть у одного пользователя. Даже в корпоративной среде лишь отдельные пользователи могут иметь права администратора компьютера.

Существует два типа элементов управления администратора:

- файл mms.cfg
- каталог Global Flash Player Trust

### файл mms.cfg

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Файл mms.cfg — это текстовый файл, с помощью которого администраторы могут разрешать или запрещать доступ к разным функциям. При запуске проигрыватель Flash Player считывает параметры безопасности из этого файла и использует их для ограничения доступа к функциям. Файл mms.cfg включает параметры, с помощью которых администратор управляет возможностями, такими как элементы управления конфиденциальностью, безопасность локальных файлов, подключения к сокетам и прочее.

SWF-файл может получать некоторые сведения об отключенных возможностях с помощью свойств `Capabilities.avHardwareDisable` и `Capabilities.localFileReadDisable`. Однако большинство параметров, содержащихся в файле mms.cfg, нельзя получить с помощью ActionScript.

Чтобы на компьютере можно было применять политики безопасности и конфиденциальности, независимые от приложения, доступ для изменения файла mms.cfg должны иметь только системные администраторы. Файл mms.cfg не предназначен для использования программами установки. Программа установки, запущенная с использованием полномочий администратора, может изменить содержимое файла mms.cfg. Компания Adobe считает, что это может подорвать доверие пользователей, и призывает создателей программ установки никогда не изменять файл mms.cfg.

Файл mms.cfg хранится в следующем местоположении:

- Windows: *система*\Macromed\Flash\mms.cfg

  (например, C:\WINDOWS\system32\Macromed\Flash\mms.cfg)

- Mac: *поддержка приложений*/Macromedia/mms.cfg

  (например, /Library/Application Support/Macromedia/mms.cfg)

Дополнительные сведения о файле mms.cfg см. в руководстве администратора Flash Player Administration Guide по адресу www.adobe.com/go/flash_player_admin_ru.

### каталог Global Flash Player Trust

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Администраторы и программы установки могут регистрировать заданные локальные SWF-файлы в качестве доверенных для всех пользователей. Такие SWF-файлы распределяются в локальную доверенную изолированную программную среду. Они могут взаимодействовать со всеми остальными SWF-файлами и загружать данные из любых местоположений, удаленных и локальных. Файлы указываются как доверенные в каталоге Global Flash Player Trust, который находится в следующем местоположении:

- Windows: *система*\Macromed\Flash\FlashPlayerTrust

  (например, C:\WINDOWS\system32\Macromed\Flash\FlashPlayerTrust)

- Mac: *поддержка приложений*/Macromedia/FlashPlayerTrust

  (например, /Library/Application Support/Macromedia/FlashPlayerTrust)

Каталог FlashPlayerTrust может содержать несколько текстовых файлов, в каждом из которых перечислены доверенные пути, по одному в строке. Каждый путь может представлять отдельный SWF-файл, HTML-файл или каталог. Строки комментариев начинаются с символа #. Например, файл конфигурации доверия проигрывателя Flash Player, содержащий следующий текст, делает доверенными все файлы в указанном каталоге и во всех его подкаталогах.

```
# Trust files in the following directories:
C:\Documents and Settings\All Users\Documents\SampleApp
```

В файле конфигурации доверия должны перечисляться только локальные пути или сетевые пути SMB. HTTP-пути, добавленные в файл конфигурации доверия, игнорируются. Доверенными могут быть только локальные файлы.

Во избежание конфликтов каждому файлу конфигурации доверия необходимо присвоить имя, соответствующее установившему его приложению, с расширением .cfg.

Распространяя локально выполняемый SWF-файл через программу установки, разработчик может предписать программе установки добавить файл конфигурации в каталог Global Flash Player Trust, чтобы предоставить полные привилегии распространяемому файлу. Программу установки должен запустить пользователь с правами администратора. В отличие от файла mms.cfg, каталог Global Flash Player Trust предусмотрен для того, чтобы регистрировать доверенные файлы с помощью программ установки. С помощью каталога Global Flash Player Trust регистрировать доверенные локальные приложения могут как администраторы, так и программы установки.

Также существуют каталоги Flash Player Trust для отдельных пользователей (см. раздел «Элементы управления пользователя» на странице 1130).

## Элементы управления пользователя

**Flash Player 9 или более поздних версий**

В проигрывателе Flash Player предусмотрено три разных механизма для управления доступом на уровне пользователя: пользовательский интерфейс «Параметры», диспетчер параметров и каталог User Flash Player Trust.

## Пользовательский интерфейс «Параметры» и диспетчер параметров

**Flash Player 9 или более поздних версий**

Пользовательский интерфейс «Параметры» является быстрым интерактивным механизмом для настройки параметров определенного домена. Менеджер параметров имеет более детальный интерфейс и предоставляет возможность вносить глобальные изменения, затрагивающие разрешения для многих или всех доменов. Кроме того, когда SWF-файл запрашивает новое разрешение, что требует принятия решений о безопасности и конфиденциальности во время выполнения, открывается диалоговое окно, в котором пользователь может изменить некоторые параметры проигрывателя Flash Player.

Диспетчер параметров и пользовательский интерфейс «Параметры» содержат опции, связанные с безопасностью, такие как параметры камеры и микрофона, параметры хранения общих объектов, параметры, связанные с устаревшим содержимом, и прочее. Приложения AIR не поддерживают доступ к диспетчеру параметров и пользовательскому интерфейсу настроек.

***Примечание.*** *Параметры, настроенные в файле mms.cfg (см. раздел ««Элементы управления администратора» на странице 1129»), не отражаются в диспетчере параметров.*

Дополнительные сведения о диспетчере параметров см. на странице www.adobe.com/go/settingsmanager_ru.

## Каталог User Flash Player Trust

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Пользователи и программы установки могут регистрировать в качестве доверенных определенные локальные SWF-файлы. Такие SWF-файлы распределяются в локальную доверенную изолированную программную среду. Они могут взаимодействовать со всеми остальными SWF-файлами и загружать данные из любых местоположений, удаленных и локальных. Пользователь указывает файл как доверенный в каталоге User Player Trust, который находится в том же каталоге, что и папка хранения общих объектов Flash (местоположение варьируется в зависимости от текущего пользователя):

- Windows: данные приложений\Macromedia\Flash Player\#Security\FlashPlayerTrust

  (например, C:\Documents and Settings\JohnD\Application Data\Macromedia\Flash Player\#Security\FlashPlayerTrust в Windows XP или C:\Users\JohnD\AppData\Roaming\Macromedia\Flash Player\#Security\FlashPlayerTrust в Windows Vista)

  В Windows папка Application Data скрыта по умолчанию. Чтобы показать скрытые папки и файлы, выберите «Мой компьютер», откройте проводник и выберите «Сервис» > «Свойства папки», затем перейдите на вкладку «Вид». На вкладке «Вид» выберите переключатель «Показывать скрытые файлы и папки».

- Mac: данные приложений/Macromedia/Flash Player/#Security/FlashPlayerTrust

  (например, /Users/JohnD/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust)

  Эти параметры действуют только для текущего пользователя, а не для всех, кто работает с этим компьютером. Если пользователь без прав администратора устанавливает приложение в свою части системы, каталог User Flash Player Trust позволяет программе установки зарегистрировать приложение как доверенное для этого пользователя.

Распространяя локально выполняемый SWF-файл через программу установки, разработчик может предписать программе установки добавить файл конфигурации в каталог User Flash Player Trust, чтобы предоставить полные привилегии распространяемому файлу. Даже в такой ситуации файл каталога User Flash Player Trust контролируется пользователем, потому что он создается в результате действия пользователя (установки).

Также существует каталог Global Flash Player Trust, который используется администраторами и программами установки для регистрации приложений для всех пользователей компьютера (см. раздел ««Элементы управления администратора» на странице 1129»).

## Элементы управления веб-сайта (файлы политики)

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы данные с веб-сервера были доступны SWF-файлам из других доменов, на сервере можно создать файл политики. *Файл политики* — это XML-файл, помещенный в специальное местоположение на сервере.

Файлы политики определяют доступ к ряду ресурсов, включая следующее:

- получение данных растровых изображений, аудио и видео;
- загрузку XML и текстовых файлов;
- импорт SWF-файлов из других изолированных программных сред в домен безопасности загружающего файла;
- доступ к подключениям к сокетам и XML-сокетам.

Объекты ActionScript создают подключения к серверу двух типов: подключения к серверу на основе документов и подключения к сокетам. Такие объекты ActionScript, как Loader, Sound, URLLoader и URLStream, создают подключения к серверу на основе документов и загружают файл с URL-адреса. Объекты Socket и XMLSocket создают подключения к сокетам, которые работают с потоковой передачей данных, а не с загруженными документами.

Так как проигрыватель Flash Player поддерживает два типа подключений к серверу, существует и два типа файлов политики: файлы политики URL-адресов и файлы политики сокетов.

- Для подключений на основе документов требуются *файлы политики URL-адресов*. С помощью этих файлов сервер указывает, что его данные и документы доступны для SWF-файлов, размещенных на определенных доменах или на любых доменах.
- Для подключений к сокетам требуются *файлы политики сокетов*, которые обеспечивают прямое сетевое подключение к TCP-сокетам нижнего уровня с помощью классов Socket и XMLSocket.

Файлы политики должны передаваться проигрывателю Flash Player с использованием того же протокола, который собирается использовать устанавливаемое подключение. Например, когда файл политики помещается на HTTP-сервер, SWF-файлы с других доменов могут загружать данные с этого сервера с использованием протокола HTTP. Однако, если на этот же сервер не добавить файл политики сокетов, то SWF-файлы из других доменов не смогут подключиться к серверу через сокеты. Другими словами, способ получения файла политики должен соответствовать способу подключения.

Далее в этом разделе приводится краткое описание использования файлов политики и синтаксиса применительно к SWF-файлам, опубликованным для проигрывателя Flash Player 10. (Для более ранних версий Flash Player файлы политики реализуются несколько иначе, так как по мере выхода новых версий улучшалась система безопасности проигрывателя Flash Player.) Дополнительные сведения о файлах политики см. в разделе «Изменения в файлах политики в Flash Player 9» центра разработчиков Flash Player Developer Center по адресу www.adobe.com/go/devnet_security_ru.

Коду, выполняемому в изолированной программной среде приложения AIR, не требуется файл политики для доступа к данным через URL или сокет. Коду в приложении AIR, выполняемому за пределами изолированной программной среды приложения, требуется файл политики.

## Основные файлы политики

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

По умолчанию проигрыватель Flash Player (и содержимое AIR, находящееся за пределами изолированной программной среды приложения AIR) сначала ищет файл политики URL с именем `crossdomain.xml` в корневом каталоге сервера, а файл политики сокетов у порта 843. Файлы в этих местоположениях называются *основными файлами политики*. (При подключении к сокетам проигрыватель Flash Player также ищет файл политики сокетов у того же порта, через который устанавливается основное подключение. Однако файл, найденный в этом местоположении, не рассматривается как основной.)

Помимо определения прав доступа, основной файл политики также содержит *метаполитику*. Она указывает, в каких местоположениях могут находиться файлы политики. По умолчанию метаполитика файлов политики для URL имеет значение «master-only» (только основной), то есть файл /crossdomain.xml является единственным допустимым файлом политики на сервере. Для файлов политики сокетов метаполитика по умолчанию имеет значение «all» (все), то есть файл политики может быть у любого порта.

***Примечание.*** *В проигрывателе Flash Player 9 и более ранних версиях метаполитика по умолчанию для файлов политики URL-адресов имела значение «all» (все), то есть файл политики может находиться в любом каталоге. Если вы развернули приложения, которые загружают файлы политики не из файла по умолчанию (/crossdomain.xml) и в данный момент они могут выполняться в проигрывателе Flash Player 10, не забудьте изменить метаполитику основного файла политики, чтобы разрешить использование дополнительных файлов политики, или попросите об этом администратора сервера. Сведения о том, как изменить метаполитику, см. в разделе «Изменения в файлах политики в Flash Player 9» центра разработчиков Flash Player Developer Center по адресу www.adobe.com/go/devnet_security_ru.*

SWF-файл может проверять файл политики под другим именем или в другом местоположении. Для этого нужно вызвать метод `Security.loadPolicyFile()`. Однако, если в основном файле политики не указано, что в данном местоположении может находиться файл политики, то вызов метода `loadPolicyFile()` не даст результатов, даже если там будет файл политики. Метод `loadPolicyFile()` следует вызывать до того, как пытаться выполнять сетевые операции, для которых требуется файл политики. Проигрыватель Flash Player автоматически ставит запросы на сетевое подключение в очередь после соответствующих запросов на получение файла политики. Например, метод `Security.loadPolicyFile()` допустимо вызвать непосредственно перед началом сетевой операции.

Проверяя основной файл политики, проигрыватель Flash Player ожидает ответа сервера в течение трех секунд. Если ответ не получен, Flash Player делает вывод, что основной файл политики не существует. Однако для вызова метода `loadPolicyFile()` нет тайм-аута по умолчанию; проигрыватель Flash Player предполагает, что вызываемый файл существует и ожидает загрузки в течение неопределенного времени. Поэтому, чтобы гарантированно загрузить основной файл политики, необходимо вызвать метод `loadPolicyFile()`.

Несмотря на то что метод называется `Security.loadPolicyFile()` (загрузитьФайлПолитик), файл политики загружается только после отправки сетевого запроса, для которого требуется файл политики. Вызов метода `loadPolicyFile()` просто сообщает проигрывателю Flash Player, в каком местоположении следует искать файл политики, когда он понадобится.

Нельзя получить уведомление об отправке или выполнении запроса файла политики, это просто не нужно. Проигрыватель Flash Player проверяет файлы политики в асинхронном режиме и автоматически ждет установки подключения, пока проверка не будет успешно выполнена.

В следующих разделах содержатся сведения, которые относятся только к файлам политики URL-адресов. Дополнительные сведения о файлах политики сокетов см. в разделе ««Подключение к сокетам» на странице 1152».

## Область действия файла политики URL-адресов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Файл политики URL-адресов применяется только к каталогу, из которого он загружен, и к его дочерним каталогам. Файл политики в корневом каталоге применяется ко всему серверу. Файл политики, загруженный из подкаталога, применяется только к этому каталогу и его подкаталогам.

Файл политики определяет доступ только к тому серверу, где он находится. Например, файл политики, находящийся по адресу https://www.adobe.com:8080/crossdomain.xml, применяется только к вызовам на загрузку данных с сервера www.adobe.com по протоколу HTTPS через порт 8080.

## Указание прав доступа в файле политики URL-адресов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Файл политики содержит один тег `<cross-domain-policy>`, в котором, в свою очередь, может быть любое количество тегов `<allow-access-from>`, начиная с нуля. Каждый тег `<allow-access-from>` содержит атрибут `domain`, который указывает либо точный IP-адрес, либо точное имя домена, либо подстановочный домен (любой домен). Подстановочные домены можно указать двумя способами:

- одной звездочкой, что соответствует всем доменам и всем IP-адресам;
- звездочкой с суффиксом, что соответствует только доменам с указанным суффиксом.

Суффикс должен начинаться с точки. Однако подстановочные домены с суффиксами могут соответствовать доменам, которые содержат только суффиксы без начальной точки. Например, «xyz.com» рассматривается как часть «*.xyz.com». Подстановочные знаки нельзя использовать в IP-адресах.

В следующем примере показан файл политики URL-адресов, который разрешает доступ SWF-файлам из доменов *.example.com, www.friendOfExample.com и 192.0.34.166.

```
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="*.example.com" />
    <allow-access-from domain="www.friendOfExample.com" />
    <allow-access-from domain="192.0.34.166" />
</cross-domain-policy>
```

Если указывается IP-адрес, доступ предоставляется только SWF-файлам, загруженным с этого IP-адреса с использованием IP-синтаксиса (например, http://65.57.83.12/flashmovie.swf). При предоставлении доступа SWF-файлам не используется синтаксис доменных имен. Проигрыватель Flash Player не выполняет разрешение DNS.

Можно разрешить доступ документам из любого домена, как показано в следующем примере.

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
<allow-access-from domain="*" />
</cross-domain-policy>
```

Каждый тег `<allow-access-from>` также имеет дополнительный атрибут `secure`, который по умолчанию имеет значение `true`. Если файл политики находится на HTTPS-сервере и нужно разрешить SWF-файлам с сервера другого типа (не HTTPS) загружать данные с HTTPS-сервера, этому атрибуту можно задать значение `false`.

Присвоение атрибуту `secure` значения `false` может ослабить защиту, обеспечиваемую протоколом HTTPS. В частности, значение `false` открывает защищенное содержимое для несанкционированного доступа или спуфинга. Компания Adobe настоятельно рекомендует не задавать атрибуту `secure` значение `false`.

Если загружаемые данные находятся на HTTPS-сервере, а загружающий их SWF-файл — на HTTP-сервере, компания Adobe рекомендует переместить загружающий SWF-файл на HTTPS-сервер. Благодаря этому все копии конфиденциальных данных будут находиться под защитой HTTPS. Однако, если вы решите оставить загружающий SWF-файл на HTTP-сервере, добавьте атрибут `secure="false"` в тег `<allow-access-from>`, как показано ниже.

```
<allow-access-from domain="www.example.com" secure="false" />
```

Также для предоставления доступа можно использовать тег `allow-http-request-headers-from`. Этот элемент предоставляет клиенту, размещающему содержимое с другого домена, разрешение отправлять вашему домену заголовки, определенные пользователем. Тег `<allow-access-from>` предоставляет другим доменам доступ для получения данных с вашего домена, а тег `allow-http-request-headers-from` предоставляет другим доменам разрешение на отправку данных вашему домену в виде заголовков. В следующем примере всем доменам разрешается отправлять заголовок SOAPAction текущему домену.

```
<cross-domain-policy>
    <allow-http-request-headers-from domain="*" headers="SOAPAction"/>
</cross-domain-policy>
```

Если тег `allow-http-request-headers-from` добавляется в основной файл политики, он применяется ко всем каталогам хоста. В противном случае он применяется только к каталогу и подкаталогам, на которые распространяется файл политики, содержащий тег.

## Предварительная загрузка файлов политики

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Загрузка данных с сервера или подключение к сокету выполняется в асинхронном режиме. Проигрыватель Flash Player просто ожидает завершения загрузки файла политики, прежде чем запустить основную операцию. Однако извлечение пикселных данных из изображений или получение сэмплов звука выполняется в синхронном режиме. Прежде чем можно будет получить данные, должна завершиться загрузка файла политики. Загружая мультимедийные данные, необходимо получить файл политики.

- Если используется метод `Loader.load()`, задайте свойство `checkPolicyFile` параметра `context`, который является объектом LoaderContext.
- Если изображение встраивается в текстовое поле с помощью тега `<img>`, задайте атрибуту `checkPolicyFile` тега `<img>` значение `"true"`, как показано ниже.

  ```
  <img checkPolicyFile = "true" src = "example.jpg">
  ```

- Если используется метод `Sound.load()`, задайте свойство `checkPolicyFile` параметра `context`, который является объектом SoundLoaderContext.
- Если используется NetStream, задайте свойство `checkPolicyFile` объекта NetStream.

Если задан один из этих параметров, проигрыватель Flash Player сначала проверяет файлы политики, уже загруженные для этого домена. Затем он ищет файл политики сервера в местоположении по умолчанию, проверяя теги `<allow-access-from>` и наличие метаполитики. В завершение, рассматриваются все вызовы метода `Security.loadPolicyFile()`, ожидающие обработки, чтобы оценить их область действия.

## Элементы управления автора (разработчика)

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В качестве основного API-интерфейса ActionScript для предоставления прав доступа используется метод `Security.allowDomain()`, который предоставляет права SWF-файлам в указанных доменах. В следующем примере SWF-файл предоставляет доступ SWF-файлам из домена www.example.com.

```
Security.allowDomain("www.example.com")
```

Этот метод предоставляет права для:

- перекрестного выполнения сценариев между SWF-файлам (см. раздел «Перекрестное выполнение сценариев» на странице 1145);
- доступа к списку отображения (см. раздел ««Обход списка отображения» на странице 1148»);
- обнаружения событий (см. раздел «Безопасность событий» на странице 1148);
- полного доступа к свойствам и методам объекта Stage (см. раздел «Безопасность рабочей области» на странице 1147).

Основная цель вызова метода `Security.allowDomain()` состоит в предоставлении разрешения SWF-файлам во внешнем домене выполнять сценарии для SWF-файла, вызывающего метод `Security.allowDomain()`. Дополнительные сведения см. в разделе ««Перекрестное выполнение сценариев» на странице 1145».

Указав IP-адрес в качестве параметра метода `Security.allowDomain()`, нельзя разрешить доступ всем сторонам с этого IP-адреса. В результате этого доступ получит только та сторона, у которой в состав URL входит указанный IP-адрес, а не соответствующее ему доменное имя. Например, если доменное имя www.example.com соответствует IP-адресу 192.0.34.166, вызов метода `Security.allowDomain("192.0.34.166")` не предоставит доступ домену www.example.com.

Чтобы предоставить доступ для всех доменов, можно использовать подстановочный знак `"*"` в качестве параметра метода `Security.allowDomain()`. Так как подстановочный знак `"*"` разрешает *всем* доменам выполнять сценарии для вызывающего SWF-файла, используйте его с осторожностью.

В языке ActionScript предусмотрен еще один API-интерфейс предоставления доступа: метод `Security.allowInsecureDomain()`. Этот метод работает так же, как `Security.allowDomain()`, но когда он вызывается из SWF-файла, находящегося на HTTPS-сервере, он также разрешает доступ к нему для других SWF-файлов, использующих незащищенный протокол, такой как HTTP. Тем не менее, не рекомендуется разрешать выполнение сценариев между файлами, использующими защищенный протокол (HTTPS), и файлами, использующими незащищенный протокол (например, HTTP), так как это может сделать защищенное содержимое уязвимым для несанкционированного доступа и спуфинга. Вот как происходит подобная атака: поскольку метод `Security.allowInsecureDomain()` разрешает доступ к данным, защищенным с помощью протокола HTTPS, для SWF-файлов, использующих протокол HTTP, злоумышленник, находящийся между HTTP-сервером и пользователями, может заменить незащищенный SWF-файл на другой, который сможет открыть ему доступ к данным на HTTPS-сервере.

***Важная информация.*** *Коду, выполняемому в изолированной программной среде приложения AIR, запрещено вызывать методы `allowDomain()` и `allowInsecureDomain()` класса Security.*

Еще один важный метод, связанный с безопасностью, `Security.loadPolicyFile()`, предписывает проигрывателю Flash Player проверять наличие файла политики в нестандартном местоположении. Дополнительные сведения см. в разделе ««Элементы управления веб-сайта (файлы политики)» на странице 1132».

# Ограничение API-интерфейсов сетевого подключения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

API-интерфейсы сетевого подключения можно ограничить двумя способами. С целью предотвращения вредоносной активности блокируется доступ к традиционно зарезервированным портам. Эту блокировку нельзя отменить с помощью кода. Чтобы контролировать доступ SWF-файла к сетевым подключениям через другие порты, можно настроить свойство `allowNetworking`.

## Заблокированные порты

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Проигрыватели Flash Player и Adobe AIR, как и обозреватели, имеют ограничения для HTTP-доступа к определенным портам. Запрещено отправлять HTTP-запросы определенным стандартным портам, которые традиционно используются для серверов, не использующих протокол HTTP.

Данные ограничения применяются ко всем API-интерфейсам, которые обращаются к сетевым URL-адресам. Единственное исключение составляют API-интерфейсы, напрямую вызывающие сокеты, например `Socket.connect()` и `XMLSocket.connect()`, или вызывают метод `Security.loadPolicyFile()`, в который загружается файл политики сокетов. Подключения к сокетам разрешаются или запрещаются с помощью файлов политики сокетов, размещаемых на целевом сервере.

Ниже перечислены API-интерфейсы ActionScript 3.0, на которые распространяется блокировка портов:

`FileReference.download()`, `FileReference.upload()`, `Loader.load()`, `Loader.loadBytes()`, `navigateToURL()`, `NetConnection.call()`, `NetConnection.connect()`, `NetStream.play()`, `Security.loadPolicyFile()`, `sendToURL()`, `Sound.load()`, `URLLoader.load()`, `URLStream.load()`

Кроме того, блокировка портов распространяется на импорт через общую библиотеку, использование тега `<img>` в текстовых полях и загрузку SWF-файлов в HTML-страницу с использованием тегов `<object>` и `<embed>`.

Кроме того, блокировка портов распространяется на использование тега `<img>` в текстовых полях и загрузку SWF-файлов в HTML-страницу с использованием тегов `<object>` и `<embed>`.

Ниже перечислены заблокированные порты:

HTTP: 20 (ftp data), 21 (ftp control)

HTTP и FTP: 1 (tcpmux), 7 (echo), 9 (discard), 11 (systat), 13 (daytime), 15 (netstat), 17 (qotd), 19 (chargen), 22 (ssh), 23 (telnet), 25 (smtp), 37 (time), 42 (name), 43 (nicname), 53 (domain), 77 (priv-rjs), 79 (finger), 87 (ttylink), 95 (supdup), 101 (hostriame), 102 (iso-tsap), 103 (gppitnp), 104 (acr-nema), 109 (pop2), 110 (pop3), 111 (sunrpc), 113 (auth), 115 (sftp), 117 (uucp-path), 119 (nntp), 123 (ntp), 135 (loc-srv / epmap), 139 (netbios), 143 (imap2), 179 (bgp), 389 (ldap), 465 (smtp+ssl), 512 (print / exec), 513 (login), 514 (shell), 515 (printer), 526 (tempo), 530 (courier), 531 (chat), 532 (netnews), 540 (uucp), 556 (remotefs), 563 (nntp+ssl), 587 (smtp), 601 (syslog), 636 (ldap+ssl), 993 (ldap+ssl), 995 (pop3+ssl), 2049 (nfs), 4045 (lockd), 6000 (x11)

## Использование параметра allowNetworking

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Можно контролировать использование SWF-файлом сетевого подключения путем установки параметра `allowNetworking` для тегов `<object>` и `<embed>` на странице HTML, содержащей SWF-содержимое.

Ниже перечислены возможные значения тега `allowNetworking`:

- `"all"` (по умолчанию): в SWF-файле разрешены все API-интерфейсы сетевых подключений;
- `"internal"`: SWF-файл не может вызывать API-интерфейсы навигации в обозревателе или взаимодействия с обозревателем, описанные далее в этом разделе, однако может вызывать другие API-интерфейсы сетевого подключения;
- `"none"`: SWF-файл не может вызывать API-интерфейсы навигации в обозревателе или взаимодействия с обозревателем, описанные далее в этом разделе, и использовать API-интерфейсы взаимодействия между SWF-файлами, также описанными далее.

Параметр `allowNetworking` предназначен для использования в основном в тех случаях, когда SWF-файл и содержащая его HTML-страница находятся в разных доменах. Значения `"internal"` и `"none"` не рекомендуется использовать, когда загружаемый SWF-файл находится в том же домене, что и содержащие его HTML-страницы, так как нельзя гарантировать, что он будет всегда загружаться на только на предназначенную для него HTML-страницу. Недоверенные стороны могут загружать SWF-файл с вашего домена без содержащей его HTML-страницы, и в этом случае ограничение `allowNetworking` будет работать не так, как задумывалось.

Вызов запрещенного API-интерфейса выдает исключение SecurityError.

Добавьте параметр `allowNetworking` и присвойте ему значение в тегах `<object>` и `<embed>` на HTML-странице, содержащей ссылку на SWF-файл, как показано в следующем примере.

```
<object classic="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    Code
base="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,124,
0"
    width="600" height="400" ID="test" align="middle">
<param name="allowNetworking" value="none" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowNetworking="none" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

HTML-страница может также выполнить сценарий для создания тегов внедрения SWF. Такой сценарий нужно изменить, задав нужные значения для свойства `allowNetworking`. HTML-страницы, созданные в Adobe Flash Professional и Adobe Flash Builder, используют функцию `AC_FL_RunContent()` для встраивания ссылок на SWF-файлы. Добавьте настройки параметра `allowNetworking` в сценарий, как показано ниже.

```
AC_FL_RunContent( ... "allowNetworking", "none", ...)
```

Следующие API-интерфейсы запрещены, когда параметр `allowNetworking` имеет значение `"internal"`:

`navigateToURL()`, `fscommand()`, `ExternalInterface.call()`

А когда параметр `allowNetworking` имеет значение `"none"`, помимо перечисленных выше API-интерфейсов также запрещаются следующие:

`sendToURL()`, `FileReference.download()`, `FileReference.upload()`, `Loader.load()`, `LocalConnection.connect()`, `LocalConnection.send()`, `NetConnection.connect()`, `NetStream.play()`, `Security.loadPolicyFile()`, `SharedObject.getLocal()`, `SharedObject.getRemote()`, `Socket.connect()`, `Sound.load()`, `URLLoader.load()`, `URLStream.load()`, `XMLSocket.connect()`

Даже если выбранное значение параметра `allowNetworking` позволяет SWF-файлу использовать API-интерфейс сетевого подключения, могут существовать другие ограничения, связанные с изолированными программными средами (см. раздел «Изолированные программные среды» на странице 1123).

Когда параметр `allowNetworking` имеет значение `"none"`, нельзя ссылаться на внешнее мультимедийное содержимое в теге `<img>` свойства `htmlText` объекта TextField (иначе выдается исключение SecurityError).

Когда параметр `allowNetworking` имеет значение `"none"`, символ из импортированной общей библиотеки, добавленной в инструмент Flash Professional (не в ActionScript), блокируется во время выполнения.

# Безопасность в полноэкранном режиме

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Проигрыватель Flash Player 9.0.27.0 и более поздние версии поддерживают полноэкранный режим, в котором содержимое, выполняемое в проигрывателе может занимать весь экран. Чтобы выйти в полноэкранный режим, необходимо задать в качестве значения свойства `displayState` объекта Stage константу `StageDisplayState.FULL_SCREEN`. Дополнительные сведения см. в разделе ««Работа в полноэкранном режиме» на странице 179».

Для SWF-файлов, выполняемых в удаленной изолированной программной среде, следует учитывать особенности, связанные с безопасностью.

Чтобы включить полноэкранный режим, в теги `<object>` и `<embed>` на HTML-странице, содержащей ссылку на SWF-файл, необходимо добавить параметр `allowFullScreen` в значении `"true"` (значение по умолчанию — `"false"`), как показано в следующем примере.

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"

codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,
18,0"
    width="600" height="400" id="test" align="middle">
<param name="allowFullScreen" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

HTML-страница может также выполнить сценарий для создания тегов внедрения SWF. Такой сценарий нужно изменить, задав нужные значения для свойства `allowFullScreen`. HTML-страницы, созданные при помощи Flash Professional и Flash Builder, используют функцию `AC_FL_RunContent()` для встраивания ссылок на SWF-файлы, поэтому необходимо добавить значения параметра `allowFullScreen`, как в следующем примере.

```
AC_FL_RunContent( ... "allowFullScreen", "true", ...)
```

Код ActionScript, включающий полноэкранный режим, можно вызвать только в ответ на событие мыши или клавиатуры. При вызове этого кода в других ситуациях проигрыватель Flash Player выдает исключение.

Когда содержимое переходит в полноэкранный режим, появляется сообщение о том, как можно выйти из него, чтобы вернуться в обычный режим. Это сообщение отображается в течение нескольких секунд, после чего автоматически исчезает.

Для содержимого, выполняемого в обозревателе, использование клавиатуры в полноэкранном режиме ограничено. В проигрывателе Flash Player 9 поддерживаются только комбинации клавиш, возвращающие приложение в обычный режим, например клавиша «Escape». Пользователи не могут вводить текст в текстовых полях или перемещаться по экрану. В Flash Player 10 или более новой версии поддерживаются определенные клавиши, не содержащие букв или цифр, а именно стрелки, пробел и клавиша табуляции. Однако ввод текста по-прежнему запрещен.

Полноэкранный режим всегда разрешается в автономном проигрывателе или в файле-проекторе. Кроме того, в этих средах полностью поддерживается использование клавиатуры (в том числе для ввода текста).

Вызов свойства `displayState` объекта Stage приводит к созданию исключения для всех вызывающих объектов, находящихся не в той же изолированной программной среде, что и владелец объекта Stage (главный SWF-файл). Дополнительные сведения см. в разделе ««Безопасность рабочей области» на странице 1147».

Администраторы могут отключить полноэкранный режим для SWF-файлов, выполняемых в обозревателе, задав параметр `FullScreenDisable = 1` в файле mms.cfg. Дополнительные сведения см. в разделе ««Элементы управления администратора» на странице 1129».

Чтобы SWF-файл мог выполняться в полноэкранном режиме, он должен быть включен в HTML-страницу.

# Безопасность в полноэкранном интерактивном режиме

**Flash Player 11.3 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Проигрыватель Flash Player 11.3 и более поздние версии поддерживают полноэкранный интерактивный режим, в котором содержимое, выполняемое в проигрывателе, может занимать весь экран *и поддерживать ввод текста*. Чтобы выйти в полноэкранный интерактивный режим, необходимо задать в качестве значения свойства `displayState` объекта Stage константу `StageDisplayState.FULL_SCREEN_INTERACTIVE`. Дополнительные сведения см. в разделе ««Работа в полноэкранном режиме» на странице 179».

Для SWF-файлов, выполняемых в удаленной изолированной программной среде, следует учитывать особенности, связанные с безопасностью.

Чтобы включить полноэкранный режим, в теги `<object>` и `<embed>` на HTML-странице, содержащей ссылку на SWF-файл, необходимо добавить параметр `allowFullScreenInteractive` в значении `"true"` (значение по умолчанию — `"false"`), как показано в следующем примере.

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"

codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,
18,0"
    width="600" height="400" id="test" align="middle">
<param name="allowFullScreenInteractive" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

HTML-страница может также выполнить сценарий для создания тегов внедрения SWF. Такой сценарий нужно изменить, задав нужные значения для свойства `allowFullScreenInteractive`. HTML-страницы, созданные при помощи Flash Professional и Flash Builder, используют функцию `AC_FL_RunContent()` для вставки ссылок на SWF-файлы, поэтому необходимо добавить значения параметра `allowFullScreenInteractive`, как в следующем примере.

```
AC_FL_RunContent( ... "allowFullScreenInteractive", "true", ...)
```

Код ActionScript, включающий полноэкранный интерактивный режим, можно вызвать только в ответ на событие мыши или клавиатуры. При вызове этого кода в других ситуациях проигрыватель Flash Player выдает исключение.

Когда содержимое переходит в полноэкранный интерактивный режим, отображается сообщение о перекрытии. Сообщение содержит домен полноэкранной страницы, инструкции по выходу из полноэкранного режима и кнопку **Разрешить**. Перекрытие присутствует до тех пор, пока пользователь не нажмет кнопку **Разрешить**, подтверждая переход в полноэкранный интерактивный режим.

Администраторы могут отключить полноэкранный интерактивный режим для SWF-файлов, выполняемых в браузерах, задав параметр `FullScreenInteractiveDisable = 1` в файле mms.cfg. Дополнительные сведения см. в разделе ««Элементы управления администратора» на странице 1129».

Чтобы SWF-файл мог выполняться в полноэкранном интерактивном режиме, он должен быть включен в HTML-страницу.

# Загрузка содержимого

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Содержимое Flash Player и AIR может загружать различные типы другого содержимого, включая следующие:

- SWF-файлы;
- изображения;
- Sound
- Видео
- HTML-файлы (только в AIR);
- JavaScript (только в AIR).

## Загрузка SWF-файлов и изображений с помощью класса Loader

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для загрузки SWF-файлов и изображений (JPG, GIF или PNG) используется класс Loader. Все SWF-файлы, кроме находящихся в локальной изолированной программной среде файловой системы, могут загружать SWF-файлы с любого сетевого домена. Загружать SWF-файлы и изображения из локальной файловой системы могут только SWF-файлы, находящиеся в локальных изолированных программных средах. Однако файлы из локальной изолированной программной среды с сетевым подключением могут загружать только те локальные SWF-файлы, которые находятся в локальной доверенной среде или в локальной среде с сетевым подключением. SWF-файлы из локальной изолированной программной среды с сетевым подключением могут загружать локальное содержимое (например, изображения), кроме SWF-файлов, но не могут получать его данные.

Возможно, при загрузке SWF-файла из ненадежного источника (такого как домен, не являющийся доменом корневого SWF-файла объекта Loader) потребуется определить маску для объекта Loader, чтобы предотвратить отрисовку загруженного содержимого (например, дочернего элемента объекта Loader) в областях рабочей области, находящихся за пределами этой маски, как показано в следующем коде.

```
import flash.display.*;
import flash.net.URLRequest;
var rect:Shape = new Shape();
rect.graphics.beginFill(0xFFFFFF);
rect.graphics.drawRect(0, 0, 100, 100);
addChild(rect);
var ldr:Loader = new Loader();
ldr.mask = rect;
var url:String = "http://www.unknown.example.com/content.swf";
var urlReq:URLRequest = new URLRequest(url);
ldr.load(urlReq);
addChild(ldr);
```

При вызове метода `load()` объекта Loader можно указать параметр `context`, представленный объектом LoaderContext. Класс LoaderContext включает три свойства, которые позволяют определить контекст применения загруженного содержимого.

- `checkPolicyFile`: используйте это свойство только при загрузке файла изображения (а не SWF-файла). Укажите его для графического файла, который находится не в том домене, в котором размещен файл, содержащий объект Loader. Если этому свойству задано значение `true`, объект Loader проверяет наличие файла политики URL-адресов на исходном сервере (см. раздел «Элементы управления веб-сайта (файлы политики)» на странице 1132). Если сервер предоставляет доступ домену Loader, код ActionScript из SWF-файлов в домене Loader может получить данные загруженного изображения. Другими словами, свойство `Loader.content` можно использовать для получения ссылки на объект Bitmap, который представляет загруженное изображение, а методы `BitmapData.draw()` или `BitmapData.drawWithQuality()` — для получения пикселных данных загруженного изображения. Метод `drawWithQuality` доступен в средах Flash Player 11.3 и AIR 3.3 и более поздних версий.
- `securityDomain`: используйте это свойство только при загрузке SWF-файлов (но не изображений). Его следует указывать для SWF-файла из домена, который отличается от домена файла, содержащего объект Loader. В настоящее время свойство `securityDomain` поддерживает только два значения: `null` (по умолчанию) и `SecurityDomain.currentDomain`. Если задать значение `SecurityDomain.currentDomain`, отправляется запрос на *импорт* загруженного SWF-файла в изолированную программную среду загружающего SWF-файла, то есть вызываемый файл работает, как если бы он загружался с сервера вызывающего SWF-файла. Это допускается, только если на сервере загружаемого SWF-файла есть файл политики URL-адресов, который позволяет доступ домену загружающего SWF-файла. Если требуемый файл политики найден, загружающий и загружаемый файлы могут выполнять перекрестные сценарии, когда начнется загрузка, так как они оказываются в одной изолированной программной среде. Обратите внимание, что импорт в изолированную программную среду в большинстве случаев можно заменить выполнением обычной загрузки, после которой загруженный SWF-файл должен вызвать метод `Security.allowDomain()`. Последний метод может быть проще в использовании, так как загруженный SWF-файл будет в своей естественной изолированной программной среде и поэтому будет иметь доступ к ресурсам собственного сервера.
- `applicationDomain`: используйте это свойство только при загрузке SWF-файлов, написанных на языке ActionScript 3.0 (а не изображений или SWF-файлов, написанных на ActionScript 1.0 или 2.0). Загружая файл, можно указать, что он должен быть помещен в определенный домен приложений, а не в новый домен, являющийся дочерним для домена приложений загружающего SWF-файла (действие по умолчанию). Обратите внимание, что домены приложений являются подразделениями доменов безопасности, и поэтому целевой домен приложения можно указать, только если загружаемый SWF-файл находится в том же домене безопасности, что и загружающий файл: он должен находиться на том же сервере, либо его необходимо импортировать в данный домен безопасности с помощью свойства `securityDomain`. Если указать домен приложения, в то время как загружаемый SWF-файл находится в другом домене безопасности, свойство `applicationDomain` игнорируется. Дополнительные сведения см. в разделе ««Работа с доменами приложений» на странице 158».

Дополнительные сведения см. в разделе ««Указание контекста загрузки» на странице 216».

Важным свойством объекта Loader является свойство `contentLoaderInfo`, представленное объектом LoaderInfo. В отличие от большинства других объектов, LoaderInfo совместно используется загружающим SWF-файлом и загружаемым содержимым и постоянно доступен для обеих сторон. Когда загружается SWF-файл, он может получить доступ к объекту LoaderInfo через свойство `DisplayObject.loaderInfo`. Объекты LoaderInfo содержат сведения, такие как прогресс загрузки, URL-адреса загружающего и загружаемого файлов, отношения доверенности между файлами и прочее. Дополнительные сведения см. в разделе ««Просмотр хода загрузки» на странице 215».

## Загрузка аудио- и видеосодержимого

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Любому содержимому, за исключением содержимого в локальной изолированной программной среде файловой системы, разрешено загружать звуковые данные и видеоданные из сетевых источников с помощью методов `Sound.load()`, `NetConnection.connect()` и `NetStream.play()`.

Только содержимое в локальной изолированной программной среде файловой системы и изолированной программной среде приложения AIR может загружать мультимедийное содержимое из локальной файловой системы. Только содержимое в локальной изолированной программной среде файловой системы, изолированной программной среде приложения AIR или локальной доверенной изолированной программной среде может получать доступ к данным в этих загруженных файлах.

Существуют также и другие ограничения доступа к данным загруженного мультимедийного содержимого. Дополнительные сведения см. в разделе ««Получение данных загруженного мультимедийного содержимого» на странице 1149».

## Загрузка SWF-файлов и изображений путем добавления тега <img> в текстовое поле

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

SWF-файлы и растровые изображения можно загружать в текстовые поля с помощью тега `<img>`, как показано ниже.

```
<img src = 'filename.jpg' id = 'instanceName' >
```

Доступ к загруженному таким образом содержимому можно получить с помощью метода `getImageReference()` экземпляра TextField, как в следующем коде.

```
var loadedObject:DisplayObject = myTextField.getImageReference('instanceName');
```

Однако следует учесть, что SWF-файлы и изображения, загруженные таким образом, помещаются в изолированную программную среду их источника.

Когда графический файл загружается с помощью тега `<img>` в текстовом поле, доступ к данным изображения можно получить только с помощью файла политики URL-адресов. Чтобы проверить наличие файла политики, следует добавить атрибут `checkPolicyFile` для тега `<img>`, как в следующем коде.

```
<img src = 'filename.jpg' checkPolicyFile = 'true' id = 'instanceName' >
```

Если SWF-файл загружается с помощью тега `<img>` в текстовом поле, доступ к данным этого файла можно открыть, вызвав метод `Security.allowDomain()`.

Когда для загрузки внешнего файла используется тег `<img>` в текстовом поле (в отличие от использования класса Bitmap, встроенного в SWF-файл), автоматически создается объект Loader, являющийся дочерним для объекта TextField, и внешний файл загружается в экземпляр Loader так же, как при загрузке файла с помощью объекта Loader в ActionScript. В данном случае метод `getImageReference()` возвращает автоматически созданный объект Loader. Для обращения к этому объекту Loader не требуется проверка безопасности, так как он находится в той же изолированной программной среде, что и вызывающий код.

Однако правила обеспечения безопасности применяются, если для доступа к загруженному мультимедийному содержимому используется ссылка на свойство `content` объекта Loader. Если содержимое — это изображение, необходимо добавить файл политики URL-адресов, а если это SWF-файл, в его код необходимо добавить вызов метода `allowDomain()`.

**Adobe AIR**

В изолированной программной среде приложения теги <img> в текстовых полях игнорируются с целью предотвращения атак фишинга. Кроме того, коду, выполняемому в изолированной программной среде приложения, запрещено вызывать метод `allowDomain()` класса Security.

---

## Содержимое, передаваемое с помощью серверов RTMP

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Сервер Flash Media Server использует протокол RTMP (Real-Time Media Protocol) для размещения данных, аудио и видео. Это мультимедийное содержимое можно загрузить с помощью метода `connect()` класса NetConnection, передав URL-адрес RTMP в качестве параметра. Сервер Flash Media Server может запретить установку подключений и загрузку содержимого в зависимости того, какой домен запрашивает файл. Дополнительные сведения см. в интерактивной документации по Flash Media Server на странице www.adobe.com/go/learn_fms_docs_ru.

Чтобы использовать методы `BitmapData.draw()`, `BitmapData.drawWithQuality()` и `SoundMixer.computeSpectrum()` для извлечения графики и звуковых данных среды выполнения из потоков RTMP, необходимо разрешить доступ на сервере. Используйте свойства `Client.videoSampleAccess` и `Client.audioSampleAccess` языка ActionScript для сервера для разрешения доступа к определенным каталогам на сервере Flash Media Server. Дополнительные сведения см. в справочнике по серверному языку ActionScript. (Метод `drawWithQuality` доступен в средах Flash Player 11.3 и AIR 3.3 и более поздних версий.)

# Перекрестное выполнение сценариев

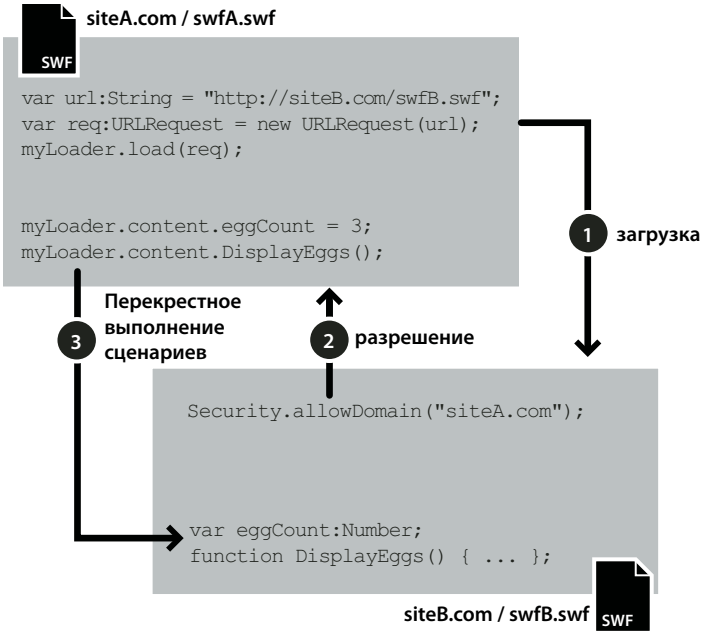
**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Если два SWF-файла, написанные на ActionScript 3.0, или два HTML-файла, выполняемых в AIR, размещены на одном домене (например, URL-адрес одного SWF-файла — http://www.example.com/swfA.swf, а другого http://www.example.com/swfB.swf), то код, определенный в одном файле может анализировать и изменять переменные, объекты, свойства, методы и прочее другого файла и наоборот. Это называется *перекрестным выполнением сценариев*.

Если два файла находятся в разных доменах, например http://siteA.com/swfA.swf и http://siteB.com/swfB.swf, то по умолчанию Flash Player или AIR не позволяет swfA.swf выполнять сценарии, связанные с swfB.swf, и наоборот. SWF-файл предоставляет доступ к SWF-файлам из других доменов путем вызова `Security.allowDomain()`. Вызывая метод `Security.allowDomain("siteA.com")`, файл swfB.swf дает разрешение на выполнение сценариев SWF-файлам с домена siteA.com.

Перекрестное выполнение сценариев не поддерживается между SWF-файлами версии AVM1 и SWF-файлами версии AVM2. SWF-файлы версии AVM1 созданы с использованием ActionScript 1.0 или ActionScript 2.0. (AVM1 и AVM2 — это сокращения от ActionScript Virtual Machine (виртуальная машина ActionScript). Тем не менее, для обмена данными между файлами версий AVM1 и AVM2 можно использовать класс LocalConnection.

В любой ситуации междоменного выполнения сценариев важно иметь четкое представление о двух участвующих в нем сторонах. В рамках данного обсуждения сторона, выполняющая перекрестный сценарий, называется *вызывающей стороной* (обычно это вызывающий SWF-файл), а вторая сторона называется *вызываемой стороной* (обычно это вызываемый SWF-файл). Когда файл siteA.swf выполняет сценарий для siteB.swf, siteA.swf является вызывающей стороной, а siteB.swf — вызываемой, как показано на следующей схеме.




Междоменные разрешения, получаемые с помощью метода `Security.allowDomain()`, не являются симметричными. В предыдущем примере файл siteA.swf может выполнять сценарии для файла siteB.swf, но файл siteB.swf не может выполнять сценарии для файла siteA.swf, так как файл siteA.swf не вызвал метод `Security.allowDomain()`, чтобы разрешить выполнение сценариев SWF-файлами из домена siteB.com. Чтобы настроить симметричные разрешения, метод `Security.allowDomain()` должны вызвать оба SWF-файла.

Проигрыватель Flash Player защищает SWF-файлы от перекрестного выполнения сценариев не только с другими SWF-файлами, но и с HTML-файлами. Выполнение сценариев HTML-файла для SWF-файла может происходить при установке обратного вызова с помощью метода `ExternalInterface.addCallback()`. Когда HTML-файл выполняет сценарии для SWF-файла в другом домене, вызываемый SWF-файл должен вызвать метод `Security.allowDomain()`, как и в том случае, когда вызывающей стороной является SWF-файл, иначе операция завершится ошибкой. Дополнительные сведения см. в разделе ««Элементы управления автора (разработчика)» на странице 1136».

Кроме того, в проигрывателе Flash Player предусмотрены элементы управления безопасностью при выполнении сценариев SWF-файлом для HTML-файла. Дополнительные сведения см. в разделе ««Управление доступом к внешним URL-адресам» на странице 1156.

## Безопасность рабочей области

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Некоторые свойства и методы объекта Stage доступны для всех спрайтов или фрагментов ролика, добавленных в список отображения.

Однако объект Stage имеет владельца, которым является первый загруженный SWF-файл. Следующие свойства и методы объекта Stage по умолчанию доступны только тем SWF-файлам, которые находятся в той же изолированной программной среде, что и владелец рабочей области.

| Свойства | Методы |
|---|---|
| `align` | `addChild()` |
| displayState | `addChildAt()` |
| frameRate | `addEventListener()` |
| `height` | `dispatchEvent()` |
| `mouseChildren` | `hasEventListener()` |
| `numChildren` | `setChildIndex()` |
| `quality` | `willTrigger()` |
| `scaleMode` | |
| `showDefaultContextMenu` | |
| `stageFocusRect` | |
| `stageHeight` | |
| `stageWidth` | |
| `tabChildren` | |
| `textSnapshot` | |
| `width` | |

Чтобы SWF-файл из другой изолированной программной среды мог получить доступ к этим свойствам и методам, SWF-файл, являющийся владельцем рабочей области, должен вызвать метод `Security.allowDomain()`, чтобы разрешить доступ для домена из внешней изолированной программной среды. Дополнительные сведения см. в разделе ««Элементы управления автора (разработчика)» на странице 1136».

Свойство `frameRate` является исключением. Значение `frameRate` может считывать любой SWF-файл. Однако изменить это свойство могут только файлы, находящиеся в той же изолированной программной среде, что и владелец рабочей области (или те, которые получили доступ благодаря вызову метода `Security.allowDomain()`).

Также существуют ограничения по использованию методов `removeChildAt()` и `swapChildrenAt()` объекта Stage, но они отличаются от остальных ограничений. В данном случае, чтобы получить возможность вызывать эти методы, код должен находиться в одном домене с владельцем не рабочей области, а затрагиваемых дочерних объектов, либо эти дочерние объекты должны вызвать метод `Security.allowDomain()`.

## Обход списка отображения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Доступ SWF-файла к экранным объектам, загруженным из других изолированных программных сред, ограничен. Чтобы SWF-файл мог получить доступ к экранному объекту, созданному SWF-файлом из другой изолированной программной среды, вызываемый SWF-файл должен разрешить доступ домену вызывающего SWF-файла с помощью метода `Security.allowDomain()`. Дополнительные сведения см. в разделе ««Элементы управления автора (разработчика)» на странице 1136».

Чтобы можно было получить доступ к объекту Bitmap, загруженному объектом Loader, на исходном сервере графического файла должен быть файл политики для URL, разрешающий домен SWF-файла, который пытается получить объект Bitmap (см. раздел «Элементы управления веб-сайта (файлы политики)» на странице 1132).

Объект LoaderInfo, соответствующий загруженному файлу (и объекту Loader), имеет следующие три свойства, определяющие отношения между загруженным объектом и объектом Loader: `childAllowsParent`, `parentAllowsChild` и `sameDomain`.

## Безопасность событий

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В целях безопасности доступ к событиям, связанным со списком отображения, ограничен в зависимости от того, в какой изолированной программной среде находится экранный объект, отправляющий событие. События в списке отображения проходят стадии восходящей маршрутизации и захвата (см. описание в разделе «Обработка событий» на странице 134). Проходя фазы восходящей маршрутизации и захвата, событие проходит путь от исходного экранного объекта вверх по цепочке родительских экранных объектов в списке отображения. Если родительский объект и исходный экранный объект находятся в разных изолированных программных средах, фазы восходящей маршрутизации и захвата завершаются, не доходя до родительского объекта, если между владельцами родительского и исходного объектов не установлены доверительные отношения. Взаимное доверие можно установить, выполнив следующие действия.

1. SWF-файл, который является владельцем родительского объекта, должен вызвать метод `Security.allowDomain()`, чтобы указать в качестве доверенного домен SWF-файла, который является владельцем исходного объекта.
2. SWF-файл, который является владельцем исходного объекта, должен вызвать метод `Security.allowDomain()`, чтобы указать в качестве доверенного домен SWF-файла, который является владельцем родительского объекта.

Объект LoaderInfo, соответствующий загруженному файлу (и объекту Loader), имеет следующие два свойства, определяющие отношения между загруженным объектом и объектом Loader: `childAllowsParent` и `parentAllowsChild`.

Для событий, отправляемых не экранными объектами, нет таких ограничений по безопасности.

# Получение данных загруженного мультимедийного содержимого

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Для доступа к загруженным данным используйте методы `BitmapData.draw()`, `BitmapData.drawWithQuality()` и `SoundMixer.computeSpectrum()`. По умолчанию нельзя получать пиксельные данные или аудиоданные из объектов графики или звука, которые визуализируются или воспроизводятся мультимедийным содержимым, загруженным в другую изолированную программную среду. Однако можно использовать следующие методы для предоставления прав доступа к этим данным в пределах изолированной программной среды:

- В содержимом, которое выполняет визуализацию или воспроизведение данных, к которым осуществляется доступ, вызовите метод `Security.allowDomain()` для предоставления прав доступа к данным для содержимого в другом домене.
- поместите файл политики URL-адресов на сервер загруженного изображения, аудио или видео. Этот файл политики должен предоставлять доступ домену SWF-файла, который пытается вызвать метод `BitmapData.draw()`, `BitmapData.drawWithQuality()` или `SoundMixer.computeSpectrum()` для получения данных из загруженного файла. Метод `drawWithQuality` доступен в средах Flash Player 11.3 и AIR 3.3 и более поздних версий.

В следующих разделах подробно описывается получение данных растрового изображения, аудио и видео.

## Получение данных растрового изображения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Методы `draw()` и `drawWithQuality()` (Flash Player 11.3; AIR 3.3) объекта BitmapData позволяют записать отображаемые в данный момент пикселы любого экранного объекта в объект BitmapData. Это могут быть пикселы объекта MovieClip, Bitmap или любого другого экранного объекта. Чтобы эти методы могли записать пикселы в объект BitmapData, должны выполняться следующие условия.

- Если исходный объект не является загруженным растровым изображением, то исходный объект и все его дочерние объекты (если это объект Sprite или MovieClip) должны находиться в том же домене, что и объект, вызывающий метод draw, либо они должны входить в состав SWF-файла, доступ к которому разрешен с помощью метода `Security.allowDomain()`.
- Если исходным объектом является загруженное растровое изображение, он должен находиться в том же домене, что и объект, вызывающий метод draw, либо на исходном сервере должен быть файл политики URL-адресов, предоставляющий доступ вызывающему домену.

Если эти условия не выполняются, выдается исключение SecurityError.

При загрузке изображения с помощью метода `load()` класса Loader можно задать параметр `context`, представленный объектом LoaderContext. Если задать свойству `checkPolicyFile` объекта LoaderContext значение `true`, проигрыватель Flash Player проверяет наличие файла политики URL-адресов на сервере, с которого загружается изображение. Если имеется файл политики, разрешающий домен загружающего SWF-файла, тогда этот файл получает доступ к данным объекта Bitmap; в противном случае — не получает.

Также можно определить свойство `checkPolicyFile` в изображении, загруженном с помощью тега `<img>` в текстовом поле. Дополнительные сведения см. в разделе ««Загрузка SWF-файлов и изображений путем добавления тега <img> в текстовое поле» на странице 1144».

## Получение аудиоданных

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

На следующие API-интерфейсы ActionScript 3.0, связанные со звуком, накладываются ограничения в целях безопасности.

- Метод `SoundMixer.computeSpectrum()`: всегда разрешен для кода, выполняемого в той же изолированной программной среде, что и звуковой файл. Для кода из других изолированных программных сред выполняется проверка системы безопасности.
- Метод `SoundMixer.stopAll()`: всегда разрешен для кода, выполняемого в той же изолированной программной среде, что и звуковой файл. Для файлов из других изолированных программных сред выполняется проверка системы безопасности.
- Свойство `id3` объекта Sound всегда разрешено для SWF-файлов, находящихся в той же изолированной программной среде, что и звуковой файл. Для кода из других изолированных программных сред выполняется проверка системы безопасности.

С каждым звуком связаны две изолированные программные среды: среда содержимого и среда владельца.

- Исходный домен звука определяет изолированную программную среду содержимого, а она определяет, можно ли получить данные звука через его свойство `id3` и вызов метода `SoundMixer.computeSpectrum()`.
- Объект, запустивший воспроизведение звука, определяет изолированную программную среду владельца, а она определяет, можно ли остановить воспроизведение звука с помощью метода `SoundMixer.stopAll()`.

При загрузке звука с помощью метода `load()` класса Sound можно задать параметр `context`, представленный объектом SoundLoaderContext. Если задать свойству `checkPolicyFile` объекта SoundLoaderContext значение `true`, среда выполнения проверяет наличие файла политики URL-адресов на сервере, с которого загружается звук. Если имеется файл политики, разрешающий домен загружающего кода, тогда этот код получает доступ к свойству `id` объекта Sound; в противном случае — не получает. Кроме того, можно задать свойство `checkPolicyFile`, чтобы сделать возможным использование метода `SoundMixer.computeSpectrum()` для загруженных звуков.

С помощью метода `SoundMixer.areSoundsInaccessible()` можно выяснить, получится остановить все звуки с помощью метода `SoundMixer.stopAll()` или не получится, так как изолированная программная среда одного или нескольких владельцев звуков является недоступной для вызывающего файла.

Вызов метода `SoundMixer.stopAll()` останавливает те звуки, владельцы которых находятся в одной изолированной программной среде с файлом, вызывающим метод `stopAll()`. Он также останавливает те звуки, воспроизведение которых было запущено SWF-файлами, вызвавшими метод `Security.allowDomain()` для предоставления доступа домену SWF-файла, вызывающего `stopAll()`. Другие звуки не останавливаются, их присутствие можно определить, вызвав метод `SoundMixer.areSoundsInaccessible()`.

Для вызова метода `computeSpectrum()` требуется, чтобы каждый воспроизводимый звук находился либо в одной изолированной программной среде с объектом, вызывающим метод, либо в исходном домене, который предоставил доступ для домена вызывающего файла. В противном случае выдается исключение SecurityError. Чтобы получить доступ к звукам, загруженным из встроенных объектов в библиотеке SWF-файла, нужно вызвать метод `Security.allowDomain()` в загруженном SWF-файле. Чтобы получить доступ к данным звуков, загруженных не из SWF-файлов (например, из mp3-файлов или видеофайлов), необходимо поместить файл политики URL-адресов на исходном сервере.

Дополнительные сведения см. в разделах ««Элементы управления автора (разработчика)» на странице 1136» и ««Элементы управления веб-сайта (файлы политики)» на странице 1132».

Чтобы получить доступ к звуковым данным из потоков RTMP, необходимо разрешить доступ на сервере. Используйте свойство`Client.audioSampleAccess` серверного языка ActionScript для разрешения доступа к определенным каталогам на сервере Flash Media Server. Дополнительные сведения см. в справочнике по серверному языку ActionScript.

## Получение видеоданных

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

С помощью метода `BitmapData.draw()` или `BitmapData.drawWithQuality()` можно захватывать пикселные данные текущего кадра видео. (Метод `drawWithQuality` доступен в средах Flash Player 11.3 и AIR 3.3 и более поздних версий.)

Видеосодержимое бывает двух типов:

- Видео, передаваемое по протоколу RTMP с сервера Flash Media Server
- Последовательное видео, загружаемое из FLV- или F4V-файла

Чтобы использовать методы draw для извлечения графики времени выполнения из потоков RTMP, необходимо разрешить доступ на сервере. Используйте свойство `Client.videoSampleAccess` серверного языка ActionScript для разрешения доступа к определенным каталогам на сервере Flash Media Server. Дополнительные сведения см. в справочнике по серверному языку ActionScript.

Когда вызывается метод draw, для которого в качестве параметра `source` передано последовательное видео, файл, вызывающий метод, должен находиться в одной изолированной программной среде с FLV-файлом, либо на сервере FLV-файла должен быть файл политики, предоставляющий доступ домену вызывающего SWF-файла. Чтобы запросить загрузку файла политики, нужно задать свойству `checkPolicyFile` объекта NetStream значение `true`.

# Загрузка данных

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Содержимое Flash Player и AIR может обмениваться данными с серверами. Операция загрузки данных отличается от загрузки мультимедийного содержимого, так как загруженная информация появляется непосредственно в программных объектах, а не отображается в качестве мультимедийного содержимого. Как правило, содержимое может загружать данные из того же домена, в котором создано содержимое. Однако для содержимого обычно требуются файлы политик для загрузки данных из других доменов (см. раздел ««Элементы управления веб-сайта (файлы политики)» на странице 1132»).

***Примечание.*** *Содержимое, выполняемое в изолированной программной среде приложения AIR, никогда не обслуживается из удаленного домена (если только разработчик намеренно не импортировал удаленное содержимое в изолированную программную среду приложения), поэтому оно не участвует в типах атак, от которых защищают файлы политик. В отношении содержимого AIR, находящегося в изолированной программной среде приложения, не накладываются ограничения на загрузку данных на основе файлов политик. Однако в отношении содержимого AIR в других изолированных программных средах действуют описанные здесь ограничения.*

## Использование классов URLLoader и URLStream

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Можно загружать такие данные, как XML или текстовый файлы. Выполнение методов `load()` классов URLLoader и URLStream регулируется правами, определенными в файлах политики URL-адресов.

Если метод `load()` используется для загрузки содержимого не из того домена, в котором находится код, вызывающий метод, среда выполнения проверяет файл политики URL-адресов на сервере загружаемых ресурсов. Если на сервере есть такой файл и он предоставляет доступ домену загружаемого содержимого, можно загрузить данные.

## Подключение к сокетам

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Среда выполнения по умолчанию ищет файл политики сокетов из порта 843. Как и в случае файлов политики URL, этот файл называется *основным файлом политики*.

Когда файлы политики были впервые введены в версии Flash Player 6, файлы политики сокетов еще не поддерживались. Подключения к сокет-серверам разрешались файлом политики, который находился в местоположении по умолчанию на HTTP-сервере с портом 80 того же хоста, что и сокет-сервер. Проигрыватель Flash Player 9 еще поддерживает эту возможность, а Flash Player 10 уже нет. В проигрывателе Flash Player 10 подключения к сокетам разрешаются только с помощью файлов политики сокетов.

Как и файлы политики URL-адресов, файлы политики сокетов поддерживают использование метаполитики, определяющей порты, у которых могут быть файлы политик. Однако вместо метаполитики «master-only» (только основной) в файлах политики сокетов по умолчанию используется метаполитика «all» (все). Таким образом, если в основном файле политики не установлены более строгие ограничения, проигрыватель Flash Player предполагает, что файл политики сокетов может быть у любого порта узла.

Доступ к подключениям к сокетам и XML-сокетам отключен по умолчанию, даже если сокет находится в том же домене, что и подключающийся SWF-файл. Чтобы разрешить доступ на уровне сокетов, можно поместить файл политики сокетов в любом из следующих местоположений:

- порт 843 (местоположение основного файла политики);
- тот же порт, что и для основного подключения к сокету;
- другой порт.

По умолчанию Flash Player ищет файл политики сокетов у порта 843 и у того же порта, который используется для основного подключения к сокету. Чтобы передавать файл политики сокетов через другой порт, SWF-файл должен вызвать метод `Security.loadPolicyFile()`.

Файл политики сокетов имеет тот же синтаксис, что и файл политики URL-адресов, только в нем также должны быть указаны порты, для которых предоставляется доступ. Если файл политик сокетов передается через порт с номером меньше 1024, он может выдавать разрешение на доступ к любому порту; если файл политик передается через порт с номером 1024 и выше, он может выдавать разрешение на доступ только к другим портам с номером 1024 или выше. Разрешенные порты указываются атрибутом `to-ports` для тега `<allow-access-from>`. Разрешено использование отдельных номеров портов, диапазонов портов и знаков подстановки.

Ниже приводится пример файла политики сокетов.

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<!-- Policy file for xmlsocket://socks.mysite.com -->
<cross-domain-policy>
    <allow-access-from domain="*" to-ports="507" />
    <allow-access-from domain="*.example.com" to-ports="507,516" />
    <allow-access-from domain="*.example.org" to-ports="516-523" />
    <allow-access-from domain="adobe.com" to-ports="507,516-523" />
    <allow-access-from domain="192.0.34.166" to-ports="*" />
</cross-domain-policy>
```

Чтобы получить файл политики сокетов через порт 843 или через тот же порт, через который установлено основное подключение к сокету, нужно вызвать метод `Socket.connect()` или `XMLSocket.connect()`. Проигрыватель Flash Player сначала проверяет основной файл политики у порта 843. Если он его находит, то проверяет наличие метаполитики, запрещающей передачу файлов политики сокетов через целевой порт. Если доступ не запрещен, проигрыватель Flash Player сначала ищет соответствующий атрибут `allow-access-from` в основном файле политики. Если он его не находит, то пытается получить файл политики через порт основного подключения к сокету.

Чтобы получить файл политики сокетов в другом местоположении, сначала нужно вызвать метод `Security.loadPolicyFile()` со специальным синтаксисом `"xmlsocket"`, как показано ниже.

```
Security.loadPolicyFile("xmlsocket://server.com:2525");
```

Вызовите метод `Security.loadPolicyFile()` перед вызовом метода `Socket.connect()` или `XMLSocket.connect()`. После этого проигрыватель Flash Player ожидает выполнения запроса файла политики, прежде чем принять решение относительно основного подключения. Однако, если в основном файле политики указано, что в данном местоположении не может находиться файл политики, то вызов метода `loadPolicyFile()` не даст результатов, даже если там будет файл политики.

Если разворачивается сокет-сервер и требуется добавить файл политики сокетов, нужно определиться, через какой порт будет передаваться этот файл: через порт основных подключений или через другой. В любом случае сервер должен дождаться первой передачи данных от клиента, прежде чем отправить ответ.

Когда проигрыватель Flash Player запрашивает файл политики, он всегда передает следующую строку после установки подключения:

```
<policy-file-request/>
```

Получив эту строку, сервер может передать файл политики. Запрос проигрывателя Flash Player всегда заканчивается нулевым байтом, и ответ сервера также должен заканчиваться нулевым байтом.

Не пытайтесь использовать одно и то же подключение и для запроса файла политики, и для основного подключения. Разорвите подключение после передачи файла политики. Если этого не сделать, проигрыватель Flash Player разрывает подключение, установленное для проверки файла политики, и только после этого устанавливает основное подключение.

## Защита данных

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Чтобы защитить данные от перехвата и изменения при передаче через Интернет, можно использовать протоколы TLS и SSL на сервере, с которого отправляются данные. К серверу можно подключаться с помощью протокола HTTPS.

В приложениях, созданных для среды выполнения AIR 2 и долее поздних версий, также можно защитить обмен данными через сокет TCP. Класс SecureSocket позволяет установить подключение к серверу сокетов, который использует TLS версии 1 или SSL версии 4.

## Отправка данных

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Отправка данных выполняется, когда код отправляет данные на сервер или в ресурс. Отправка данных всегда разрешена для содержимого из сетевого домена. Локальный SWF-файл может отправлять данные на сетевые адреса, только если он находится в локальной доверенной изолированной программной среде, локальной изолированной программной среде с сетевым подключением или изолированной программной среде приложения AIR. Дополнительные сведения см. в разделе ««Локальные изолированные программные среды» на странице 1124».

Отправлять данные на URL-адрес можно с помощью функции `flash.net.sendToURL()`. Другие методы также отправляют запросы URL-адресам. К их числу относятся методы загрузки, такие как `Loader.load()` и `Sound.load()` и методы загрузки данных, такие как `URLLoader.load()` и `URLStream.load()`.

## Отправка и загрузка файлов

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Метод `FileReference.upload()` запускает загрузку выбранного пользователем файла на удаленный сервер. Перед тем как вызвать метод `FileReference.upload()`, следует вызвать метод `FileReference.browse()` или `FileReferenceList.browse()`.

Код, вызывающий метод `FileReference.browse()` или `FileReferenceList.browse()`, можно вызвать только в ответ на событие мыши или клавиатуры. При вызове этого кода в других ситуациях проигрыватель Flash Player 10 и более поздних версий выдает исключение. Однако для инициированного пользователем события не требуется вызывать эти методы из изолированной программной среды приложения AIR.

Вызов метода `FileReference.download()` открывает диалоговое окно, в котором пользователь может загрузить файл с удаленного сервера.

***Примечание.*** *Если ваш сервер требует проверки подлинности пользователей, только Flash-приложения, запущенные в обозревателе (то есть с использованием дополнительного модуля обозревателя или элемента управления ActiveX), могут представить пользователю диалоговое окно для ввода имени пользователя и пароля (для проверки подлинности) и только потом перейти к загрузкам. Проигрыватель Flash Player не разрешает отправку данных на сервер, требующий проверки подлинности.*

Отправка и загрузка не разрешаются, если вызывающий SWF-файл находится в локальной изолированной программной среде файловой системы.

По умолчанию SWF-файл не может инициировать отправку содержимого на другой сервер или загрузку с другого сервера. SWF-файл может выполнять эти операции, только если на этом сервере имеется файл политики, предоставляющий разрешение для домена вызывающего SWF-файла.

# Загрузка встроенного содержимого из SWF-файлов, импортированных в домен безопасности

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Загружая SWF-файл, можно задать параметр `context` для метода `load()` соответствующего объекта Loader. Этому параметру присваивается объект LoaderContext. Когда свойству `securityDomain` этого объекта LoaderContext задается значение `Security.currentDomain`, проигрыватель Flash Player проверяет файл политики URL-адресов на сервере загружаемого SWF-файла. Если на сервере есть такой файл и он предоставляет доступ домену загружающего SWF-файла, метод сможет загрузить SWF-файл как импортированное мультимедийное содержимое. Таким образом, загружающий файл может получить доступ к объектам из библиотеки SWF-файла.

SWF-файл может получить доступ к классам SWF-файлов, загружаемых из других изолированных программных сред, и другим путем: в загружаемом SWF-файле необходимо вызвать метод `Security.allowDomain()`, который предоставляет доступ домену вызывающего SWF-файла. Можно добавить вызов метода `Security.allowDomain()` в конструктор основного класса загружаемого SWF-файла, а затем добавить в загружающий SWF-файл прослушиватель для события `init`, которое отправляется свойством `contentLoaderInfo` объекта Loader. Отправка этого события свидетельствует о том, что загружаемый SWF-файл вызвал метод `Security.allowDomain()` в конструкторе и классы загружаемого файла стали доступными для загружающего файла. Загружающийся SWF-файл может извлекать классы из загруженного SWF-файла путем вызова `Loader.contentLoaderInfo.applicationDomain.getDefinition()` или Loader.contentLoaderInfo.applicationDomain.getQualifiedDefinitionNames() (Flash Player 11.3 и AIR 3.3 или более поздних версий).

# Работа с устаревшим содержимым

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

В Flash Player 6 домен, используемый для определенных параметров проигрывателя, выбирался исходя из конечной порции домена SWF-файла. В таким параметрам относятся параметры доступа к камере и микрофону, квот хранения и хранения постоянных общих объектов.

Если домен SWF-файла включает больше двух сегментов, например www.example.com, первый сегмент домена (www) удаляется, и используется оставшаяся часть домена. Так в Flash Player 6 и www.example.com и store.example.com используют example.com в качестве домена для этих параметров. Подобным образом и www.example.co.uk, и store.example.co.uk используют example.co.uk в качестве домена для параметров. В результате этого могут возникнуть проблемы, связанные с тем, что SWF-файлы из совершенно разных доменов, например example1.co.uk и example2.co.uk будут иметь доступ к одним и тем же общим объектам.

В Flash Player 7 и более поздних версиях параметры проигрывателя по умолчанию выбираются в соответствии с точным доменом SWF-файла. Например, SWF-файл с www.example.com будет использовать параметры проигрывателя для www.example.com, а SWF-файл с store.example.com — отдельные параметры для store.example.com.

Если в SWF-файле версии ActionScript 3.0 свойству `Security.exactSettings` задано значение `true` (используемое по умолчанию), проигрыватель Flash Player использует в параметрах точно определенные домены. Если задано значение `false`, Flash Player использует параметры домена, как в Flash Player 6. Если требуется задать для `exactSettings` значение не по умолчанию, это необходимо сделать до появления событий, требующих выбора параметров проигрывателя Flash Player, таких как использование камеры или микрофона или получение постоянного общего объекта.

Если SWF-файл опубликован в проигрывателе версии 6 и из него созданы постоянные общие объекты, то для получения этих постоянных общих объектов через SWF-файл, использующий ActionScript 3.0, необходимо задать свойству `Security.exactSettings` значение `false`, прежде чем вызывать метод `SharedObject.getLocal()`.

# Установка прав доступа LocalConnection

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Класс LocalConnection позволяет передавать сообщения между двумя приложениями Flash Player или AIR. Объект LocalConnection поддерживает обмен данными только между содержимым Flash Player или AIR, выполняемым на одном клиентском компьютере, однако содержимое может выполняться в разных приложениях. Например, SWF-файл, выполняемый в браузере, SWF-файл, выполняемый в проекторе, и приложение AIR могут обмениваться данными с использованием класса LocalConnection.

Для каждой связи LocalConnection существуют отправитель и прослушиватель. Проигрыватель Flash Player по умолчанию разрешает взаимодействие с помощью класса LocalConnection между кодом, выполняемым в одном домене. Для кода, выполняемого в разных изолированных программных средах, прослушиватель должен предоставить право доступа отправителю с помощью метода `LocalConnection.allowDomain()`. Строка, передаваемая в качестве аргумента методу `LocalConnection.allowDomain()`, может содержать точные имена доменов, IP-адреса или подстановочный знак `*`.

Метод `allowDomain()` изменился по сравнению с тем, каким он был в ActionScript 1.0 и 2.0. В этих ранних версиях `allowDomain()` представлял собой реализуемый метод обратного вызова. В ActionScript 3.0 `allowDomain()` является встроенным методом вызываемого экземпляра класса LocalConnection. После этого изменения `allowDomain()` функционирует практически так же, как и `Security.allowDomain()`.

SWF-файл может использовать свойство `domain` экземпляра класса LocalConnection для определения его домена.

# Управление доступом к внешним URL-адресам

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Выполнение внешних сценариев и доступ к внешним URL-адресам (через HTTP-адреса, ссылки «mailto:» и т. д.) осуществляется с помощью следующих API-интерфейсов.

- функция `flash.system.fscommand()`;
- метод `ExternalInterface.call()`;
- функция `flash.net.navigateToURL()`.

Для содержимого, загруженного из локальной файловой системы, вызовы этих методов успешны, только если код и содержащая веб-страница (если таковая имеется) находятся в локальной доверенной изолированной программной среде или изолированной программной среде приложения AIR. Вызовы этих методов не выполняются, если содержимое находится в локальной изолированной программной среде файловой системы или с сетевым подключением.

Для содержимого, которое не загружено локально, все эти API-интерфейсы могут взаимодействовать с веб-страницей, в которую они встроены, в зависимости от значения параметра AllowScriptAccess, который описывается ниже. Функция `flash.net.navigateToURL()` также может взаимодействовать с любым открытым окном или фреймом обозревателя, а не только со страницей, в которую встроен SWF-файл. Дополнительные сведения об этой возможности см. в разделе ««Использование функции navigateToURL()» на странице 1158».

Параметр `AllowScriptAccess` в HTML-коде, который загружает SWF-файл, управляет возможностью доступа к внешним URL-адресам через SWF-файл. Этот параметр нужно задать внутри тега PARAM или EMBED. Если для параметра `AllowScriptAccess` не задано значение, SWF-файл и HTML-страница могут взаимодействовать, только если находятся в одном и том же домене.

Параметр `AllowScriptAccess` может иметь одно из трех возможных значений: `"always"`, `"sameDomain"` или `"never"`.

- Если параметр `AllowScriptAccess` имеет значение `"always"`, SWF-файл может обращаться к HTML-странице, в которую он встроен, даже если он находится в другом домене.
- Если параметр `AllowScriptAccess` имеет значение `"sameDomain"`, SWF-файл может обращаться к HTML-странице, в которую он встроен, только если они находятся в одном домене. Это значение используется для параметра `AllowScriptAccess` по умолчанию. Используйте эту настройку или не задавайте никакого значения для `AllowScriptAccess`, чтобы SWF-файл из одного домена не мог обращаться к сценарию HTML-страницы из другого домена.
- Если свойство `AllowScriptAccess` имеет значение `"never"`, SWF-файл не может обращаться ни к каким HTML-страницам. Это значение не используется, начиная с выпуска Adobe Flash CS4 Professional. Это не рекомендуется и не является необходимым, если вы не размещаете недоверенные SWF-файлы на собственном домене. Если возникла необходимость размещения недоверенных SWF-файлов, компания Adobe рекомендует создать отдельный субдомен, на который следует помещать все недоверенное содержимое.

Ниже приводится пример настройки тега `AllowScriptAccess` HTML-страницы, которая разрешает доступ к внешним URL-адресам другого домена.

```
<object id='MyMovie.swf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.adobe.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,0,0'
height='100%' width='100%'>
<param name='AllowScriptAccess' value='always'/>
<param name='src' value=''MyMovie.swf'/>
<embed name='MyMovie.swf' pluginspage='http://www.adobe.com/go/getflashplayer'
src='MyMovie.swf' height='100%' width='100%' AllowScriptAccess='never'/>
</object>
```

## Использование функции navigateToURL()

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Помимо обеспечения безопасности с помощью описанного выше параметра `allowScriptAccess`, функция `navigateToURL()` имеет дополнительный второй параметр — `target`. С помощью параметра `target` можно указать имя окна или фрейма HTML-страницы, которому требуется отправить URL-запрос. К таким запросам применяются дополнительные ограничения по безопасности, которые варьируются в зависимости от того, используется метод `navigateToURL()` со сценарием или без него.

Если в метод включен сценарий, например `navigateToURL("javascript: alert('Hello from Flash Player.')")`, применяются следующие правила.

- Если SWF-файл находится в локальной доверенной изолированной среде, запрос выполняется успешно.
- Если целью является HTML-страница, в которую встроен SWF-файл, применяются правила для параметра `allowScriptAccess`, описанные выше.
- Если цель включает содержимое, загруженное из домена SWF-файла, запрос выполняется успешно.
- Если цель включает содержимое, загруженное из другого домена, и не выполнены предыдущие два условия, запрос не выполняется.

Если в метод не включен сценарий (например, ссылки HTTP, HTTPS и `mailto:`), запрос не выполняется при наличии всех перечисленных ниже условий:

- в значении параметра target присутствуют специальные ключевые слова `"_top"` или `"_parent"` и
- SWF-файл встроен в веб-страницу, размещенную на другом домене, и
- SWF-файл встроен с использованием параметра `allowScriptAccess`, для которого не задано значение `"always"`.

## Дополнительные сведения

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Дополнительные сведения о доступе к внешним URL см. в описании следующих элементов в *справочнике ActionScript® 3.0 для платформы Adobe® Flash® Platform:*

- функция flash.system.fscommand();
- метод call() класса ExternalInterface;
- функция `flash.net.navigateToURL()`.

# Общие объекты

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Проигрыватель Flash Player предоставляет возможность использовать *общие объекты*, являющиеся объектами ActionScript, которые постоянно хранятся за пределами SWF-файла либо локально в файловой системе пользователя, либо удаленно на RTMP-сервере. Общие объекты, как и другое мультимедийное содержимое в проигрывателе Flash Player, распределяются по изолированным программным средам. Однако для общих объектов используется немного другая модель изолированных программных сред, так как они не являются ресурсами, к которым можно обращаться с других доменов. Вместо этого общие объекты всегда

извлекаются из хранилища общих объектов, индивидуального для домена каждого SWF-файла, который вызывает методы класса SharedObject. Обычно хранилища общих файлов бывают даже более специфичными: по умолчанию каждый SWF-файл использует хранилище общих файлов, индивидуальное для его полного исходного URL-адреса. Дополнительные сведения об общих объектах см. в разделе ««Общие объекты» на странице 754».

SWF-файл может использовать параметр `localPath` методов `SharedObject.getLocal()` и `SharedObject.getRemote()`, чтобы использовать хранилище общих объектов, связанное только с частью его URL-адреса. Таким образом SWF-файл может разрешить использование общих объектов SWF-файлами с других URL-адресов. Даже если передать значение `'/'` для параметра `localPath`, все равно указывается хранилище общих объектов, индивидуальное для его домена.

Пользователи могут ограничить доступ к общим объектам с помощью диалогового окна параметров Flash Player или диспетчера настроек. По умолчанию общие объекты не могут содержать более 100 кб данных для каждого домена. Администраторы и пользователи также могут устанавливать ограничения на возможность записи данных в файловой системе. Дополнительные сведения см. в разделах ««Элементы управления администратора» на странице 1129» и ««Элементы управления пользователя» на странице 1130».

Чтобы сделать общий объект защищенным, задайте свойство `true` для параметра `secure` метода `SharedObject.getLocal()` или `SharedObject.getRemote()`. Используя параметр `secure`, необходимо учитывать следующее.

- Если данный параметр имеет значение `true`, проигрыватель Flash Player создает новый защищенный общий объект или получает ссылку на существующий защищенный общий объект. Данный защищенный общий объект может считываться или записываться только теми SWF-файлами, доставленными по протоколу HTTPS, которые вызывают метод `SharedObject.getLocal()` с параметром `secure`, имеющим значение `true`.
- Если данный параметр имеет значение `false`, проигрыватель Flash Player создает новый общий объект или получает ссылку на существующий общий объект, который может быть считан или записан SWF-файлами, доставленными посредством подключения по другому протоколу (не HTTPS).

Если вызывающий SWF-файл размещен на незащищенном URL-адресе (без протокола HTTPS), то при передаче значения `true` для параметра `secure` метода `SharedObject.getLocal()` или `SharedObject.getRemote()` выдается исключение SecurityError.

Выбор хранилища общих объектов осуществляется на базе исходного URL-адреса SWF-файла. Это правило действует даже в двух случаях, когда SWF-файл имеет сложный URL-адрес: при загрузке с импортом и при динамической загрузке. При загрузке с импортом SWF-файл загружается с использованием свойства `LoaderContext.securityDomain` в значении `SecurityDomain.currentDomain`. В этой ситуации загружаемый SWF-файл получает псевдо-URL-адрес, в начале которого указывается домен загружающего SWF-файла, а затем исходный URL-адрес. Динамическая загрузка происходит, когда SWF-файл загружается с помощью метода `Loader.loadBytes()`. В этой ситуации загружаемый SWF-файл получает псевдоадрес, в начале которого указывается полный URL-адрес загружающего SWF-файла, а затем числовой идентификатор. И при загрузке с импортом и при динамической загрузке псевдо-URL SWF-файла можно получить с помощью свойства `LoaderInfo.url`. Псевдо-URL используется для выбора хранилища общих объектов точно так же, как и настоящий адрес. Можно указать параметр `localPath` общего объекта, в котором псевдо-URL используется частично или полностью.

Пользователи и администраторы могут отключить использование *сторонних общих объектов*. При этом общие объекты используются любыми SWF-файлами, выполняемыми в веб-обозревателях, когда исходный URL-адрес SWF-файла находится не в том домене, который указан в адресной строке обозревателя. Пользователи и администраторы могут отключить использование сторонних общих объектов, чтобы обеспечить конфиденциальность и избежать междоменного отслеживания. Во избежание такого ограничения

можно сделать так, чтобы SWF-файлы, использующие общие объекты, загружались только в пределах структур HTML-страницы. Это гарантирует, что SWF-файл загружается с того домена, который отображается в адресной строке обозревателя. При попытке использовать общие объекты из стороннего SWF-файла возможность использования сторонних общих объектов отключается и методы `SharedObject.getLocal()` и `SharedObject.getRemote()` возвращают нулевое значение `null`. Дополнительные сведения см. на странице www.adobe.com/products/flashplayer/articles/thirdpartylso.

# Доступ к камере, микрофону, буферу обмена и клавиатуре

**Flash Player 9 и более поздних версий, Adobe AIR 1.0 и более поздних версий**

Когда SWF-файл пытается получить доступ к камере или микрофону пользователя с помощью метода `Camera.get()` или `Microphone.get()`, проигрыватель Flash Player отображает диалоговое окно «Конфиденциальность», в котором пользователь может разрешить или запретить доступ к камере или микрофону. Пользователь и администратор также могут отключить доступ к камере для отдельных или всех сайтов с помощью элементов управления в файле mms.cfg, окне параметров и диспетчере параметров (см. разделы ««Элементы управления администратора» на странице 1129» и ««Элементы управления пользователя» на странице 1130»). Если пользователь применил ограничения, методы `Camera.get()` и `Microphone.get()` возвращают значение `null`. С помощью свойства `Capabilities.avHardwareDisable` можно определить, запретил (`true`) или разрешил (`false`) администратор доступ к камере и микрофону.

Метод `System.setClipboard()` разрешает SWF-файлу заменять содержимое буфера обмена строкой обычного текста. Это не представляет опасности. Соответствующий метод `getClipboard()` отсутствует, чтобы предотвратить риск несанкционированного получения паролей и других конфиденциальных данных, вырезанных или скопированных в буфер обмена.

Приложение, выполняемое в проигрывателе Flash Player, может отслеживать только те события клавиатуры и мыши, которые происходят в пределах его фокуса. Содержимое, выполняемое в проигрывателе Flash Player, не может регистрировать события клавиатуры и мыши в других приложениях.

# Безопасность AIR

**Adobe AIR 1.0 и более поздних версий**

## Основы безопасности AIR

**Adobe AIR 1.0 и более поздних версий**

Приложения AIR выполняются с применением таких же ограничений безопасности, как и в собственных приложениях. В общем приложения AIR, как и собственные приложения, предоставляют широкий доступ к возможностям операционной системы, таким как чтение и запись файлов, запуск приложений, рисование на экране и обмен данными по сети. Ограничения операционной системы, применимые к собственным приложениям, например привилегии конкретных пользователей, в равной степени применяются к приложениям AIR.

Хотя модель безопасности Adobe® AIR™ произошла от модели безопасности Adobe® Flash® Player, договор безопасности отличается от договора безопасности, применимого к содержимому в браузере. Такой договор предлагает разработчикам безопасные инструменты, обеспечивающие более широкую функциональность и такую степень свободы, которая не применима к приложению на основе обозревателя.

Приложения AIR пишутся с использованием либо скомпилированного байт-кода (содержимое SWF), либо интерпретируемого сценария (JavaScript, HTML), что позволяет среде выполнения управлять памятью. Это минимизирует уязвимость приложений AIR, связанную с управлением памятью, например переполнение буфера и повреждение памяти. Это некоторые из наиболее типичных слабых мест, влияющих на настольные приложения, написанные при помощи собственного кода.

## Установка и обновления

**Adobe AIR 1.0 и более поздних версий**

Приложения AIR распространяются с помощью файлов установщика AIR, которые имеют расширение `air`, или с помощью собственных установщиков, которые используют формат файлов и расширение собственной платформы. Например, установщик Windows использует файлы EXE, а на Android собственным форматом являются файлы APK.

После установки Adobe AIR и открытия файла установщика AIR среда выполнения AIR берет на себя управление процессом установки. Если используется собственный установщик, процессом установки управляет операционная система.

***Примечание.*** *Разработчики могут указать версию, имя приложения и источник издателя при использовании формата файла AIR, но исходную процедуру установки приложения изменить невозможно. Такое ограничение выгодно для пользователей, так как все приложения AIR имеют общую безопасную, отлаженную и последовательную процедуру установки, управляемую средой выполнения. Если требуется настройка приложения, ее можно выполнить при его первом выполнении.*

### Местоположение установки среды выполнения

**Adobe AIR 1.0 и более поздних версий**

Приложения AIR, использующие формат файла AIR, требуют предварительной установки среды выполнения на компьютер пользователя, так же как SWF-файлы требуют предварительной установки программного модуля обозревателя проигрывателя Flash Player.

Среда выполнения устанавливается в следующее местоположение на персональном компьютере:

- Mac OS: `/Library/Frameworks/`
- Windows: `C:\Program Files\Common Files\Adobe AIR`
- Linux: `/opt/Adobe AIR/`

В случае с системой Mac OS для установки обновленной версии приложения пользователь должен иметь соответствующие системные привилегии для установки в каталог приложения. В Windows и Linux пользователь должен обладать правами администратора.

***Примечание.*** *В ОС iOS среда выполнения AIR не устанавливается отдельно. Каждое приложение AIR представляет собой автономное приложение.*

Среду выполнения можно установить двумя способами: с использованием функции автоматической установки (установка напрямую с веб-обозревателя) или путем установки вручную. Приложения AIR, упакованные как собственные программы установки, могут выполнять установку среды выполнения AIR в рамках обычного процесса установки приложения (при таком способе распространении среды выполнения AIR требуется соглашение о повторном распространении с Adobe).

## Автоматическая установка (среда выполнения и приложение)

**Adobe AIR 1.0 и более поздних версий**

Благодаря функции автоматической установки разработчики могут предоставить пользователям, которые еще не установили Adobe AIR, возможность упрощенной установки. Для автоматической установки разработчик создает SWF-файл, представляющий приложение для установки. Когда пользователь щелкает SWF-файл для установки приложения, SWF-файл пытается обнаружить среду выполнения. Если среда выполнения не обнаружена, она устанавливается и сразу же активируется во время установки приложения разработчика.

## Установка вручную

**Adobe AIR 1.0 и более поздних версий**

Другим способом является самостоятельная загрузка и установка среды выполнения пользователем вручную до открытия файла AIR. В таком случае разработчик может распространять файл AIR различными способами (например, по электронной почте или при помощи HTML-ссылки на веб-сайте). При открытии файла AIR среда выполнения начинает управлять установкой приложения.

## Процедура установки приложения

**Adobe AIR 1.0 и более поздних версий**

Модель безопасности AIR позволяет пользователям принять решение об установке приложения AIR. Установка AIR имеет некоторые преимущества перед технологиями установки собственных приложений, которые облегчают принятие решения пользователями:

- Среда выполнения обеспечивает последовательную процедуру установки для всех операционных систем, даже если приложение AIR устанавливается по ссылке в веб-обозревателе. В большинстве случаев установка собственных приложений зависит от предоставления обозревателем или другим приложением информации о безопасности, если таковая вообще предоставляется.
- Во время процедуры установки приложения AIR определяется источник приложения и информация о привилегиях, доступных для приложения (если пользователь подтверждает продолжение установки).
- Среда выполнения управляет процессом установки приложения AIR. Приложение AIR не может манипулировать процессом установки, управляемым средой выполнения.

В общем, пользователям не следует устанавливать настольные приложения из ненадежного источника или из источника, который невозможно проверить. Необходимость проверки безопасности собственных приложений справедлива как для приложений AIR, так и для других устанавливаемых приложений.

## Каталог установки приложения

**Adobe AIR 1.0 и более поздних версий**

Каталог установки можно задать одним из следующих способов:

1. Пользователь задает целевой каталог во время установки. Приложение будет установлено в указанное пользователем место.
2. Если пользователь не меняет каталог установки, приложение устанавливается в соответствии с путем, заданным средой выполнения по умолчанию:
   - Mac OS: `~/Applications/`
   - Windows XP и предыдущие версии: `C:\Program Files\`
   - Windows Vista: `~/Apps/`
   - Linux: /opt/

   Если разработчик задал настройку `installFolder` в файле дескриптора приложения, приложение устанавливается в подкаталог этого каталога.

## Файловая система AIR

**Adobe AIR 1.0 и более поздних версий**

Во время установки приложений AIR все файлы, которые разработчик включил в файл установщика AIR, копируются на локальный компьютер пользователя. Установленное приложение состоит из:

- Windows: каталог, содержащий все файлы, входящие в состав файла установщика AIR. Среда выполнения также создает файл exe во время установки приложения AIR.
- Linux: каталог, содержащий все файлы, входящие в состав файла установщика AIR. Во время установки приложения AIR среда выполнения также создает выполняемый двоичный файл.
- Mac OS: файл `app` со всем содержимым файла установщика AIR. Содержимое можно проверить при помощи параметра «Показать содержимое пакета» в приложении Finder. Среда выполнения создает файл app как часть установки приложения AIR.

Выполнение приложения AIR:

- Windows: запуск файла .exe в папке установки или ярлыка, соответствующего этому файлу (например, ярлыка в меню «Пуск» или на рабочем столе).
- Linux: запуск файла .bin в папке установки, выбор приложения в меню «Приложения» (Applications), запуск с помощью псевдонима или ярлыка на рабочем столе.
- Mac OS: запуск файла .app или псевдонима, указывающего на него.

Файловая система приложения также включает в себя вложенные каталоги, связанные с функционированием системы. Например, информация, записанная в зашифрованное локальное хранилище, сохраняется во вложенный каталог каталога с именем, соответствующим идентификатору приложения.

## Хранилище приложения AIR

**Adobe AIR 1.0 и более поздних версий**

Приложения AIR обладают привилегиями на запись в любое местоположение на жестком диске компьютера пользователя; однако разработчикам рекомендуется использовать путь `app-storage:/` для локального хранилища приложения. Файлы, записанные в `app-storage:/` из приложения, находятся в стандартном местоположении, в зависимости от операционной системы пользователя:

- В ОС Mac OS: каталог хранения данных приложения зависит от версии AIR:
  - **AIR 3.2 и более ранних версий**: `<appData>/<appId>/Local Store/`, где `<appData>` — это папка пользовательских настроек, как правило: `/Users/<user>/Library/Preferences`
  - **AIR 3.3 и более поздних версий**: `<path>/Library/Application Support/<appID>/Local Store`, где `<path>` — это `/Users/<user>/Library/Containers/<bundle-id>/Data` (изолированная среда) или `/Users/<user>` (при выполнении за пределами изолированной среды)
- В Windows: каталогом хранилища приложения является `<appData>\<appId>\Local Store\` , где `<appData>` — это «специальная папка» пользователя CSIDL_APPDATA, как правило, `C:\Documents and Settings\<user>\Application Data`
- В Linux: `<appData>/<appID>/Local Store/`, где `<appData>` — это `/home/<user>/.appdata`

Получить доступ к каталогу хранилища приложения можно при помощи свойства `air.File.applicationStorageDirectory`. К содержимому каталога можно получить доступ при помощи метода `resolvePath()` класса File. Дополнительные сведения см. в разделе ««Работа с файловой системой» на странице 701».

## Обновление Adobe AIR

**Adobe AIR 1.0 и более поздних версий**

При установке пользователем приложения AIR, требующего обновленной версии среды выполнения, среда выполнения автоматически устанавливает необходимое обновление.

Для обновления среды выполнения на компьютере необходимы права администратора.

## Обновление приложений AIR

**Adobe AIR 1.0 и более поздних версий**

Разработка и развертывание обновлений представляют собой одну из наиболее серьезных проблем безопасности, с которыми сталкиваются приложения, написанные с использованием собственного кода. API-интерфейс AIR имеет механизм, упрощающий эти процессы: метод `Updater.update()` может быть вызван при запуске приложения для проверки удаленного местоположения файла AIR. При наличии подходящего обновления файл AIR загружается, устанавливается и выполняется перезапуск приложения. Разработчики могут использовать этот класс не только для обеспечения новой функциональности, но и для реагирования на потенциальные проблемы безопасности.

Класс Updater можно использовать только для обновления приложений, распространяемых в виде файлов AIR. Приложения, которые распространяются как собственные, должны использовать собственные средства обновления операционной системы, если они доступны.

***Примечание.*** *Разработчики могут указать версию приложения, задав свойство versionNumber в файле дескриптора приложения.*

### Удаление приложения AIR

**Adobe AIR 1.0 и более поздних версий**

При удалении приложения AIR удаляются все файлы в каталоге приложения. Однако удаляются не все файлы, записанные приложением в местоположения за пределами каталога приложения. Удаление приложений AIR не отменяет изменений, внесенных приложением AIR в файлы за пределами каталога приложения.

### Настройки реестра Windows для администраторов

**Adobe AIR 1.0 и более поздних версий**

В Windows администраторы могут настроить компьютер на запрещение (или разрешение) установки приложения AIR и обновлений среды выполнения. Такие настройки содержатся в реестре Windows под следующим обозначением: HKLM\Software\Policies\Adobe\AIR. Среди них следующие:

| Настройка реестра | Описание |
|---|---|
| AppInstallDisabled | Указывает на то, что установка и удаление приложения AIR разрешены. Если разрешено, имеет значение 0, если нет — значение 1. |
| UntrustedAppInstallDisabled | Указывает, что установка недоверенных приложений AIR (которые не имеют доверенного сертификата) разрешена. Если разрешено, имеет значение 0, если нет — значение 1. |
| UpdateDisabled | Указывает на то, что обновление среды выполнения разрешено либо в фоновом режиме, либо как часть явной установки. Если разрешено, имеет значение 0, если нет — значение 1. |

## Безопасность HTML в Adobe AIR

**Adobe AIR 1.0 и более поздних версий**

В данном разделе описана HTML-архитектура безопасности AIR, а также порядок использования встроенных и обычных кадров и моста изолированной программной среды для установки приложений на базе HTML и безопасной интеграции HTML-содержимого в SWF-приложения.

Среда выполнения реализует правила и предоставляет механизмы для преодоления возможных проблем безопасности в HTML и JavaScript. Те же правила применяются к приложению, исходно написанному на JavaScript, и при загрузке содержимого HTML и JavaScript в приложение на базе SWF. Содержимое в изолированной программной среде приложения и в других изолированных программных средах имеют разные привилегии. При загрузке содержимого во встроенный или обычный кадр среда выполнения реализует безопасный механизм *моста изолированной программной среды*, который позволяет содержимому во встроенном или обычном кадре безопасно обмениваться данными с содержимым в изолированной программной среде безопасности приложения.

AIR SDK содержит три класса для визуализации HTML-содержимого.

Класс HTMLLoader обеспечивает плотную интеграцию кода JavaScript и API-интерфейсов AIR.

Класс StageWebView — это класс визуализации HTML, и он обеспечивает крайне ограниченную интеграцию с домашним приложением AIR. Содержимое, загружаемое классом StageWebView, никогда не помещается в изолированную программную среду приложения, не имеет доступа к данным и не может вызывать функции домашнего приложения AIR. В платформах для настольных систем класс StageWebView использует встроенный HTML-механизм AIR, построенный на основе Webkit, который также используется классом HTMLLoader. В мобильных платформах класс StageWebView использует механизм управления HTML, предоставленный операционной системой. Таким образом, на мобильных платформах при использовании класса StageWebView действуют такие же требования безопасности и возникают аналогичные проблемы безопасности, какие существуют в системном веб-браузере.

Класс TextField может отображать строки текста HTML. Выполнение сценариев JavaScript невозможно, но текст может содержать ссылки и загруженные извне изображения.

Дополнительные сведения см. в разделе ««Избежание ошибок безопасности в JavaScript» на странице 1056».

## Обзор конфигурации приложения на базе HTML

**Adobe AIR 1.0 и более поздних версий**

Кадры и встроенные кадры обеспечивают удобную структуру организации содержимого HTML в AIR. Кадры позволяют поддерживать сохраняемость данных и безопасно работать с удаленным содержимым.

Поскольку HTML в AIR сохраняет обычную постраничную структуру, окружение HTML полностью обновляется, если верхний кадр содержимого HTML «переходит» на другую страницу. При помощи кадров и встроенных кадров можно поддерживать сохраняемость данных в AIR, как и в случае с веб-приложением, выполняющимся в обозревателе. Определение основных объектов приложения в верхнем кадре позволит им сохраниться до перехода кадра на новую страницу. При помощи дочерних или встроенных кадров можно загружать и отображать переходные части приложения. (Существует несколько способов поддержания сохраняемости данных, которые можно использовать в дополнение к кадрам или вместо них. К ним относятся файлы cookie, локальные общие объекты, локальное хранилище файлов, зашифрованное хранилище файлов и локальное хранилище базы данных.)

Поскольку HTML в AIR занимает свое обычное промежуточное положение между исполняемым кодом и данными, AIR размещает содержимое в верхнем кадре среды HTML в изолированной программной среде приложения. После передачи события `load` для страницы AIR ограничивает любые действия, такие как `eval()`, которые могут преобразовать текстовую строку в исполняемый объект. Это ограничение действует, даже если приложение не загружает удаленное содержимое. Чтобы разрешить HTML-содержимому выполнять эти запрещенные операции, необходимо использовать кадры или встроенные кадры, чтобы поместить содержимое в изолированную программную среду вне приложения. (Выполнение содержимого в дочернем кадре изолированной программной среды может быть необходимо при использовании некоторых прикладных структур JavaScript, основанных на функции `eval()`.) Полный перечень ограничений, распространяющихся на JavaScript в изолированной программной среде приложения, см. в разделе «Ограничения по использованию кода в содержимом из различных изолированных программных сред» на странице 1168.

Поскольку HTML в AIR сохраняет способность загружать удаленное, возможно, небезопасное содержимое, AIR реализует стратегию «одинакового источника», которая предотвращает взаимодействие содержимого из одного домена с содержимым из другого. Чтобы разрешить взаимодействие содержимого приложения и содержимого из другого домена, можно создать мост в качестве интерфейса между родительским и дочерним кадром.

### Отношения между родительской и дочерней изолированными программными средами

**Adobe AIR 1.0 и более поздних версий**

AIR добавляет атрибуты `sandboxRoot` и `documentRoot` в элементы HTML frame и iframe. Эти атрибуты позволяют воспринимать содержимое приложения так, будто оно загружено из другого домена:

| Атрибут | Описание |
|---|---|
| sandboxRoot | URL-адрес для определения изолированной программной среды и домена, куда нужно поместить содержимое кадра. Необходимо использовать схемы URL-адресов `file:`, `http:` или `https:`. |
| documentRoot | URL-адрес, с которого загружается содержимое кадра. Необходимо использовать схемы URL-адресов `file:`, `app:` или `app-storage:`. |

Следующий пример преобразует содержимое, установленное в подкаталоге изолированной программной среды приложения, для выполнения в удаленной изолированной программной среде и домене www.example.com:

```
<iframe
    src="ui.html"
    sandboxRoot="http://www.example.com/local/"
    documentRoot="app:/sandbox/">
</iframe>
```

### Создание моста между родительским и дочерним кадрами из различных изолированных программных сред или доменов

**Adobe AIR 1.0 и более поздних версий**

AIR добавляет свойства `childSandboxBridge` и `parentSandboxBridge` для объекта `window` любого дочернего кадра. Эти свойства позволяют задать мосты в качестве интерфейсов между родительским и дочерним кадрами. Каждый мост работает в одном направлении:

`childSandboxBridge` — Свойство `childSandboxBridge` позволяет дочернему кадру предоставить интерфейс для содержимого родительского кадра. Для предоставления интерфейса необходимо назначить свойству `childSandbox` функцию или объект дочернего кадра. После этого можно получить доступ к этому объекту или к этой функции из содержимого родительского кадра. Следующий пример показывает, как сценарий, выполняющийся в дочернем кадре, может предоставить доступ родительскому кадру к объекту, содержащему функцию и свойство:

```
var interface = {};
interface.calculatePrice = function(){
    return .45 + 1.20;
}
interface.storeID = "abc"
window.childSandboxBridge = interface;
```

Если дочернее содержимое находится во встроенном кадре, идентификатор `id` которого имеет значение `"child"`, можно получить доступ к интерфейсу из родительского содержимого, считав свойство `childSandboxBridge` кадра:

```
var childInterface = document.getElementById("child").childSandboxBridge;
air.trace(childInterface.calculatePrice()); //traces "1.65"
air.trace(childInterface.storeID)); //traces "abc"
```

`parentSandboxBridge` — Свойство `parentSandboxBridge` позволяет родительскому кадру предоставить интерфейс для содержимого дочернего кадра. Для предоставления интерфейса необходимо назначить свойству `parentSandbox` дочернего кадра функцию или объект родительского кадра. После этого можно получить доступ к этому объекту или к этой функции из содержимого дочернего кадра. Следующий пример показывает, как сценарий, выполняющийся в родительском кадре, может предоставить доступ дочернему кадру к объекту, содержащему функцию save:

```
var interface = {};
interface.save = function(text){
    var saveFile = air.File("app-storage:/save.txt");
    //write text to file
}
document.getElementById("child").parentSandboxBridge = interface;
```

При помощи этого интерфейса содержимое дочернего кадра может сохранить текст в файл с именем save.txt. Однако иной доступ к файловой системе отсутствует. В общем, содержимое приложения должно предоставлять другим изолированным программным средам как можно более ограниченный интерфейс. Дочернее содержимое может вызывать функцию сохранения следующим образом:

```
var textToSave = "A string.";
window.parentSandboxBridge.save(textToSave);
```

Если дочернее содержимое пытается задать свойство объекта `parentSandboxBridge`, среда выполнения выдает исключение SecurityError. Если родительское содержимое пытается задать свойство объекта `childSandboxBridge`, среда выполнения выдает исключение SecurityError.

## Ограничения по использованию кода в содержимом из различных изолированных программных сред

**Adobe AIR 1.0 и более поздних версий**

Как было сказано во введении к данной теме, «Безопасность HTML в Adobe AIR» на странице 1165, среда выполнения реализует правила и предоставляет механизмы для преодоления возможных проблем безопасности в HTML и JavaScript. В данном разделе описываются подобные ограничения. При попытке кода вызвать API-интерфейсы с ограниченным доступом среда выполнения выдает ошибку с сообщением «Нарушение защиты среды выполнения Adobe AIR со стороны кода JavaScript в изолированной программной среде безопасности приложения».

Дополнительные сведения см. в разделе ««Избежание ошибок безопасности в JavaScript» на странице 1056».

### Ограничения по использованию функции JavaScript eval() и подобных методов

**Adobe AIR 1.0 и более поздних версий**

На содержимое HTML в изолированной программной среде безопасности приложения распространяются ограничения по использованию API-интерфейсов, которые могут динамически преобразовывать строки в исполняемый код после загрузки кода (после отправки события `onload` элемента `body` и завершения выполнении функции обработчика `onload`). Это предотвращает случайный ввод (и исполнение) приложением кода из источников, не относящихся к приложению (например, из потенциально небезопасных сетевых доменов).

Например, если приложение использует строковые данные из удаленного источника для записи в свойство innerHTML элемента DOM, такая строка может включать в себя код (JavaScript), выполняющий небезопасные операции. Однако во время загрузки кода риск вставки удаленных строк в элемент DOM отсутствует.

Одно из ограничений касается использования функции JavaScript `eval()`. После загрузки кода из изолированной программной среды приложения и завершения работы обработчика события onload использование функции `eval()` становится ограниченным. Использование функции `eval()` *после* загрузки кода из изолированной программной среды безопасности приложения подчиняется следующим правилам:

- Допускается использование выражений с литералами. Например:

```
eval("null");
eval("3 + .14");
eval("'foo'");
```

- Допускается использование литералов объектов. Например:

```
{ prop1: val1, prop2: val2 }
```

- Функции установщика/получателя литералов объектов *запрещены*. Например:

```
{ get prop1() { ... }, set prop1(v) { ... } }
```

- Допускается использование литералов массивов, например:

```
[ val1, val2, val3 ]
```

- Выражения, включающие в себя считывание свойств, *запрещены*. Например:

```
a.b.c
```

- Вызов функций *запрещен*.
- Определения функций *запрещены*.
- Задание свойств *запрещено*.
- Литералы функций *запрещены*.

Однако во время загрузки кода, до отправки события `onload` и завершения выполнения функции обработчика события `onload`, эти ограничения не действуют для содержимого в изолированной программной среде безопасности приложения.

Например, после того как код уже загружен, при выполнении следующего кода среда выполнения выдает исключение:

```
eval("alert(44)");
eval("myFunction(44)");
eval("NativeApplication.applicationID");
```

Динамически создаваемый код, как, например, тот, что создается при вызове функции `eval()`, представляет собой угрозу безопасности, будучи разрешенным в изолированной программной среде приложения. Например, приложение может случайно выполнить строку, загруженную с сетевого домена и содержащую зловредный код. Это может быть код, который удалит или изменит файлы на компьютере пользователя. Или код, передающий данные о содержимом локального файла недоверенному сетевому домену.

Генерирование динамического кода может осуществляться следующим образом:

- Вызов функции `eval()`.
- Использование свойств `innerHTML` или функций DOM для вставки тегов сценариев, которые загружают сценарий вне каталога приложения.
- Использование свойств `innerHTML` или функций DOM для вставки тегов сценариев со встроенным кодом (вместо загрузки сценария при помощи атрибута `src`).
- Задание атрибута `src` для тегов `script` для загрузки файла JavaScript, находящегося вне каталога приложения.

- Использование схемы URL-адресов `javascript` (как в `href="javascript:alert('Test')"`).
- Использование функции `setInterval()` или `setTimout()` в случаях, когда первый параметр (задающий асинхронное выполнение функции) является строкой (для вычисления), а не именем функции (как в `setTimeout('x = 4', 1000)`).
- Вызов `document.write()` или `document.writeln()`.

Код в изолированной программной среде безопасности приложения может использовать только эти методы во время загрузки содержимого.

Эти ограничения *не* запрещают использование `eval()` с литералами объектов JSON. Это позволяет содержимому приложения работать с библиотекой JavaScript формата JSON. Однако нельзя использовать перегруженный код JSON (с обработчиками событий).

Для других платформ Ajax и библиотек кода JavaScript проверьте, работает ли код платформы или библиотеки при ограничениях, распространяющихся на динамически генерируемые коды. Если коды не работают, поместите любое содержимое, которое использует платформу или библиотеку, в изолированную программную среду, отличную от изолированной программной среды приложения. Дополнительные сведения см. в разделах ««Ограничения для JavaScript внутри AIR» на странице 1126» и ««Выполнение сценариев между содержимым приложения и другим содержимым» на странице 1178». Список платформ Ajax, которые поддерживают изолированную программную среду безопасности приложения, приведен на сайте Adobe http://www.adobe.com/products/air/develop/ajax/features/.

В отличие от содержимого в изолированной программной среде безопасности приложения содержимое JavaScript из других изолированных программных сред безопасности *может* вызвать функцию `eval()` для исполнения динамически генерируемого кода в любое время.

### Ограничения доступа к API-интерфейсам AIR (для изолированных программных сред вне приложения)

**Adobe AIR 1.0 и более поздних версий**

Код JavaScript в изолированной программной среде безопасности, отличной от изолированной программной среды приложения, не имеет доступа к объекту `window.runtime`, поэтому не может исполнять API-интерфейсы AIR. Если содержимое в изолированной программной среде, отличной от изолированной программной среды приложения, вызывает следующий код, приложение выдает исключение TypeError:

```
try {
    window.runtime.flash.system.NativeApplication.nativeApplication.exit();
}
catch (e)
{
    alert(e);
}
```

Исключение относится к типу TypeError (с неопределенным значением), поскольку содержимое в такой изолированной программной среде не распознает объект `window.runtime`, поэтому он воспринимается как неопределенный.

Предоставить функциональность среды выполнения содержимому из изолированной программной среды, отличной от изолированной программной среды приложения, можно при помощи моста сценария. Дополнительные сведения см. в разделе ««Выполнение сценариев между содержимым приложения и другим содержимым» на странице 1178».

### Ограничения на вызов XMLHttpRequest

**Adobe AIR 1.0 и более поздних версий**

Содержимое HTML в изолированной программной среде безопасности приложения не может использовать синхронные методы XMLHttpRequest для загрузки данных из источников вне изолированной программной среды приложения во время загрузки содержимого HTML и события `onLoad`.

По умолчанию содержимое HTML в изолированных программных средах, отличных от изолированной программной среды приложения, не может использовать объект JavaScript XMLHttpRequest для загрузки данных из доменов, отличных от домена, вызывающего запрос. Теги `frame` или `iframe` могут включать в себя атрибут `allowcrosscomainxhr`. Установка для этого атрибута любого другого значения кроме значения null позволяет содержимому в кадре или во встроенном кадре использовать объект JavaScript XMLHttpRequest для загрузки данных из доменов, отличных от домена кода, вызывающего запрос:

```
<iframe id="UI"
    src="http://example.com/ui.html"
    sandboxRoot="http://example.com/"
    allowcrossDomainxhr="true"
    documentRoot="app:/">
</iframe>
```

Дополнительные сведения см. в разделе ««Выполнение сценариев между содержимым из различных доменов» на странице 1172».

### Ограничения по загрузке элементов CSS, frame, iframe и img (для содержимого в изолированных программных средах вне приложения)

**Adobe AIR 1.0 и более поздних версий**

Содержимое HTML в удаленных (сетевых) изолированных программных средах может загружать содержимое CSS, `frame`, `iframe` и `img` только из удаленных изолированных программных сред (с сетевых URL-адресов).

Содержимое HTML в локальной изолированной программной среде файловой системы, локальной изолированной программной среде с сетевым подключением или в локальной доверенной изолированной программной среде может загружать содержимое CSS, frame, iframe и `img` только из локальных изолированных программных сред (не из изолированных программных сред приложения или удаленных изолированных программных сред).

### Ограничения по вызову метода JavaScript window.open()

**Adobe AIR 1.0 и более поздних версий**

Если в окне, созданном путем вызова метода JavaScript `window.open()`, отображается содержимое не из изолированной программной среды приложения, заголовок окна начинается с заголовка основного (запускающего) окна, за которым следует двоеточие. Невозможно удалить эту часть заголовка окна с экрана при помощи кода.

Содержимое не из изолированных программных сред приложения может успешно вызвать метод JavaScript `window.open()` только в ответ на событие, инициированное вводом с мыши или клавиатуры. Это предотвращает создание содержимым, не относящимся к приложению, окон, используемых с целью мошенничества (например, с целью информационных атак). Кроме того, обработчик событий мыши или клавиатуры не может задать выполнение метода `window.open()` после задержки (например, вызвав функцию `setTimeout()`).

Содержимое HTML в изолированных программных средах remote (сетевых) может использовать метод `window.open()` только для открытия содержимого в удаленных сетевых изолированных программных средах. Оно не может использовать метод `window.open()` для открытия содержимого из изолированной программной среды приложения или локальных изолированных программных сред.

Содержимое в локальной изолированной программной среде файловой системы, локальной изолированной программной среде с сетевым подключением или в локальной доверенной изолированной программной среде (см. раздел ««Изолированные программные среды» на странице 1123») может использовать метод `window.open()` только для открытия содержимого в локальных изолированных программных средах. Оно не может использовать метод `window.open()` для открытия содержимого из изолированной программной среды приложения или изолированной программной среды remote.

### Ошибки при вызове кода с ограниченным доступом

**Adobe AIR 1.0 и более поздних версий**

При вызове кода, который нельзя использовать в изолированной программной среде ввиду ограничений безопасности, среда выполнения выдает ошибку JavaScript: «Нарушение защиты среды выполнения Adobe AIR со стороны кода JavaScript в изолированной программной среде безопасности приложения».

Дополнительные сведения см. в разделе ««Избежание ошибок безопасности в JavaScript» на странице 1056».

### Защита изолированной программной среды при загрузке HTML-содержимого из строки

**Adobe AIR 1.0 и более поздних версий**

Метод `loadString()` класса HTMLLoader позволяет создавать HTML-содержимое во время выполнения. Однако данные, используемые в качестве HTML-содержимого, могут быть повреждены, если они будут загружаться из незащищенного источника в Интернете. По этой причине по умолчанию HTML-содержимое, созданное с помощью метода `loadString()`, не помещается в изолированную программную среду приложения и не имеет доступа к прикладным интерфейсам программирования AIR. Но можно установить для свойства `placeLoadStringContentInApplicationSandbox` объекта HTMLLoader значение true, чтобы размещать HTML-содержимое, созданное методом `loadString()`, в изолированную программную среду приложения. Дополнительные сведения см. в разделе ««Загрузка HTML-содержимого из строки» на странице 1055».

## Выполнение сценариев между содержимым из различных доменов

**Adobe AIR 1.0 и более поздних версий**

При установке приложений AIR они наделяются особыми привилегиями. Очень важно, чтобы эти привилегии не были предоставлены другому содержимому, в том числе удаленным файлам и локальным файлам, которые не являются частью приложения.

## О мосте изолированной программной среды в AIR

**Adobe AIR 1.0 и более поздних версий**

Как правило, содержимое из одних доменов не может выполнять сценарии в других доменах. Для защиты приложений AIR от случайной утечки конфиденциальной информации или перехвата управления на содержимое в изолированной программной среде `application` (содержимое, установленное с приложением) накладываются следующие ограничения:

- Код в изолированной программной среде приложения не может предоставить доступ другим изолированным средам путем вызова метода `Security.allowDomain()`. При вызове этого метода из защищенной изолированной программной среды приложения происходит ошибка.
- Импорт непрограммного содержимого в изолированную программную среду приложения путем задания свойства `LoaderContext.securityDomain` или `LoaderContext.applicationDomain` запрещен.

Бывают случаи, когда главному приложению AIR требуется, чтобы содержимое с удаленного домена имело управляемый доступ к сценариям в главном приложении AIR, или наоборот. Для этого среда выполнения создает *мост изолированной программной среды*, который служит шлюзом между двумя изолированными программными средами. Мост изолированной программной среды обеспечивает открытое взаимодействие между изолированной программной средой remote и изолированной программной средой приложения.

Мост изолированной программной среды предоставляет два объекта, доступ к которым имеет и загруженный, и загружаемый сценарий.

- Объект `parentSandboxBridge` позволяет загружаемому содержимому предоставить сценариям загруженного содержимого доступ к свойствам и функциям.
- Объект `childSandboxBridge` позволяет загруженному содержимому предоставить сценариям загружаемого содержимого доступ к свойствам и функциям.

Объекты, доступ к которым предоставлен при помощи моста изолированной программной среды, передаются по значению, а не по ссылке. Все данные сериализуются. Это значит, что объекты, предоставленные одной стороной моста, не могут быть заданы другой стороной, и что все предоставленные объекты нетипизированные. Более того, предоставлять доступ можно только к простым объектам и функциям, а не к сложным.

Если дочернее содержимое пытается задать объект в качестве parentSandboxBridge, среда выполнения выдает исключение SecurityError. Таким же образом, если родительское содержимое пытается задать объект в качестве childSandboxBridge, среда выполнения выдает исключение SecurityError.

## Пример моста изолированной программной среды (SWF)

**Adobe AIR 1.0 и более поздних версий**

Предположим, приложение AIR «музыкальный магазин» хочет разрешить удаленным SWF-файлам сообщать стоимость альбомов, но чтобы удаленный SWF-файл не сообщал о том, что это распродажная цена. Для этого класс StoreAPI предоставляет метод для получения цены, но скрывает распродажную цену. Экземпляр этого класса StoreAPI затем присваивается свойству `parentSandboxBridge` объекта LoaderInfo объекта Loader, который загружает удаленный SWF.

Ниже приводится код для музыкального магазина AIR:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
title="Music Store" creationComplete="initApp()">
    <mx:Script>
        import flash.display.Loader;
        import flash.net.URLRequest;

        private var child:Loader;
        private var isSale:Boolean = false;

        private function initApp():void {
            var request:URLRequest =
                    new URLRequest("http://[www.yourdomain.com]/PriceQuoter.swf")

            child = new Loader();
            child.contentLoaderInfo.parentSandboxBridge = new StoreAPI(this);
            child.load(request);
            container.addChild(child);
        }
        public function getRegularAlbumPrice():String {
            return "$11.99";
        }
        public function getSaleAlbumPrice():String {
            return "$9.99";
        }
        public function getAlbumPrice():String {
            if(isSale) {
                return getSaleAlbumPrice();
            }
            else {
                return getRegularAlbumPrice();
            }
        }
    </mx:Script>
    <mx:UIComponent id="container" />
</mx:WindowedApplication>
```

Объект StoreAPI вызывает главное приложение для получения обычной цены альбома, но возвращает «Нет данных» при вызове метода `getSaleAlbumPrice()`. Следующий код определяет класс StoreAPI:

```
public class StoreAPI
{
    private static var musicStore:Object;

    public function StoreAPI(musicStore:Object)
    {
        this.musicStore = musicStore;
    }

    public function getRegularAlbumPrice():String {
        return musicStore.getRegularAlbumPrice();
    }

    public function getSaleAlbumPrice():String {
        return "Not available";
    }

    public function getAlbumPrice():String {
        return musicStore.getRegularAlbumPrice();
    }
}
```

Следующий код является примером SWF-файла PriceQuoter, который сообщает данные о магазинной цене, но не может сообщить о распродажной цене:

```
package
{
    import flash.display.Sprite;
    import flash.system.Security;
    import flash.text.*;

    public class PriceQuoter extends Sprite
    {
        private var storeRequester:Object;

        public function PriceQuoter() {
            trace("Initializing child SWF");
            trace("Child sandbox: " + Security.sandboxType);
            storeRequester = loaderInfo.parentSandboxBridge;

            var tf:TextField = new TextField();
            tf.autoSize = TextFieldAutoSize.LEFT;
            addChild(tf);

            tf.appendText("Store price of album is: " + storeRequester.getAlbumPrice());
            tf.appendText("\n");
            tf.appendText("Sale price of album is: " + storeRequester.getSaleAlbumPrice());
        }
    }
}
```

### Пример моста изолированной программной среды (HTML)

**Adobe AIR 1.0 и более поздних версий**

В содержимом HTML свойства `parentSandboxBridge` и `childSandboxBridge` добавляются к объекту окна JavaScript дочернего документа. Пример задания функций моста в содержимом HTML см. в разделе «Настройка интерфейса моста изолированной программной среды» на странице 1076.

### Ограничение доступа к API-интерфейсу

**Adobe AIR 1.0 и более поздних версий**

При создании моста изолированной программной среды важно предоставлять доступ к API-интерфейсам высокого уровня, которые ограничивают степень их возможного неправомочного использования. Имейте в виду, что содержимое, вызывающее функцию моста, может быть подвержено риску (например, за счет ввода кода). Так, например, предоставление доступа к методу `readFile(path:String)` (который считывает содержимое произвольного файла) через мост может привести к его неправомочному использованию. В этом случае лучше предоставить доступ к API-интерфейсу `readApplicationSetting()`, который не использует путь и считывает конкретный файл. Более семантический подход снижает вред, который может причинить приложение в случае, если одна из его частей подвергается риску.

### Дополнительные разделы справки

«Перекрестное выполнение сценариев в содержимом из различных изолированных программных сред безопасности» на странице 1074

«Изолированная программная среда приложения AIR» на странице 1125

## Запись на диск

**Adobe AIR 1.0 и более поздних версий**

Приложения, выполняемые в веб-обозревателе, имеют ограничения по взаимодействию с локальной файловой системой пользователя. Веб-обозреватели реализуют политики безопасности, которые обеспечивают защиту компьютера пользователя при загрузке веб-содержимого. Например, SWF-файлы, выполняющиеся в проигрывателе Flash Player в обозревателе, не могут напрямую взаимодействовать с файлами, уже записанными на компьютер пользователя. Общие объекты и файлы cookie могут быть записаны на компьютер пользователя с целью сохранения предпочтений пользователя и других данных, но этим и ограничивается взаимодействие с файловой системой. Поскольку приложения AIR устанавливаются как собственные, к ним применяются другие условия безопасности, которые включают в себя возможность чтения и записи по всей локальной файловой системе.

Такая свобода подразумевает большую ответственность разработчиков. Снижение защиты приложения подвергает риску не только функциональность самого приложения, но и целостность компьютерной системы пользователя. По этой причине разработчики должны ознакомиться с «Рекомендации по обеспечению защиты для разработчиков» на странице 1179.

Разработчики AIR могут получать доступ к файлам и записывать их в локальную файловую систему, используя некоторые правила написания схемы URL-адресов:

| Схема URL-адресов | Описание |
|---|---|
| app:/ | Псевдоним, указывающий на каталог приложения. Файлы, доступ к которым осуществляется по этому пути, входят в изолированную программную среду приложения и имеют полный набор привилегий, предоставленный средой выполнения. |
| app-storage:/ | Псевдоним, указывающий на каталог локального хранилища, стандартизированный средой выполнения. Файлы, доступ к которым осуществляется по этому пути, входят в изолированную программную среду, отличную от изолированной программной среды приложения. |
| file:/// | Псевдоним, указывающий на корневой каталог жесткого диска пользователя. Файлы, доступ к которым осуществляется по этому пути, входят в изолированную программную среду приложения, если существуют в каталоге приложения, в противном случае — в другую изолированную программную среду. |

***Примечание.*** *Приложения AIR не могут изменить содержимое, используя схему URL-адресов app:. Кроме того, считывание из каталога приложения может осуществляться только с использованием настроек администратора.*

Если административные ограничения доступа к компьютеру пользователя отсутствуют, приложения AIR имеют право на запись в любое местоположение на жестком диске пользователя. Разработчикам рекомендуется использовать путь `app-storage:/` для локального хранилища, относящегося к приложению. Файлы, записанные в каталог `app-storage:/` из приложения, помещаются в стандартное местоположение:

- В Mac OS: каталогом хранилища приложения является `<appData>/<appId>/Local Store/`, где `<appData>` — это папка установок пользователя. Как правило, это `/Users/<user>/Library/Preferences`
- В Windows: каталогом хранилища приложения является `<appData>\<appId>\Local Store\`, где `<appData>` — это специальная папка пользователя CSIDL_APPDATA. Как правило, это `C:\Documents and Settings\<userName>\Application Data`
- В Linux: `<appData>/<appID>/Local Store/`, где `<appData>` — это `/home/<user>/.appdata`

Если приложение подразумевает взаимодействие с существующими в файловой системе пользователя файлами, обязательно ознакомьтесь с «Рекомендации по обеспечению защиты для разработчиков» на странице 1179.

## Безопасная работа с недоверенным содержимым

**Adobe AIR 1.0 и более поздних версий**

Содержимое, не входящее в изолированную программную среду приложения, может обеспечить для приложения дополнительную функциональность по выполнению сценариев, но только если оно отвечает критериям безопасности, установленным средой выполнения. В данном разделе объясняются условия безопасности AIR, применимые к содержимому не из изолированной программной среды приложения.

### Security.allowDomain()

**Adobe AIR 1.0 и более поздних версий**

Приложения AIR накладывают более строгие ограничения на выполнение сценариев содержимым не из изолированной среды приложения, чем ограничения, накладываемые подключаемым модулем обозревателя проигрывателя Flash Player на выполнение сценариев с ненадежным содержимым. Например, для проигрывателя Adobe Flash в обозревателе, если SWF-файл, назначенный изолированной программной среде `local-trusted`, вызывает метод `System.allowDomain()`, доступ к сценариям разрешается для всех SWF,

загруженных из указанного домена. Аналогичный подход не реализуется для содержимого `application` в приложениях AIR, так как он предполагает предоставление файлу, не имеющему отношения к приложению, слишком широкого доступа к файловой системе пользователя. Удаленные файлы не могут получить прямой доступ к изолированной программной среде приложения, независимо от вызова метода `Security.allowDomain()`.

## Выполнение сценариев между содержимым приложения и другим содержимым

**Adobe AIR 1.0 и более поздних версий**

Приложения AIR, выполняющие сценарии между содержимым приложения и другим содержимым, имеют более сложную организацию безопасности. Файлы, не входящие в изолированную программную среду приложения, осуществляют доступ к свойствам и методам файлов, входящих в изолированную программную среду приложения, только через мост изолированной программной среды. Мост изолированной программной среды служит шлюзом между содержимым приложения и другим содержимым, обеспечивая открытое взаимодействие между двумя файлами. При правильном использовании мосты изолированной программной среды обеспечивают дополнительную защиту, отказывая содержимому, не имеющему отношения к приложению, в доступе к ссылкам на объекты, которые являются частью содержимого приложения.

Преимущество моста изолированной программной среды лучше продемонстрировать на конкретном примере. Предположим, приложение AIR «музыкальный магазин» хочет предоставить API-интерфейс рекламодателям, создающим собственные SWF-файлы, с которыми впоследствии может взаимодействовать это приложение. Необходимо, чтобы магазин предоставлял рекламодателям методы для поиска в магазине исполнителей и компакт-дисков, но в то же время ограничил доступ к некоторым методам и свойствам для стороннего SWF-файла в целях безопасности.

Подобную функциональность может обеспечить мост изолированной программной среды. По умолчанию содержимое, загруженное в приложение AIR во время выполнения извне, не имеет доступа к методам и свойствам главного приложения. В случае с пользовательской реализацией моста изолированной программной среды разработчик может предоставить удаленному содержимому доступ к службам, предоставив доступ к этим методам и свойствам. Мост изолированной программной среды можно считать мостом между доверенным и недоверенным содержимым, обеспечивающим взаимодействие между содержимым загрузчика и загружаемым содержимым без предоставления доступа к ссылкам на объекты.

Дополнительные сведения о безопасном использовании моста изолированной программной среды см. в разделе ««Выполнение сценариев между содержимым из различных доменов» на странице 1172».

## Защита от динамически генерируемого небезопасного содержимого SWF

**Adobe AIR 1.0 и более поздних версий**

Метод `Loader.loadBytes()` позволяет приложению генерировать содержимое SWF из байтового массива. Однако атаки на данные, загруженные из удаленных источников, могут причинить серьезный вред при загрузке содержимого. Особенно это относится к загрузке данных в изолированную программную среду приложения, в которой сгенерированное содержимое SWF может получить доступ к полному набору API-интерфейсов AIR.

Возможно допустимое использование метода `loadBytes()` без создания исполняемого кода SWF. Например, метод `loadBytes()` можно использовать для формирования данных изображения в целях управления временем вывода изображения. Допустимо также применение этого метода *на основе* исполняемого кода, например динамическое создание содержимого SWF для воспроизведения аудиоданных. По умолчанию метод `loadBytes()` в AIR *не* позволяет загружать содержимое SWF; он позволяет загружать только изображения. В AIR свойство `loaderContext` метода `loadBytes()` имеет свойство `allowLoadBytesCodeExecution`, которое можно установить на значение `true`, чтобы явно разрешить приложению использовать метод `loadBytes()` для загрузки исполняемого содержимого SWF. Следующий код иллюстрирует использование этой функции:

```
var loader:Loader = new Loader();
var loaderContext:LoaderContext = new LoaderContext();
loaderContext.allowLoadBytesCodeExecution = true;
loader.loadBytes(bytes, loaderContext);
```

Если при вызове метода `loadBytes()` для загрузки содержимого SWF свойство `allowLoadBytesCodeExecution` объекта LoaderContext имеет значение `false` (по умолчанию), объект Loader выдает исключение SecurityError.

***Примечание.*** *В будущем выпуске Adobe AIR данный API-интерфейс может измениться. В таком случае может потребоваться повторная компиляция содержимого, использующего свойство `allowLoadBytesCodeExecution` класса LoaderContext.*

## Рекомендации по обеспечению защиты для разработчиков

**Adobe AIR 1.0 и более поздних версий**

Хотя приложения AIR основаны на веб-технологиях, разработчики должны иметь в виду, что эти приложения не работают в изолированной программной среде безопасности обозревателя. Это значит, что можно создать приложения AIR, которые могут причинить преднамеренный или случайный вред локальной системе. AIR старается свести такой риск к минимуму, но приложения все-таки могут иметь уязвимые места. В данном разделе описаны важные потенциальные угрозы защите.

### Риск при импорте файлов в изолированную программную среду безопасности приложения

**Adobe AIR 1.0 и более поздних версий**

Файлы, существующие в каталоге приложения, входят в изолированную программную среду приложения и имеют полный набор привилегий среды выполнения. Приложения, записывающие в локальную файловую систему, как правило, выполняют запись в каталог `app-storage:/`. Этот каталог существует отдельно от файлов приложения на компьютере пользователя, поэтому его файлы не входят в изолированную программную среду приложения и не представляют большой угрозы безопасности. Разработчикам следует учитывать следующие моменты:

- Помещайте файл в файл AIR (в установленное приложение) только при необходимости.
- Помещайте файл сценария в файл AIR (в установленное приложение), только если его поведение является полностью понятным и доверенным.
- Не выполняйте запись в каталог приложения и не изменяйте его содержимое. Среда выполнения запрещает приложениям выполнять запись или изменять файлы и каталоги при помощи схемы URL-адресов `app:/` и выдает при этом исключение SecurityError.

- Не используйте данные из сетевого источника в качестве параметров для методов API-интерфейса AIR, которые могут привести к выполнению кода. Сюда относится использование метода `Loader.loadBytes()` и функции JavaScript `eval()`.

## Риск при использовании внешнего источника для определения путей

**Adobe AIR 1.0 и более поздних версий**

Приложение AIR может быть подвергнуто риску при использовании данных или содержимого из внешнего источника. По этой причине будьте очень осторожны при использовании данных из сети или файловой системы. Ответственность за доверие лежит исключительно на разработчиках и создаваемых ими сетевых подключениях, но загрузка внешних данных изначально опасна. Не рекомендуется использовать ввод таких данных в чувствительных операциях. Разработчикам не рекомендуется следующее:

- Использовать данные из источника в сети для определения имени файла.
- Использовать данные из источника в сети для формирования URL-адреса, по которому приложение отправляет конфиденциальную информацию.

## Риск при использовании, хранении или передаче небезопасных учетных данных

**Adobe AIR 1.0 и более поздних версий**

Хранение учетных данных пользователя в локальной файловой системе пользователя повышает риск их неправомочного использования. Разработчикам следует учитывать следующие моменты:

- Шифруйте учетные данные при их записи в локальную файловую систему, если требуется хранить эти данные локально. Среда выполнения предоставляет зашифрованное хранилище, уникальное для каждого установленного приложения, посредством класса EncryptedLocalStore. Дополнительные сведения см. в разделе ««Зашифрованная локальная система хранения данных» на странице 764».
- Не передавайте незашифрованные учетные данные пользователей сетевому источнику, если он не является надежным и для передачи не используется протокол HTTPS или TLS (Transport Layer Security).
- Никогда не указывайте пароль по умолчанию при создании учетных данных. Позвольте пользователю создать свой пароль. Пользователи, использующие пароль по умолчанию, не изменяя его, подвергаются риску атаки со стороны злоумышленников, которым известен пароль по умолчанию.

## Риск атак на устаревшие версии

**Adobe AIR 1.0 и более поздних версий**

Во время установки приложения среда выполнения проверяет, не установлена ли уже версия приложения. Если данное приложение уже установлено, среда выполнения сравнивает строку версии установленного приложения с версией устанавливаемого приложения. Если версии разные, пользователь может выбрать обновление установки. Среда выполнения не гарантирует, что устанавливаемая версия приложения является более новой, чем предыдущая. Она проверяет только отличие версий. Нарушитель защиты может предложить пользователю устаревшую версию, чтобы ослабить защиту. По этой причине разработчикам рекомендуется осуществлять проверку версии при запуске приложения. Рекомендуется, чтобы приложения выполняли проверку наличия необходимых обновлений в сети. Таким образом, даже если по вине нарушителя защиты пользователь запустит устаревшую версию, при проверке версии выяснится, что требуется ее обновление. Кроме того, использование более ясной схемы определения версии приложения снизит риск установки пользователями устаревшей версии обманным путем.

## Подписание кода

**Adobe AIR 1.0 и более поздних версий**

Все файлы установщика AIR должны иметь подписанный код. Подписание кода представляет собой криптографический процесс подтверждения того, что указанный источник программного обеспечения является достоверным. Приложения AIR можно подписать с помощью сертификата, выпущенного внешним центром сертификации, или с помощью самозаверяющего сертификата, созданного разработчиком. Настоятельно рекомендуется использовать коммерческий сертификат, выданный известной сертификационной организацией. Такой сертификат позволяет пользователям быть уверенными в том, что они устанавливают именно ваше приложение, а не подделку. Однако самозаверяющие сертификаты можно создать либо при помощи `adt` из пакета SDK, либо при помощи Flash, Flash Builder или другого приложения, использующего `adt` для создания сертификатов. Самозаверяющие сертификаты никак не гарантируют, что устанавливаемое приложение является неподдельным. Их следует использовать только для тестирования приложения перед официальным выпуском.

## Безопасность на устройствах Android

**Adobe AIR 2.5 и более новых версий**

На устройствах Android, как и на персональных компьютерах, среда AIR применяет собственную модель безопасности системы. При этом в среде AIR действуют свои правила безопасности, которые разработаны для того, чтобы упростить для разработчиков процесс создания безопасных приложений, использующих подключение к Интернету.

Поскольку приложения AIR на устройствах Android используют пакетный формат Android, установка выполняется в соответствии с моделью безопасности Android. Установщик приложений AIR не используется.

Модель безопасности Android отражает три основных аспекта.

- Разрешения
- Подписи приложений
- Идентификаторы пользователей приложений

**Разрешения Android**

Многие функции Android защищены механизмом полномочий операционной системы. Чтобы воспользоваться защищенной функцией, дескриптор приложения AIR должен указывать, что приложению требуются определенные полномочия. Когда пользователь попытается установить приложение, перед запуском процесса установки операционная система Android покажет пользователю список всех запрашиваемых полномочий.

Для большинства приложений AIR в дескрипторе требуется указать полномочия Android. По умолчанию полномочия не включаются. Для защищенных функций, вызываемых средой выполнения AIR, требуются следующие полномочия:

**ACCESS_COARSE_LOCATION** Разрешает приложению доступ к данным о местоположении, предоставляемым сетью WIFI или сотовой сетью, которые получаются через класс Geolocation.

**ACCESS_FINE_LOCATION:** Разрешает приложению доступ к данным GPS через класс Geolocation.

**ACCESS_NETWORK_STATE и ACCESS_WIFI_STATE:** Разрешает приложению доступ к сведениям о сети через класс NetworkInfo.

**CAMERA:** Разрешает приложению доступ к камере.

**INTERNET:** Разрешает приложению выполнять запросы по сети. Также разрешает удаленную отладку.

**READ_PHONE_STATE:** Разрешает среде выполнения AIR приглушать звук при поступлении входящих вызовов.

**RECORD_AUDIO:** Разрешает приложению доступ к микрофону.

**WAKE_LOCK и DISABLE_KEYGUARD** Разрешает приложению предотвращать переход устройства в режим сна с помощью параметров класса SystemIdleMode.

**WRITE_EXTERNAL_STORAGE:** Разрешает приложению выполнять запись на внешнюю карту памяти устройства.

### Подписи приложений

Все пакеты приложений, создаваемые для платформы Android, должны быть подписаны. Приложения AIR на Android упаковываются с применением собственного формата Android APK, поэтому они подписываются в соответствии с правилами Android, а не правилами AIR. Несмотря на то что Android и AIR используют подписание кода схожим образом, также имеются значительные отличия.

- На Android подпись подтверждает, что закрытый ключ находится во владении разработчика, но не используется для проверки личности разработчика.
- Срок действия сертификата приложений, добавляемых на Android Маркет, должен составлять не менее 25 лет.
- Android не поддерживает перенос подписи пакета в другой сертификат. Если обновление подписано другим сертификатом, пользователь должен удалить исходное приложение, прежде чем устанавливать обновленную версию.
- Два приложения, подписанные с использованием одного и того же сертификата, могут содержат общий идентификатор, который позволит каждому из этих приложений получить доступ к кэшу и файлам данных другого приложения (AIR не позволяет организовать такой обмен ).

### Идентификаторы пользователей приложений

На Android используется ядро Linux. Каждому устанавливаемому приложению назначается идентификатор пользователя Linux, который определяет полномочия приложения для выполнения таких операций, как доступ к файлам. Доступ к файлам приложения, в хранилище приложений и временных каталогах защищен с помощью разрешений файловой системы. Чтение, изменение и удаление файлов во внешнем хранилище (то есть на SD-карте) могут выполнять другие приложения или пользователь, если SD-карта подключена к компьютеру как запоминающее устройство.

Приложения AIR не имеют общего доступа к файлам cookie, полученным при выполнении интернет-запросов.

### Конфиденциальность фонового изображения

Когда пользователь переключает приложение в фоновый режим, некоторые версии Android захватывают снимок экрана, который используется в качестве миниатюры в списке недавно использованных приложений. Этот снимок экрана сохраняется в памяти устройства и доступен взломщику при физическом управлении устройством.

Если в программе отображается конфиденциальная информация, следует предусмотреть для нее защиту от попадания на фоновый снимок экрана. Событие `deactivate`, отправляемое объектом NativeApplication, сигнализирует о готовности программы переключиться в фоновый режим. Используйте это событие для удаления или скрытия любой конфиденциальной информации.

### Дополнительные разделы справки

Android: безопасность и разрешения

### Зашифрованные данные на Android

Приложения AIR на устройствах Android могут шифровать данные при сохранении, используя настройки шифрования, доступные во встроенной базе данных SQL. Для оптимальной безопасности следует использовать ключ шифрования на базе пользовательского пароля, который вводится при каждом запуске приложения. Локально сохраненный ключ шифрования или пароль трудно или невозможно скрыть от злоумышленника, который имеет доступ к файлам приложения. Если злоумышленник сможет извлечь ключ, то шифрование данных не обеспечит дополнительной защиты, помимо безопасности файловой системы на базе идентификаторов пользователя, которая встроена в ОС Android.

Класс EncryptedLocalStore можно использовать для хранения данных, но эти данные не шифруются на устройствах Android. Вместо этого модель безопасности Android основана на использовании присваиваемого приложению идентификатора пользователя, который служит для защиты данных от использования другими приложениями. Приложения, использующие общий идентификатор пользователя и подписанные одним сертификатом для подписи кода, пользуются одним зашифрованным локальным хранилищем.

***Важная информация.*** *На взломанном телефоне приложения, выполняющиеся с использованием прав суперпользователя, могут получать доступ к файлам любого приложения. Таким образом, данные, находящиеся в зашифрованном локальном хранилище, не защищены на взломанных устройствах.*

## Безопасность на устройствах iOS

В ОС iOS AIR использует собственную систему безопасности. При этом в среде AIR действуют свои правила безопасности, которые разработаны для того, чтобы упростить для разработчиков процесс создания безопасных приложений, использующих подключение к Интернету.

Поскольку приложения AIR на устройствах iOS используют пакетный формат iOS, установка выполняется в соответствии с моделью безопасности iOS. Установщик приложений AIR не используется. Кроме того, на устройствах iOS не используется отдельная среда выполнения AIR. Каждое приложение AIR содержит весь код, необходимый для работы.

#### Подписи приложений

Все пакеты приложений, создаваемые для платформы iOS, должны быть подписаны. Поскольку приложения AIR для iOS упаковываются в собственном формате IPA, они подписываются в соответствии с требованиям iOS, а не с требованиями AIR. Хотя iOS и AIR используют подписание кода схожим образом, есть и значительные различия.

- В ОС iOS сертификат, используемый для подписи приложения, должен выдаваться компанией Apple. Нельзя использовать сертификаты, выданные другими центрами сертификации.
- В ОС iOS сертификаты, выданные Apple, обычно действительны в течение одного года.

#### Конфиденциальность фонового изображения

Когда пользователь переключает программу в фоновый режим в ОС iOS, операционная система получает снимок экрана, используемый для анимации перехода. Этот снимок экрана сохраняется в памяти устройства и доступен взломщику при физическом управлении устройством.

Если в программе отображается конфиденциальная информация, следует предусмотреть для нее защиту от попадания на фоновый снимок экрана. Событие `deactivate`, отправляемое объектом NativeApplication, сигнализирует о готовности программы переключиться в фоновый режим. Используйте это событие для удаления или скрытия любой конфиденциальной информации.

# Глава 65. Работа с примерами ActionScript

Выполнение примера кода ActionScript 3.0 — это один из лучших способов узнать, как работают определенные классы и методы. Примеры можно использовать по-разному в зависимости от того, какое устройство используется или планируется использовать.

**Компьютеры с Flash Professional или Flash Builder** Сведения об использовании этих сред разработки для выполнения примеров кода ActionScript 3.0 см. в «Работа с примерами ActionScript 3.0 во Flash Professional» на странице 1187 или «Работа с примерами ActionScript 3.0 в Flash Builder» на странице 1188. Используйте инструкции trace и другие средства отладки, чтобы глубже понять принцип работы примера кода.

**Мобильные устройства** Примеры кодов ActionScript 3.0 можно выполнять на мобильных устройствах, которые поддерживают Flash Player 10.1 и более поздние версии. См. раздел «Работа с примерами ActionScript 3.0 на мобильных устройствах» на странице 1189. Эти примеры кода также можно выполнять на компьютере с Flash Professional или Flash Builder.

**Телевизионные устройства** Хотя эти примеры кода нельзя выполнять на телевизионных устройствах, все равно можно извлечь полезную информацию, выполнив их на компьютере. Сведения о разработке программ для телевизионных устройств см. в разделе Платформа Flash для ТВ на веб-сайте Adobe Developer Connection.

## Типы примеров

Представлены следующие типы примеров кодов ActionScript 3.0:

- Примеры фрагментов кода (приводятся во всей документации по ActionScript 3.0)
- Примеры на базе классов (приводятся в основном в Справочнике по ActionScript 3.0)
- Практические примеры, содержащие несколько исходных файлов (загрузите исходные ZIP-файлы со страницы www.adobe.com/go/learn_programmingAS3samples_flash_ru)

### Примеры с фрагментами кода

Пример с фрагментом кода выглядит следующим образом:

```
var x:int = 5;
trace(x); // 5
```

Фрагменты кода содержат только ту часть кода, которая требуется для демонстрации одного понятия. Как правило, они не содержат инструкций с пакетами или классами.

### Примеры на базе классов

Многие примеры содержат исходный код для полного класса ActionScript. Пример на базе класса выглядит следующим образом:

```
package {
    public class Example1 {
        public function Example1():void {
            var x:int = 5;
            trace(x); //5
        }
    }
}
```

В примере на базе класса код включает инструкцию пакета, объявление класса и функцию конструктора.

**Практические примеры, содержащие несколько исходных файлов**

Во многих разделах «Руководства разработчика по ActionScript 3.0» приводятся практические примеры, демонстрирующие использование определенных возможностей ActionScript в прикладном, жизненном контексте. Такие примеры обычно содержат несколько файлов, включая:

- один или несколько исходных файлов ActionScript;
- файл .FLA для использования в Flash Professional;
- один или несколько файлов MXML для работы в Flash Builder;
- файлы с данными, файлы изображений, звуковые файлы и другие файлы, используемые в приложении (необязательно).

Практические примеры обычно предоставляются в виде архивных ZIP-файлов.

Список примеров, используемых в руководстве для разработчиков и включенных в ZIP-файл

ZIP-файл для Flash Professional CS5 и Flex 4 (загрузите со страницы www.adobe.com/go/learn_programmingAS3samples_flash_ru) содержит следующие примеры:

- AlarmClock («Пример обработки событий: будильник» на странице 151)
- AlgorithmicVisualGenerator («Пример API рисования: наглядный алгоритмический генератор» на странице 250)
- ASCIIArt («Пример строк: ASCII-графика» на странице 20)
- CapabilitiesExplorer («Пример возможностей: определение возможностей системы» на странице 937)
- CustomErrors («Пример обработки ошибок: программа CustomErrors» на странице 75)
- DisplayObjectTransformer («Пример геометрии: применение матричного преобразования к экранному объекту» на странице 235)
- FilterWorkbench («Пример фильтрации экранных объектов: Filter Workbench» на странице 315)
- GlobalStockTicker («Пример. Интернационализация приложения, передающего котировки ценных бумаг» на странице 1025)
- IntrovertIM_HTML («Пример внешнего API: связь между ActionScript и JavaScript в веб-обозревателе» на странице 911)
- NewsLayout («Пример класса TextField: форматирование текста в газетном стиле» на странице 419)
- PlayList («Пример массивов: PlayList» на странице 50)
- PodcastPlayer («Пример звука: Podcast Player» на странице 501)
- ProjectionDragger («Пример: перспективная проекция» на странице 386)
- ReorderByZ («Использование объектов Matrix3D для изменения порядка отображения» на странице 391)
- RSSViewer («Пример XML в ActionScript: загрузка данных RSS из Интернета» на странице 122)

- RuntimeAssetsExplorer («Пример фрагмента ролика: RuntimeAssetsExplorer» на странице 357)
- SimpleClock («Пример даты и времени: простые аналоговые часы» на странице 6)
- SpinningMoon («Пример растрового изображения: анимация вращающейся луны» на странице 275)
- SpriteArranger («Пример экранного объекта: SpriteArranger» на странице 220)
- TelnetSocket («Пример сокета TCP: создание клиента Telnet» на странице 859)
- VideoJukebox («Пример видеоданных: Video Jukebox» на странице 543)
- WikiEditor («Пример регулярных выражений: синтаксический анализатор Wiki» на странице 98)
- WordSearch («Пример действий мыши: WordSearch» на странице 620)

Практические пример также можно найти во многих статьях по быстрому началу работы в центрах разработчиков Flash Developer Center и Flex Developer Center.

# Работа с примерами ActionScript 3.0 во Flash Professional

Используйте одну из следующих процедур (в зависимости от типа примера), чтобы выполнить пример кода с помощью Flash Professional.

**Выполнение примера с фрагментом кода в Flash Professional**

Чтобы выполнить пример с фрагментом кода в Flash Professional, выполните следующие действия.

1. Выберите команды «Файл» > «Создать».
2. В диалоговом окне «Создать документ» выберите документ Flash и нажмите кнопку «OK».

   После этого появится окно нового документа Flash.
3. Щелкните первый кадр первого слоя на панели «Временная шкала».
4. На панели «Действия» введите или вставьте фрагмент кода из примера.
5. Выберите команды «Файл» > «Сохранить». Присвойте файлу имя и нажмите кнопку «OK».
6. Чтобы проверить пример, выберите пункт «Управление» > «Тестировать ролик».

**Выполнение примера на базе класса в Flash Professional**

Чтобы выполнить пример на базе класса в Flash Professional, выполните следующие действия.

1. Выберите команды «Файл» > «Создать».
2. В диалоговом окне «Новый документ» выберите «Файл ActionScript» и нажмите кнопку «OK». После этого откроется окно редактора.
3. Скопируйте код примера на базе класса и вставьте его в окно редактора.

   Если класс является основным классом документа для программы, он должен расширять класс MovieClip:

```
import flash.display.MovieClip;
public class Example1 extends MovieClip{
//...
}
```

Также следует убедиться, что все классы, на которые имеются ссылки в примере, объявлены с использованием инструкций `import`.

4 Выберите команды «Файл» > «Сохранить». Присвойте файлу имя, совпадающее с именем класса, использованного в примере (например, ContextMenuExample.as).

***Примечание.*** *Некоторые примеры класса, например пример класса flashx.textLayout.container.ContainerController, имеют несколько уровней объявления пакета (`package flashx.textLayout.container.examples {`). В таком случае сохраните файл в подпапку, соответствующую объявлению пакета (flashx/textLayout/container/examples) или удалите имя пакета (код ActionScript должен начинаться с текста `package {` ), чтобы тестировать файл из любого расположения.*

5 Выберите команды «Файл» > «Создать».

6 В диалоговом окне «Новый документ» выберите «Документ Flash (ActionScript 3.0)» и нажмите кнопку «OK». После этого появится окно нового документа Flash.

7 На панели «Свойства» в поле «Класс документа» введите названия класса, которое используется в примере и соответствует названию сохраненного ранее исходного файла ActionScript (например, ContextMenuExample).

8 Выберите команды «Файл» > «Сохранить». Присвойте FLA-файлу имя, совпадающие с именем класса, использованного в примере (например, ContextMenuExample.as).

9 Чтобы проверить пример, выберите пункт «Управление» > «Тестировать ролик».

### Выполнение практического примера в Flash Professional

Практические примеры обычно предоставляются в виде архивных ZIP-файлов. Чтобы выполнить практический пример в Flash Professional, выполните следующие действия.

1 Распакуйте архивный файл в любую папку на свой выбор.

2 В инструменте Flash Professional выберите «Файл» > «Открыть».

3 В окне обзора выберите папку, в которую распакован архивный файл. Выберите файл FLA в этой папке и нажмите кнопку «Открыть».

4 Чтобы проверить пример, выберите пункт «Управление» > «Тестировать ролик».

## Работа с примерами ActionScript 3.0 в Flash Builder

Используйте одну из следующих процедур (в зависимости от типа примера), чтобы выполнить пример кода с помощью Flash Builder.

### Выполнение примера с фрагментом кода в Flash Builder

Чтобы выполнить пример с фрагментом кода в Flash Builder, выполните следующие действия.

1 Можно создать новый проект Flex (выберите «Файл» > «Создать» > «Проект Flex») или создать новое приложение MXML внутри существующего проекта Flex (выберите «Файл» > «Создать» > «Приложение MXML»). Присвойте проекту или программе описательное имя (например, ContextMenuExample).

2 В созданном файле MXML добавьте тег `<mx:Script>`.

3 Вставьте содержимое фрагмента кода между тегами `<mx:Script>` и `</mx:Script>`. Сохраните файл MXML.

4 Чтобы выполнить код, откройте меню «Выполнить» и выберите команду «Выполнить» для основного файла MXML (например, «Выполнить» > «Выполнить ContextMenuExample»).

**Выполнение примера на базе класса в Flash Builder**

Чтобы выполнить пример на базе класса в Flash Builder, выполните следующие действия.

1 Выберите команды «Файл» > «Создать» > «Проект ActionScript».

2 Введите имя основного класса (например, ContextMenuExample) в поле «Имя проекта». Для остальных полей используйте значения по умолчанию (или измените их в соответствии с требованиями конкретной среды). Нажмите кнопку «Готово», чтобы создать проект и основной файл ActionScript.

3 Удалите все созданное содержимое из файла ActionScript. Вставьте код примера, включая инструкции пакета и импорта, в файл ActionScript и сохраните его.

   ***Примечание.*** *Некоторые примеры класса, например пример класса flashx.textLayout.container.ContainerController, имеют несколько уровней объявления пакета (`package flashx.textLayout.container.examples {`). В таком случае сохраните файл в подпапку, соответствующую объявлению пакета (flashx/textLayout/container/examples) или удалите имя пакета (код ActionScript должен начинаться с текста `package {` ), чтобы тестировать файл из любого расположения.*

4 Чтобы выполнить код, откройте меню «Выполнить» и выберите команду «Выполнить» для основного класса ActionScript (например, «Выполнить» > «Выполнить ContextMenuExample»).

**Выполнение практического примера во Flash Builder**

Практические примеры обычно предоставляются в виде архивных ZIP-файлов. Чтобы выполнить практический пример в Flash Builder, выполните следующие действия.

1 Распакуйте архивный файл в любую папку на свой выбор. Присвойте папке описательное имя (например, ContextMenuExample).

2 В Flash Builder выберите «Файл» > «Создать» > «Проект Flex». В области «Местоположение проекта» щелкните «Обзор» и выберите папку, содержащую файлы примера. В поле «Имя проекта» введите имя папки (например, ContextMenuExample). Для остальных полей используйте значения по умолчанию (или измените их в соответствии с требованиями конкретной среды). Нажмите «Далее», чтобы продолжить.

3 На панели «Вывод» щелкните «Далее», чтобы принять значение по умолчанию.

4 На панели путей к исходным файлам щелкните «Обзор» рядом с полем «Основной файл приложения». Выберите основной файл MXML примера из папки. Нажмите «Готово», чтобы создать файлы проекта.

5 Чтобы выполнить код, откройте меню «Выполнить» и выберите команду «Выполнить» для основного файла MXML (например, «Выполнить» > «Выполнить ContextMenuExample»).

# Работа с примерами ActionScript 3.0 на мобильных устройствах

Примеры кодов ActionScript 3.0 можно выполнять на мобильных устройствах, которые поддерживают Flash Player 10.1. Однако, как правило, код выполняется, чтобы проверить работу определенных классов и методов. В таком случае выполните пример на стационарном устройстве, например на персональном компьютере. На персональном компьютере можно использовать инструкции trace и другие инструменты отладки в Flash Professional или Flash Builder для лучшего понимания образца кода.

Если все же требуется выполнить пример на мобильном устройстве, можно скопировать файл на устройство или на веб-сервер. Чтобы скопировать файлы на устройство и выполнить код в браузере, выполните следующие действия.

1. Создайте SWF-файл, следуя инструкциям в разделе «Работа с примерами ActionScript 3.0 во Flash Professional» на странице 1187 или «Работа с примерами ActionScript 3.0 в Flash Builder» на странице 1188. В Flash Professional SWF-файл создается при выборе «Управление» > «Тестировать ролик». В Flash Builder SWF-файл создается при выполнении, отладке или создании проекта Flash Builder.
2. Скопируйте SWF-файл в каталог на мобильном устройстве. Используйте для этого программные средства устройства.
3. В адресной строке браузера на мобильном устройстве введите URL с префиксом file:// для SWF-файла. Например введите `file:://applications/myExample.swf`.

Чтобы скопировать файлы на веб-сервер и выполнить код в браузере устройства, выполните следующие действия.

1. Создайте SWF-файл и HTML-файл. Сначала создайте SWF-файл, следуя инструкциям в разделе «Работа с примерами ActionScript 3.0 во Flash Professional» на странице 1187 или «Работа с примерами ActionScript 3.0 в Flash Builder» на странице 1188. В Flash Professional при выборе «Управление» > «Тестировать ролик» создается только SWF-файл. Чтобы создать оба файла, сначала установите флажки Flash и HTML на вкладке «Форматы» в диалоговом окне «Параметры публикации». Затем выберите «Файл» > «Опубликовать», чтобы создать сразу оба файла: HTML и SWF. В Flash Builder при выполнении, отладке или создании проекта Flash Builder создаются сразу и SWF-файл и HTML-файл.
2. Скопируйте SWF-файл и HTML-файл в каталог на веб-сервере.
3. В адресной строке браузера на мобильном устройстве введите URL с префиксом http:// для HTML-файла. Например, введите `http://www.myWebServer/examples/myExample.html`.

Перед выполнением примера на мобильном устройстве просмотрите список потенциальных проблем.

### Размер рабочей области

Размер используемой рабочей области при выполнении примеров на мобильном устройстве намного меньше, чем при работе на стационарном устройстве. Для многих примеров не требуется определенного размера рабочей области. При создании SWF-файла укажите размер рабочий области в соответствии с характеристиками используемого устройства. Например, укажите 176 x 208 пикселов.

Практические примеры в «Руководстве разработчика по ActionScript 3.0» предназначены для иллюстрации различных понятий и классов ActionScript 3.0. Реализованные в них интерфейсы предназначены для корректной работы и хорошей наглядности на персональном или портативном компьютере. Хотя примеры работают на мобильных устройствах, размер рабочей области и дизайн пользовательского интерфейса не походят для маленьких экранов. Adobe рекомендует выполнять практические примеры на компьютере с целью изучения ActionScript, а затем использовать соответствующие фрагменты кода в мобильных приложениях.

### Текстовые поля вместо инструкций trace

При выполнении примера на мобильном устройстве нет возможности увидеть результаты использования инструкций trace в примере. Чтобы увидеть вывод, создайте экземпляр класса TextField. После этого добавьте текст из инструкций trace в свойство `text` текстового поля.

Чтобы создать текстовое поле для отслеживания вывода, можно использовать следующую функцию.

```
function createTracingTextField(x:Number, y:Number,
                                width:Number, height:Number):TextField {

    var tracingTF:TextField = new TextField();
    tracingTF.x = x;
    tracingTF.y = y;
    tracingTF.width = width;
    tracingTF.height = height;

    // A border lets you more easily see the area the text field covers.
    tracingTF.border = true;
    // Left justifying means that the right side of the text field is automatically
    // resized if a line of text is wider than the width of the text field.
    // The bottom is also automatically resized if the number of lines of text
    // exceed the length of the text field.
    tracingTF.autoSize = TextFieldAutoSize.LEFT;

    // Use a text size that works well on the device.
    var myFormat:TextFormat = new TextFormat();
    myFormat.size = 18;
    tracingTF.defaultTextFormat = myFormat;

    addChild(tracingTF);
    return tracingTF;
}
```

Например, добавьте эту функцию в класс документа в качестве частной функции. Затем в других методах класса документа отслеживайте данные с помощью подобного кода:

```
var traceField:TextField = createTracingTextField(10, 10, 150, 150);
// Use the newline character "\n" to force the text to the next line.
traceField.appendText("data to trace\n");
traceField.appendText("more data to trace\n");
// Use the following line to clear the text field.
traceField.appendText("");
```

Метод `appendText()` принимает в качестве параметра только одно значение. Это значение является строкой (экземпляр String или строковый литерал). Для распечатки значения нестроковой переменной необходимо сначала преобразовать его в строку. Легче всего сделать это, вызвав метод объекта `toString()`:

```
var albumYear:int = 1999;
traceField.appendText("albumYear = ");
traceField.appendText(albumYear.toString());
```

### Размер шрифта

Во многих примерах используются текстовые поля для иллюстрации понятий. В некоторых случаях изменение размера шрифта в текстовом поле облегчает чтение на мобильном устройстве. Например, если в примере используется экземпляр TextField с именем `myTextField`, измените для него размер шрифта с помощью следующего кода.

```
// Use a text size that works well on the device.
var myFormat:TextFormat = new TextFormat();
myFormat.size = 18;
myTextField.defaultTextFormat = myFormat
```

**Захват действий пользователя**

Операционная система и браузер мобильного устройства захватывают некоторые события пользовательского ввода, на которые не реагирует SWF-содержимое. Поведение в каждом конкретном случае зависит от операционной системы и браузера, что может привести к неожиданному поведению примеров, выполняемых на мобильных устройствах. Дополнительные сведения см. в разделе ««Приоритет событий KeyboardEvent» на странице 604».

Кроме того, во многих примерах пользовательские интерфейсы разработаны для персонального или портативного компьютера. Так, большинство практических примеров в «Руководстве разработчика по ActionScript 3.0» предназначены для просмотра на персональном компьютере. Поэтому в некоторых случаях вся рабочая не помещается на экране мобильного устройства. Возможность прокрутки содержимого в окне может поддерживаться не всеми браузерами. Более того, в примерах не предусмотрена обработка событий прокрутки и панорамирования. Поэтому в некоторых примерах пользовательский интерфейс не подходит для работы с маленьким экраном. Adobe рекомендует выполнять примеры на компьютере с целью изучения ActionScript, а затем использовать соответствующие фрагменты кода в мобильных приложениях.

Дополнительные сведения см. в разделе ««Панорамирование и прокрутка экранных объектов» на странице 192».

**Обработка фокуса**

В некоторых примерах требуется перевести фокус в то или иное поле. Это может потребоваться, например, для ввода текста или выбора кнопки. Чтобы перевести фокус в поле, используйте указатель мобильного устройства, например перо или палец. Для перевода фокуса в поле также можно использовать клавиши навигации на мобильном устройстве. Чтобы нажать кнопку, которая находится в фокусе, используйте клавишу выбора на мобильном устройстве, выполняющую ту же функцию, что клавиша Enter на компьютере. На некоторых устройствах кнопку можно выбрать двойным нажатием.

Дополнительные сведения о фокусе см. в разделе ««Управление фокусом» на странице 598».

**Обработка событий мыши**

Во многих примерах прослушиваются события мыши. На компьютере эти события происходят, например, когда пользователь наводит мышь на экранный объект или нажимает кнопку мыши над экранным объектом. На мобильных устройствах события использования указателя, такого как перо или пальца, называются событиями прикосновения. Flash Player 10.1 сопоставляет события прикосновения с соответствующими событиями мыши. Благодаря сопоставлению SWF-содержимое, разработанное до появления Flash Player 10.1 продолжает работать. Это обеспечивает правильную работы примеров при выделении или перетаскивании экранных объектов с помощью указателя.

**Производительность**

Мобильные устройства менее производительны, чем стационарные устройства. Некоторые примеры, потребляющие значительные ресурсы ЦП, могут медленно выполняться на мобильных устройствах. Так, в примере «Пример API рисования: наглядный алгоритмический генератор» на странице 250 выполняются обширные вычисления и отрисовка при смене каждого кадра. Выполнение этого примера на компьютере демонстрирует работу различных API-интерфейсов рисования. Однако этот пример не подходит для некоторых мобильных устройств вследствие их ограниченной производительности.

Дополнительные сведения о производительности на мобильных устройствах см. в разделе «*Оптимизация производительности для платформы Flash Platform*».

**Передовой опыт**

В примерах не учитываются оптимальные методы разработки приложения для мобильных устройств В ходе разработки необходимо учитывать ограничения оперативной памяти и процессора мобильных устройств. Подобным образом, при разработке пользовательского интерфейса для маленького экрана предъявляются не такие требования, как при создании приложения для монитора компьютера. Дополнительные сведения о разработке приложений для мобильных устройств *см. в разделе «Оптимизация производительности для платформы Flash Platform»*.

# Глава 66. Поддержка SQL в локальных базах данных

Adobe AIR включает ядро базы данных SQL, которое обеспечивает поддержку локальных баз данных SQL, включая многие стандартные функции SQL, посредством использования открытого исходного кода системы базы данных SQLite. Среда выполнения не определяет способ и место хранения данные баз данных в файловой системе. Каждая база данных полностью хранится в одном файле. Разработчик сам задает местоположение в файловой системе для хранения файла базы данных. Одно приложение AIR может обращаться к одной или нескольким отдельным базам данных (то есть к отдельным файлам баз данных). В этом документе приводятся общие сведения о синтаксисе SQL, а также о поддержке типов данных для локальных баз данных SQL в Adobe Air. Этот документ не является полным справочником по SQL. В нем описываются специфические сведения о диалекте SQL, который поддерживается в Adobe AIR. Среда выполнения почти полностью поддерживает стандартный диалект SQL-92. Так как существует большое количество справочников, веб-сайтов, книг и учебных материалов по SQL, при создании этого документа мы не преследовали цель сделать его исчерпывающим справочником или учебником по SQL. В нем освещаются только те аспекты синтаксиса SQL, которые поддерживаются средой выполнения AIR, а также различия между диалектом SQL-92 и поддерживаемым диалектом SQL.

Условные обозначения в определениях инструкций SQL

В определениях инструкций, которые встречаются в этом документе, используются следующие условные обозначения.

- Регистр текста
  - ВЕРХНИЙ РЕГИСТР: ключевые слова SQL литерала пишутся прописными буквами.
  - нижний регистр: шаблонные термины или названия предложений пишутся строчными буквами.
- Символы в определениях
  - ::= обозначает определение предложения или инструкции.
- Символы группировки и чередования
  - | Символ прямой черты используется между альтернативными вариантами и может читаться «или».
  - [] Элементы в квадратных скобках являются необязательными. В квадратные скобки может быть заключен один элемент или набор альтернативных элементов.
  - () Если набор альтернативных элементов (набор элементов, разделенных вертикальными линиями) заключен в круглые скобки, значит, это обязательная группа элементов, которые являются возможными значениями для одного обязательного элемента.
- Квантификаторы
  - + Символ плюса, который следует за элементом в круглых скобках, обозначает, что предшествующий элемент может встречаться один или несколько раз.
  - * Символ звездочки, который следует за элементом в квадратных скобках, обозначает, что предшествующий элемент может встречаться 0 или больше раз.
- Символы литерала
  - * Символ звездочки в имени столбца или между круглыми скобками после имени функции обозначает символ звездочки литерала, а не квантификатор «0 или больше».

- . Символ точки представляет точку литерала.
- , Символ запятой обозначает запятую литерала.
- () Круглые скобки вокруг одного предложения или элемента обозначают, что эти скобки являются обязательными символами скобок литерала.
- Остальные символы представляют соответствующие символы литерала, если не указано иное.

# Поддерживаемый синтаксис SQL

Следующие примеры синтаксиса SQL поддерживаются ядром базы данных SQL в Adobe AIR. Примеры содержат объяснения разных типов инструкций и предложений, выражений, встроенных функций и операторов. Рассматриваются следующие темы:

- Общий синтаксис SQL
- Инструкции обработки данных (SELECT, INSERT, UPDATE и DELETE)
- Инструкции определения данных (инструкции CREATE, ALTER и DROP для таблиц, индексов, видов и триггеров)
- Специальные инструкции и предложения
- Встроенные функции (агрегатные, скалярные и функции форматирования даты и времени)
- Операторы
- Параметры
- Неподдерживаемые возможности SQL
- Дополнительные возможности SQL

## Общий синтаксис SQL

Помимо специфического синтаксиса для различных инструкций и выражений применяются следующие общие правила синтаксиса SQL.

**Чувствительность к регистру** В инструкциях SQL, включая имена объектов, не учитывается регистр. Тем не менее, инструкции SQL часто записываются с использованием ключевых слов в верхнем регистре, поэтому данный документ следует этому правилу. Хотя в синтаксисе SQL регистр не учитывается, он учитывается в текстовых значениях литералов SQL. Кроме того, регистр может иметь значение при выполнении операций сравнения и сортировки в соответствии с последовательностью сортировки, определенной для столбца или операции. Дополнительные сведения см. в разделе «COLLATE».

**Пробел** Символ пробела (например, обычный пробел, разделитель табуляции, новая строка и т. д.) следует использовать для разделения отдельных слов в инструкции SQL. Однако пробелом не обязательно разделять слова и символы. Тип и количество символов пробела в инструкции SQL значения не имеют. Пробелы, такие как отступы и разрывы строк, можно использовать для форматирования инструкций SQL с целью удобочитаемости, при этом они не будут оказывать влияния на значение инструкции.

## Инструкции обработки данных

Инструкции обработки данных являются самыми распространенными из всех инструкций SQL. Эти инструкции предназначены для получения, сложения, изменения и удаления данных в таблицах базы данных. Поддерживаются следующие инструкции обработки данных: SELECT, INSERT, UPDATE и DELETE.

### SELECT

Инструкция SELECT используется для запроса информации в базе данных. В качестве ответа на SELECT возвращается 0 или больше строк данных, где каждая строка содержит фиксированное число столбцов. Число столбцов в результате определяется по списку имен столбцов или выражений в результате, который находится между SELECT и необязательными ключевыми словами FROM.

```
sql-statement   ::=  SELECT [ALL | DISTINCT] result
                     [FROM table-list]
                     [WHERE expr]
                     [GROUP BY expr-list]
                     [HAVING expr]
                     [compound-op select-statement]*
                     [ORDER BY sort-expr-list]
                     [LIMIT integer [( OFFSET | , ) integer]]
result          ::=  result-column [, result-column]*
result-column   ::=  * | table-name . * | expr [[AS] string]
table-list      ::=  table [ join-op table join-args ]*
table           ::=  table-name [AS alias] |
                     ( select ) [AS alias]
join-op         ::=  , | [NATURAL] [LEFT | RIGHT | FULL] [OUTER | INNER | CROSS] JOIN
join-args       ::=  [ON expr] [USING ( id-list )]
compound-op     ::=  UNION | UNION ALL | INTERSECT | EXCEPT
sort-expr-list  ::=  expr [sort-order] [, expr [sort-order]]*
sort-order      ::=  [COLLATE collation-name] [ASC | DESC]
collation-name  ::=  BINARY | NOCASE
```

В качестве результата может использоваться произвольное выражение. Если результирующим выражением является символ «*», то все столбцы всех таблиц заменяются одним этим выражением. Если выражение содержит имя таблицы после «.*»,то результатом будут все столбцы в этой таблице.

Ключевое слово DISTINCT возвращает подмножество строк, каждая из которых отлична от остальных. Значения NULL не считаются отличными друг от друга. По умолчанию возвращаются все строки результата, этого также можно достичь при помощи ключевого слова ALL.

Запрос выполняется по одной или нескольким таблицам, имена которых следуют за ключевым словом FROM. Если несколько имен таблиц разделены запятыми, то при выполнении запроса используется перекрестное объединение различных таблиц. Чтобы определить способ объединения таблиц, также можно использовать синтаксис JOIN. Единственными поддерживаемым типом внешнего объединения является LEFT OUTER JOIN. Выражение предложения ON в join-args должно разрешаться в логическое значение. Подзапрос в круглых скобках может использоваться как таблица в предложении FROM. Предложение FROM можно полностью опустить. В таком случае возвращается одна строка, содержащая значения результирующего списка выражений.

Предложение WHERE служит для ограничения числа строк, извлекаемых с помощью запроса. Выражения предложения WHERE должны разрешаться в логическое значение. Фильтрация с помощью предложения WHERE выполняется перед группировкой, потому выражения WHERE не могут включать агрегатные функции.

Предложения GROUP BY служат для объединения одной или нескольких строк результата в одну строку вывода. Предложение GROUP BY особенно полезно, когда результат содержит агрегатные функции. Выражения в приложении GROUP BY могут не входить в список выражений, содержащийся в инструкции SELECT.

Предложение HAVING похоже на WHERE тем, что оно ограничивает число строк, возвращаемых инструкцией. Однако предложение HAVING применяется после выполнения группировки, определенной предложением GROUP BY. По этой причине выражение HAVING может применяться к значениям, которые включают агрегатные функции. Выражения предложения HAVING могут не входить в список SELECT. Как и WHERE, выражение HAVING должно разрешаться в логическое значение.

Предложение ORDER BY обеспечивает сортировку строк вывода. Аргумент sort-expr-list, передаваемый в предложении ORDER BY, представляет собой список выражений, используемых в качестве ключа для сортировки. Выражения могут не входить в результат простой инструкции SELECT, но в составной инструкции SELECT (в которой используется один из операторов compound-op) каждое выражение сортировки должно точно соответствовать одному из столбцов результата. За каждым аргументом sort может следовать необязательное предложение порядка сортировки sort-order, которое содержит ключевое слово COLLATE и имя функции сортировки, используемое для упорядочения текста, и/или ключевое слово ASC или DESC, указывающее порядок сортировки (по возрастанию или по убыванию). Аргумент sort-order может быть опущен, в этом случае будет использоваться порядок сортировки по умолчанию (по возрастанию). Определение предложения COLLATE и функций сортировки см. в разделе «COLLATE».

Предложение LIMIT определяет верхний предел для числа строк, возвращаемых в результате. Отрицательное значение LIMIT указывает на то, что верхний предел не установлен. Необязательное предложение OFFSET после LIMIT указывает, сколько строк следует пропустить в начале набора результатов. В составном запросе SELECT предложение LIMIT может следовать только за последней инструкцией SELECT, при этом ограничение применяется ко всему запросу. Обратите внимание, что если в предложении LIMIT используется ключевое слово OFFSET, то ограничением является первое целое число, а смещением — второе. Если вместо ключевого слова OFFSET стоит запятая, то смещением является первое число, а ограничением — второе. Такое кажущееся противоречие является намеренным: оно позволяет обеспечить максимальную совместимость с устаревшими системами баз данных SQL.

Составная инструкция SELECT образуется из двух или нескольких простых инструкций SELECT, объединенных с помощью одного из операторов UNION, UNION ALL, INTERSECT или EXCEPT. Во всех инструкциях SELECT, которые входят в составную инструкцию SELECT, должно быть указано одинаковое число столбцов результата. Может использоваться только одно предложение ORDER BY, оно должно следовать за последней инструкцией SELECT (и перед единственным предложением LIMIT, если оно имеется). Операторы UNION и UNION ALL объединяют результаты предшествующей и последующей инструкций SELECT в одну таблицу. Разница между ними заключается в том, что при использовании UNION все строки результата являются уникальными, а при использовании UNION ALL могут встречаться повторения. Оператор INTERSECT возвращает результаты, совпавшие в предшествующей и последующей инструкциях SELECT. Оператор EXCEPT возвращает результат предшествующей инструкции SELECT после удаления из него результата последующей инструкции SELECT. Когда объединяется три и более инструкции SELECT, группировка выполняется начиная с первой и кончая последней.

Определения допустимых выражений см. в разделе «Выражения».

Начиная с версии AIR 2.5, оператор SQL CAST поддерживается при считывании данных BLOB для их преобразования в объекты ActionScript ByteArray. Например, следующий код считывает необработанные данные, которые не сохранены в формате AMF, и сохраняет их в объекте ByteArray:

```
stmt.text = "SELECT CAST(data AS ByteArray) AS data FROM pictures;";
stmt.execute();
var result:SQLResult = stmt.getResult();
var bytes:ByteArray = result.data[0].data;
```

### INSERT

Инструкция INSERT встречается в двух основных формах и служит для заполнения таблиц данными.

```
sql-statement  ::=  INSERT [OR conflict-algorithm] INTO [database-name.] table-name [(column-
list)] VALUES (value-list) |
                    INSERT [OR conflict-algorithm] INTO [database-name.] table-name [(column-
list)] select-statement
               REPLACE INTO [database-name.] table-name [(column-list)] VALUES (value-list) |
                    REPLACE INTO [database-name.] table-name [(column-list)] select-statement
```

Первая форма (с ключевым словом VALUES) создает новую строку в существующей таблице. Если аргумент column-list не определен, то число значений должно соответствовать числу столбцов в таблице. Если аргумент column-list определен, то число значений должно соответствовать числу указанных столбцов. Столбцы таблицы, которые не отображаются в списке столбцов, заполняются значением по умолчанию, определенным при создании таблицы, или значением NULL, если значение по умолчанию не определено.

Вторая форма инструкции INSERT получает свои данные из инструкции SELECT. Если аргумент column-list не определен, число столбцов, полученное в результате SELECT, должно точно соответствовать числу столбцов в таблице; в противном случае оно должно соответствовать числу столбцов, перечисленных в аргументе column-list. Для каждой строки результата SELECT создается новая запись в таблице. Инструкция SELECT может быть простой или составной. Определения разрешенных инструкций SELECT см. в разделе «SELECT».

Необязательный аргумент conflict-algorithm позволяет определить альтернативный алгоритм разрешения конфликтов между ограничениями, который должен использоваться при выполнении данной команды. Описание и определение алгоритмов конфликтов см. в разделе ««Специальные инструкции и предложения» на странице 1206».

Две формы REPLACE INTO инструкции равнозначны использованию стандартной формы INSERT [OR conflict-algorithm] с алгоритмом решения конфликтов REPLACE (то есть формы INSERT OR REPLACE...).

Две формы REPLACE INTO инструкции равнозначны использованию стандартной формы INSERT [OR conflict-algorithm] с алгоритмом решения конфликтов REPLACE (то есть формы INSERT OR REPLACE...).

### UPDATE

Команда update изменяет существующие записи в таблице.

```
sql-statement  ::=  UPDATE [database-name.] table-name SET column1=value1, column2=value2,...
[WHERE expr]
```

Команда состоит из ключевого слова UPDATE, а за ним следует имя таблицы, записи в которой требуется обновить. После ключевого слова SET укажите название столбца и значение, на которое будет изменено значение в столбце, в виде списка с разделением запятой. Выражение предложения WHERE указывает строку или строки, записи в которых будут обновлены.

### DELETE

Команда удаления служит для удаления записей из таблицы.

```
sql-statement  ::=  DELETE FROM [database-name.] table-name [WHERE expr]
```

Команда состоит из ключевых слов DELETE FROM, а за ними следует имя таблицы, из которой требуется удалить записи.

Если отсутствует предложение WHERE, удаляются все строки таблицы. Если имеется предложение WHERE, то удаляются только строки, соответствующие выражению. Выражение предложения WHERE должно разрешаться в логическое значение. Определения допустимых выражений см. в разделе «Выражения».

## Инструкции определения данных

Инструкции определения данных служат для создания, модификации и удаления объектов базы данных, включая таблицы, виды, индексы и триггеры. Поддерживаются следующие инструкции определения данных.

- Таблицы:
  - CREATE TABLE
  - ALTER TABLE
  - DROP TABLE
- Индексы:
  - CREATE INDEX
  - DROP INDEX
- Виды:
  - CREATE VIEWS
  - DROP VIEWS
- Триггеры:
  - CREATE TRIGGERS
  - DROP TRIGGERS

### CREATE TABLE

Инструкция CREATE TABLE состоит из ключевых слов CREATE TABLE, за которыми следует имя новой таблицы, а затем (в круглых скобках) список определений и ограничений столбцов. Имя таблицы может быть идентификатором или строкой.

```
sql-statement      ::=  CREATE [TEMP | TEMPORARY] TABLE [IF NOT EXISTS] [database-name.] table-
name
                        ( column-def [, column-def]* [, constraint]* )
sql-statement      ::=  CREATE [TEMP | TEMPORARY] TABLE [database-name.] table-name AS select-
statement
column-def          ::=  name [type] [[CONSTRAINT name] column-constraint]*
type                ::=  typename | typename ( number ) | typename ( number , number )
column-constraint   ::=  NOT NULL [ conflict-clause ] |
                         PRIMARY KEY [sort-order] [ conflict-clause ] [AUTOINCREMENT] |
                         UNIQUE [conflict-clause] |
                         CHECK ( expr ) |
                         DEFAULT default-value |
                         COLLATE collation-name
constraint          ::=  PRIMARY KEY ( column-list ) [conflict-clause] |
                         UNIQUE ( column-list ) [conflict-clause] |
                         CHECK ( expr )
conflict-clause     ::=  ON CONFLICT conflict-algorithm
conflict-algorithm  ::=  ROLLBACK | ABORT | FAIL | IGNORE | REPLACE
default-value     ::= NULL | string | number | CURRENT_TIME | CURRENT_DATE | CURRENT_TIMESTAMP
sort-order          ::=  ASC | DESC
collation-name      ::=  BINARY | NOCASE
column-list         ::=  column-name [, column-name]*
```

Каждое определение столбца содержит имя столбца, за которым следует тип данных и одно или несколько необязательных ограничений. Тип данных ограничивает данные, которые могут храниться в данном столбце. При попытке сохранить в столбце значение другого типа среда выполнения по возможности преобразует значение в соответствующий тип и выдает ошибку, если это не удается. Дополнительные сведения см. в разделе «Поддержка типов данных».

Ограничение столбца NOT NULL указывает, что столбец не может содержать значения NULL.

Ограничение UNIQUE создает индекс для указанных столбцов. Этот индекс должен содержать уникальные ключи: любые две строки не должны содержать повторяющихся значений или комбинаций значений в указанных столбцах. Инструкция CREATE TABLE может иметь несколько ограничений UNIQUE, включая несколько столбцов, в определении которых есть ограничение UNIQUE, и/или несколько ограничений UNIQUE на уровне таблицы.

Ограничение CHECK определяет выражение, которое должно вычисляться и быть верным (true), чтобы можно было вставить или обновить данные строки. Выражение CHECK должно разрешаться в логическое значение.

Предложение COLLATE в определении столбца указывает, какую функцию сортировки текста следует использовать при сравнении текстовых записей в столбце. По умолчанию используется функция сортировки BINARY. Сведения о предложении COLLATE и функциях сортировки см. в разделе «COLLATE».

Ограничение DEFAULT определяет значение по умолчанию, которое должно использоваться при выполнении инструкции INSERT. В качестве значения может использоваться NULL, строковая константа или число. В качестве значения по умолчанию может также использоваться одно специальных из ключевых слов (без учета регистра): CURRENT_TIME, CURRENT_DATE или CURRENT_TIMESTAMP. Если используется значение NULL, строковая константа или число, оно буквально вставляется в столбец каждый раз, когда в инструкции INSERT не определено значение для столбца. Если используется значение CURRENT_TIME, CURRENT_DATE или CURRENT_TIMESTAMP, то в столбец вставляется текущая дата и/или время UTC. Для CURRENT_TIME используется формат HH:MM:SS (ЧЧ:ММ:СС). Для CURRENT_DATE используется формат YYYY-MM-DD (ГГГГ-ММ-ДД). Для CURRENT_TIMESTAMP используется формат YYYY-MM-DD HH:MM:SS (ГГГГ-ММ-ДД ЧЧ:ММ:СС.)

При указании ограничения PRIMARY KEY обычно просто создается индекс UNIQUE для соответствующих столбцов. Однако если ограничение PRIMARY KEY применяется только к одному столбцу с типом данных INTEGER (или если указан любой синоним, такой как int), то этот столбец используется базой данных как действительный первичный ключ для таблицы. Это означает, что данный столбец может содержать только уникальные целочисленные значения. (Обратите внимание, что во многих случаях внедрения SQLite только тип данных столбца INTEGER определяет функцию столбца как внутреннего первичного ключа, но в Adobe AIR это поведение также определяется синонимами INTEGER, такими как int.)

Если в таблице нет столбца INTEGER PRIMARY KEY, целочисленный ключ автоматически создается при вставке каждой строки. Первичный ключ для строки можно всегда получить с помощью одного из специальных имен: ROWID, OID или _ROWID_. Эти имена можно использовать независимо от того, объявляются они явно как INTEGER PRIMARY KEY или генерируются как внутреннее значение. Однако если в таблице явно объявлен столбец INTEGER PRIMARY KEY, имя столбца в данных результата соответствует действительному имени, а не специальному.

Столбец INTEGER PRIMARY KEY также может содержать ключевое слово AUTOINCREMENT. При использовании ключевого слова AUTOINCREMENT база данных автоматически генерирует и вставляет последовательно увеличиваемый целочисленный ключ в столбце INTEGER PRIMARY KEY, когда выполняется инструкция INSERT, в которой не указано явного значения для столбца.

В инструкции CREATE TABLE может быть только одно ограничение PRIMARY KEY. Оно может входить в состав определения одного столбца или быть единственным ограничением PRIMARY KEY на уровне таблицы. К столбцу первичного ключа применяется неявное ограничение NOT NULL.

Необязательный аргумент conflict-clause, следующий за ограничением, позволяет указать для данного ограничения альтернативный алгоритм решения конфликтов между ограничениями по умолчанию. По умолчанию используется значение ABORT. Для всех ограничений в одной таблице можно определить разные алгоритмы решения конфликтов по умолчанию. Если в инструкции INSERT или UPDATE указан другой алгоритм решения конфликтов, то он используется вместо алгоритма, определенного в инструкции CREATE TABLE. Дополнительные сведения см. в подразделе ON CONFLICT раздела ««Специальные инструкции и предложения» на странице 1206».

Дополнительные ограничения, такие как FOREIGN KEY, не приводят к появлению ошибки, но среда выполнения их игнорирует.

Если между ключевыми словами CREATE и TABLE встречается слово TEMP или TEMPORARY, то создаваемая таблица будет отображаться только в рамках текущего подключения к базе данных (экземпляра SQLConnection). Она автоматически удаляется после закрытия подключения к базе данных. Все индексы, созданные для временной таблицы, также являются временными. Временные таблицы и индексы хранятся в отдельном файле, а не в главном файле базы данных.

Если указан необязательный префикс имени базы данных database-name, то таблица создается в именованной базе данных (которая была подключена к экземпляру SQLConnection с помощью метода attach() с указанием имени базы данных). Нельзя одновременно использовать префикс database-name и ключевое слово TEMP, если префикс database-name не является temp. Если имя базы данных не указано и не используется ключевое слово TEMP, то таблица создается в главной базе данных (которая была подключена к экземпляру SQLConnection с помощью метода open() или openAsync()).

Не существует произвольных пределов для числа столбцов или ограничений в таблице. Также не установлено произвольных пределов для объема данных в строке.

Форма CREATE TABLE AS определяет таблицу как набор результатов запроса. Имена столбцов таблицы соответствуют именам столбцов в результате.

Если присутствует необязательное предложение IF NOT EXISTS и существует другая таблица с тем же именем, то база данных игнорирует команду CREATE TABLE.

Таблицу можно удалить с помощью инструкции DROP TABLE, а с помощью инструкции ALTER TABLE можно внести ряд изменений.

### ALTER TABLE

Команда ALTER TABLE дает пользователю возможность переименовать или добавить столбец в существующей таблице. Удалить столбец из таблицы невозможно.

```
sql-statement ::= ALTER TABLE [database-name.] table-name alteration
alteration    ::= RENAME TO new-table-name
alteration    ::= ADD [COLUMN] column-def
```

Синтаксис RENAME TO можно использовать, чтобы присвоить таблице с именем, определенном параметром [database-name.] в аргументе table-name, новое имя, переданное в аргументе new-table-name. Эту команду нельзя использовать для перемещения таблицы из одной подключенной базы данных в другую. Она предназначена только для переименования таблицы в рамках одной базы данных.

Если в переименовываемой таблице есть триггеры или индексы, они останутся подключенными к таблице после ее переименования. Однако, если существуют определения или инструкции видов, которые выполняются триггерами, связанными с переименовываемой таблицей, они не изменяются автоматически для использования нового имени таблицы. Если переименованная таблица имеет связанные виды или триггеры, необходимо вручную удалить и воссоздать эти определения видов или триггеры, используя новое имя таблицы.

Синтаксис ADD [COLUMN] служит для добавления нового столбца в существующую таблицу. Новый столбец всегда добавляется в конце списка существующих столбцов. Предложение column-def может принимать любые формы, допустимые в инструкции CREATE TABLE, с учетом следующих ограничений.

- Столбец не должен иметь ограничения PRIMARY KEY или UNIQUE.
- Столбец не должен иметь значения по умолчанию CURRENT_TIME, CURRENT_DATE или CURRENT_TIMESTAMP.
- Если указано ограничение NOT NULL, столбец не может иметь значения по умолчанию NULL.

На время выполнения инструкции ALTER TABLE не влияет объем данных в таблице.

### DROP TABLE

Инструкция DROP TABLE удаляет таблицу, добавленную с помощью инструкции CREATE TABLE. Удаляется таблица, имя которой указано в аргументе table-name. Она полностью удаляется из базы данных и файла на диске. Таблицу невозможно восстановить. Все индексы, связанные с этой таблицей, также удаляются.

```
sql-statement  ::=  DROP TABLE [IF EXISTS] [database-name.] table-name
```

По умолчанию инструкция DROP TABLE не уменьшает размер файла базы данных. Пустое место в базе данных сохраняется и используется при последующих операциях INSERT. Чтобы удалить свободное место из базы данных, используйте метод SQLConnection.clean(). Если при создании базы данных параметру autoClean было задано значение true, место освобождается автоматически.

Необязательное предложение IF EXISTS подавляет появление ошибки, которая обычно выдается, если указанная таблица не существует.

### CREATE INDEX

Команда CREATE INDEX содержит ключевые слова CREATE INDEX, за которыми следуют имя нового индекса, ключевое слово ON, имя созданной ранее таблицы, которую требуется проиндексировать, и заключенный в круглые скобки список имен столбцов в таблице, значения которых используются для ключа индекса.

```
sql-statement  ::=  CREATE [UNIQUE] INDEX [IF NOT EXISTS] [database-name.] index-name
                    ON table-name ( column-name [, column-name]* )
column-name    ::=  name [COLLATE collation-name] [ASC | DESC]
```

После каждого имени столбца может следовать ключевое слово ASC или DESC, которое указывает порядок сортировки, но во время выполнения этот порядок игнорируется. Сортировка всегда осуществляется в порядке возрастания.

Предложение COLLATE после каждого имени столбца определяет последовательность сортировки, которая используется для текстовых значений в этом столбце. По умолчанию используется последовательность сортировки, определенная для данного столбца в инструкции CREATE TABLE. Если последовательность сортировки не указана, используется значение BINARY. Определение предложения COLLATE и функций сортировки см. в разделе «COLLATE».

Не существует произвольных ограничений для числа индексов, которые можно подключить к одной таблице. Также не установлено ограничений для числа столбцов в индексе.

### DROP INDEX

Инструкция DROP INDEX удаляет индекс, добавленный с помощью инструкции CREATE INDEX. Указанный индекс полностью удаляется из файла базы данных. Восстановить индекс можно только путем повторного ввода соответствующей команды CREATE INDEX.

```
sql-statement ::= DROP INDEX [IF EXISTS] [database-name.] index-name
```

По умолчанию инструкция DROP INDEX не уменьшает размер файла базы данных. Пустое место в базе данных сохраняется и используется при последующих операциях INSERT. Чтобы удалить свободное место из базы данных, используйте метод SQLConnection.clean(). Если при создании базы данных параметру autoClean было задано значение true, место освобождается автоматически.

### CREATE VIEW

Команда CREATE VIEW присваивает имя предварительно определенной инструкции SELECT. Это новое имя можно использовать в предложении FROM другой инструкции SELECT вместо имени таблицы. Виды часто применяются для упрощения запросов путем объединения сложных (и часто вызываемых) наборов данных в структуру, которую можно использовать в других операциях.

```
sql-statement ::= CREATE [TEMP | TEMPORARY] VIEW [IF NOT EXISTS] [database-name.] view-name AS
select-statement
```

Если между ключевыми словами CREATE и VIEW встречается слово TEMP или TEMPORARY, то создаваемый вид будет отображаться только для текущего экземпляра SQLConnection, с помощью которого была открыта база данных. Он автоматически удаляется после закрытия базы данных.

Если указан префикс [database-name], вид создается в названной базе данных (которая была подключена к экземпляру SQLConnection с помощью метода attach() с указанием имени базы данных). Нельзя одновременно использовать префикс [database-name] и ключевое слово TEMP, если префикс [database-name] не является временным. Если имя базы данных не указано и не используется ключевое слово TEMP, то вид создается в главной базе данных (которая была подключена к экземпляру SQLConnection с помощью метода open() или openAsync()).

Виды доступны только для чтения. Для вида нельзя использовать инструкцию DELETE, INSERT или UPDATE, если определен хотя бы один триггер связанного типа (INSTEAD OF DELETE, INSTEAD OF INSERT, INSTEAD OF UPDATE). Сведения о создании триггера для вида см. в разделе «CREATE TRIGGER».

Вид удаляется из базы данных с помощью инструкции DROP VIEW.

### DROP VIEW

Инструкция DROP VIEW удаляет вид, созданный с помощью инструкции CREATE VIEW.

```
sql-statement ::= DROP VIEW [IF EXISTS] view-name
```

Удаляется вид, имя которого указано в аргументе view-name. Вид удаляется из базы данных, но данные в самих таблицах остаются без изменений.

### CREATE TRIGGER

Инструкция CREATE TRIGGER используется для добавления триггеров в схему базы данных. Триггер представляет собой операцию базы данных (trigger-action), которая автоматически выполняется в случае определенного события базы данных (database-event).

```
sql-statement   ::=  CREATE [TEMP | TEMPORARY] TRIGGER [IF NOT EXISTS] [database-name.] trigger-
name
                      [BEFORE | AFTER] database-event
                      ON table-name
                      trigger-action
sql-statement   ::=  CREATE [TEMP | TEMPORARY] TRIGGER [IF NOT EXISTS] [database-name.] trigger-
name
                      INSTEAD OF database-event
                      ON view-name
                      trigger-action
database-event  ::=  DELETE |
                      INSERT |
                      UPDATE |
                      UPDATE OF column-list
trigger-action  ::=  [FOR EACH ROW] [WHEN expr]
                      BEGIN
                        trigger-step ;
                        [ trigger-step ; ]*
                      END
trigger-step    ::=  update-statement |
                      insert-statement |
                      delete-statement |
                      select-statement
column-list     ::=  column-name [, column-name]*
```

Триггер можно определить так, чтобы он запускался каждый раз, когда выполняется инструкция DELETE, INSERT или UPDATE для определенной таблицы базы данных, или каждый раз, когда обновляется инструкция UPDATE одного или нескольких указанных столбцов таблицы. Триггеры являются постоянными, если при их создании не используется ключевое слово TEMP или TEMPORARY. В таком случае триггер удаляется, когда закрывается подключение к главной базе данных, открытое экземпляром SQLConnection. Если не заданы временные рамки (BEFORE или AFTER), по умолчанию используется триггер BEFORE.

Поддерживаются только триггеры FOR EACH ROW, поэтому синтаксис FOR EACH ROW не является обязательным. При использовании триггера FOR EACH ROW инструкции trigger-step выполняются для каждой строки базы данных, вставляемой, обновляемой или удаляемой с помощью инструкции, запускающей триггер, если выражение предложения WHEN является верным.

Если используется предложение WHEN, инструкции SQL, указанные как trigger-step, выполняются только для строк, для которых верно предложение WHEN. Если предложение WHEN не определено, инструкции SQL выполняются для всех строк.

В теле триггера (предложение trigger-action) значения до и после изменения (pre-change и post-change) затрагиваемой таблицы можно получить с помощью специальных имен таблиц OLD и NEW. Структура таблиц OLD и NEW соответствует структуре таблицы, в которой создан триггер. Таблица OLD содержит строки, измененные или удаленные инструкцией, запустившей триггер, в том состоянии, в котором они находились до выполнения операций инструкции. Таблица NEW содержит строки, измененные или созданные инструкцией, запустившей триггер, в том состоянии, в котором они находятся после выполнения операций инструкции. И предложение WHEN и инструкции trigger-step могут получать значения из вставленной, удаленной или обновленной строки, используя ссылки формы NEW.column-name и OLD.column-name, где column-name — это имя столбца из таблицы, с которой связан триггер. Доступность ссылок таблиц OLD и NEW зависит от типа события database-event, обрабатываемого триггером.

- INSERT: ссылки NEW являются действительными.
- UPDATE: ссылки NEW и OLD являются действительными.

- DELETE: ссылки OLD являются действительными.

Указанное время (BEFORE, AFTER или INSTEAD OF) определяет, когда выполняются инструкции trigger-step относительно вставки, изменения или удаления связанной строки. Предложение ON CONFLICT может быть включено в состав инструкции UPDATE или INSERT в trigger-step. Однако если предложение ON CONFLICT указано в составе инструкции, которая запускает триггер, то используется именно эта политика решения конфликтов.

Помимо триггеров таблицы, можно создать триггер INSTEAD OF для вида. Если для вида определен один или несколько триггеров INSTEAD OF INSERT, INSTEAD OF DELETE или INSTEAD OF UPDATE, то выполнение соответствующего типа инструкции (INSERT, DELETE, или UPDATE) для вида не считается ошибкой. В таком случае выполнение инструкции INSERT, DELETE или UPDATE для вида запускает соответствующие триггеры. Так как триггер относится к типу INSTEAD OF, таблицы, используемые в качестве основы вида, не изменяются инструкцией, которая запускает триггер. Однако триггеры можно использовать для выполнения операций модификации базовых таблиц.

При создании триггера для таблицы со столбцом INTEGER PRIMARY KEY важно помнить следующее. Если триггер типа BEFORE изменяет столбец INTEGER PRIMARY KEY строки, которая должна обновляться инструкцией, запускающий триггер, обновление не происходит. Чтобы обойти эту проблему, необходимо создать таблицу со столбцом PRIMARY KEY вместо INTEGER PRIMARY KEY.

Триггер можно удалить с помощью инструкции DROP TRIGGER. При удалении таблицы или вида автоматически удаляются и все связанные триггеры.

### Функция RAISE ()

Специальную функцию SQL RAISE() можно использовать в инструкции триггера trigger-step. Эта функция имеет следующий синтаксис:

```
raise-function  ::=  RAISE ( ABORT, error-message ) |
                     RAISE ( FAIL, error-message ) |
                     RAISE ( ROLLBACK, error-message ) |
                     RAISE ( IGNORE )
```

Когда одна из первых трех форм вызывается в процессе выполнения триггера, выполняется указанное действие обработки ON CONFLICT (ABORT, FAIL или ROLLBACK), а выполнение текущей инструкции прерывается. Действие ROLLBACK рассматривается как сбой выполнения инструкции, поэтому экземпляр SQLStatement, для которого выполнялся метод execute(), отправляет событие ошибки (SQLErrorEvent.ERROR). Объект SQLError в свойстве error отправленного объекта события имеет свойство details, которое содержит сообщение ошибки error-message, переданное в функции RAISE().

Когда вызывается функция RAISE(IGNORE), прекращается выполнение оставшейся части текущего триггера, инструкции, запустившей его выполнение, а также любых последующих триггеров. Откат изменений базы данных не выполняется. Если инструкция, запустившая выполнение триггера, является частью другого триггера, то выполнение этого триггера возобновляется с начала следующего шага. Дополнительные сведения об алгоритмах решения конфликтов см. в разделе «ON CONFLICT (алгоритмы конфликтов)».

### DROP TRIGGER

Инструкция DROP TRIGGER удаляет триггер, созданный с помощью инструкции CREATE TRIGGER.

```
sql-statement  ::=  DROP TRIGGER [IF EXISTS] [database-name.] trigger-name
```

Триггер удаляется из базы данных. Обратите внимание, что триггеры автоматически удаляются при удалении связанной с ними таблицы.

## Специальные инструкции и предложения

В этом разделе описано несколько предложений, которые являются расширениями SQL, предоставляемыми средой выполнения, а также два элемента языка, которые можно использовать во многих инструкциях, комментариях и выражениях.

### COLLATE

Предложение COLLATE используется в инструкциях SELECT, CREATE TABLE и CREATE INDEX для определения алгоритма сравнения, который должен использоваться при сравнении или сортировке значений.

```
sql-statement   ::=  COLLATE collation-name
collation-name  ::=  BINARY | NOCASE
```

По умолчанию для столбцов используется тип сортировки BINARY. Когда сортировка BINARY используется со значениями класса хранения TEXT, двоичная сортировка выполняется путем сравнения байтов в памяти, которые представляют значение независимо от кодировки текста.

Последовательность сортировки NOCASE применяется только к значениям класса хранения TEXT. При использовании сортировки NOCASE выполняется сравнение без учета регистра.

Последовательность сортировки не используется для классов хранения типа NULL, BLOB, INTEGER или REAL.

Чтобы использовать для столбца другой тип сортировки вместо BINARY, предложение COLLATE необходимо включить в определение столбца в инструкции CREATE TABLE. При любом сравнении двух значений TEXT используется последовательность сортировки для определения результатов сравнения в соответствии со следующими правилами.

- Если при использовании операторов двоичного сравнения любой из операндов является столбцом, то тип сортировки, по умолчанию используемый для данного столбца, определяет последовательность сортировки, применяемую для выполнения сравнения. Если оба операнда являются столбцами, то используемая последовательность сортировки определяется по типу сортировки левого операнда. Если ни один из операндов не является столбцом, используется последовательность сортировки BINARY.
- Оператор BETWEEN...AND равносилен использованию двух выражений с операторами >= и <=. Например, выражение x BETWEEN y AND z эквивалентно выражению x >= y AND x <= z. Таким образом, оператор BETWEEN...AND следует описанному правилу для определения последовательности сортировки.
- Оператор IN по своему поведению напоминает оператор =, используемый для определения последовательности сортировки. Например, последовательность сортировки, используемая для выражения x IN (y, z), соответствует типу сортировки по умолчанию для x, если x является столбцом. В противном случае используется сортировка BINARY.
- Предложению ORDER BY, которое входит в состав инструкции SELECT, можно явно назначить последовательность сортировки, которая должна использоваться для данной операции. В таком случае всегда используется явно назначенная последовательность сортировки. И наоборот, если выражение, сортируемое предложением ORDER BY, является столбцом, порядок сортировки определяется по типу сортировки, используемому по умолчанию для столбца. Если выражение не является столбцом, используется последовательность сортировки BINARY.

### EXPLAIN

Модификатор команды EXPLAIN является нестандартным расширением SQL.

```
sql-statement  ::=  EXPLAIN sql-statement
```

Если ключевое слово EXPLAIN стоит перед любой другой инструкцией SQL, то вместо фактического выполнения команды результат сообщает последовательность инструкций виртуальной машины, которые бы использовались для выполнения команды, если бы не было ключевого слова EXPLAIN. EXPLAIN представляет собой расширенную функцию, с помощью которой разработчики могут изменять текст инструкции SQL, чтобы оптимизировать производительность или отладить некорректно работающую инструкцию.

### ON CONFLICT (алгоритмы конфликтов)

Предложение ON CONFLICT не является отдельной командой SQL. Это нестандартное предложение, которое может встречаться во многих других командах SQL.

```
conflict-clause     ::=  ON CONFLICT conflict-algorithm
conflict-clause     ::=  OR conflict-algorithm
conflict-algorithm  ::=  ROLLBACK |
                         ABORT |
                         FAIL |
                         IGNORE |
                         REPLACE
```

Первая форма предложения ON CONFLICT, в которой используются ключевые слова ON CONFLICT, применяется в инструкции CREATE TABLE. Для инструкции INSERT или UPDATE используется вторая форма, где ключевые слова ON CONFLICT меняются на OR, чтобы придать синтаксису более естественный вид. Например, вместо INSERT ON CONFLICT IGNORE инструкция принимает вид INSERT OR IGNORE. Хотя используются разные ключевые слова, обе формы имеют один и тот же смысл предложения.

Предложение ON CONFLICT определяет алгоритм, которые используется для разрешения конфликтов ограничений. Существует пять алгоритмов: ROLLBACK, ABORT, FAIL, IGNORE и REPLACE. По умолчанию используется алгоритм ABORT. Ниже приводится объяснение пяти алгоритмов конфликтов.

**ROLLBACK** В случае нарушения ограничения сразу выполняется откат (ROLLBACK), что завершает текущую транзакцию. Выполнение команды прерывается, и экземпляр SQLStatement отправляет событие ошибки. Если нет активных транзакций (кроме подразумеваемой транзакции, которая создается для каждой команды), этот алгоритм ведет себя так же, как ABORT.

**ABORT** В случае нарушения ограничения, команда отменяет все свои внесенные ранее изменения, и экземпляр SQLStatement отправляет событие ошибки. Откат (ROLLBACK) не выполняется, поэтому изменения, выполненные предшествующими командами в рамках транзакции, сохраняются. Алгоритм ABORT используется как поведение по умолчанию.

**FAIL** В случае нарушения ограничения выполнение команды прекращается и экземпляр SQLStatement отправляет событие ошибки. Однако все изменения, которые внесла инструкция в базу данных до нарушения, сохраняются и не отменяются. Например, если при выполнении инструкции UPDATE встречается нарушение ограничения в строке 100, которую требуется обновить, то изменения в первых 99 строках сохраняются, а строка 100 и следующие за ней остаются без изменений.

**IGNORE** В случае нарушения ограничения та строка, которая содержит это нарушение, не вставляется или не изменяется. Данная строка игнорируется, и выполнение команды продолжается. Другие строки до и после той, которая содержит нарушение ограничения, вставляются или обновляются в обычном порядке. Ошибка при этом не возникает.

**REPLACE** Когда возникает нарушение ограничения UNIQUE, перед вставкой или обновлением текущей строки удаляются существующие строки, вызвавшие это нарушение. Следовательно, вставка или обновление происходит всегда, и выполнение команды продолжается. Ошибка при этом не возникает. Если происходит нарушение ограничения NOT NULL, значение NULL заменяется значением по умолчанию, установленным для данного столбца. Если для столбца не задано значение по умолчанию, используется алгоритм ABORT.

Если происходит нарушение ограничения CHECK, используется алгоритм IGNORE. Когда эта стратегия решения конфликтов удаляет строку, чтобы реализовать ограничение, триггер удаления в этой строке не запускается.

Алгоритм, определенный в предложении OR инструкции INSERT или UPDATE, переопределяет любой алгоритм, указанный в инструкции CREATE TABLE. Если алгоритм не указан в инструкции CREATE TABLE или в выполняемой инструкции INSERT или UPDATE, используется алгоритм ABORT.

### REINDEX

Команда REINDEX служит для удаления и воссоздания одного или нескольких индексов. Эта команда полезна, когда происходит изменение определения последовательности сортировки.

```
sql-statement  ::=  REINDEX collation-name
sql-statement  ::=  REINDEX [database-name .] ( table-name | index-name )
```

Первая форма воссоздает все индексы во всех подключенных базах данных, которые используют названную последовательность сортировки. Вторая форма перестраивает все индексы, связанные с таблицей, имя которой указано в аргументе table-name. Если в аргументе index-name задано имя индекса, удаляется и воссоздается только указанный индекс.

### КОММЕНТАРИИ

Комментарии не являются командами SQL, но они могут встречаться в запросах SQL. Среда выполнения рассматривает их как пробел. Комментарии могут начинаться в любом месте, где может быть пробел, в том числе внутри многострочных выражений.

```
comment             ::=  single-line-comment |
                         block-comment
single-line-comment ::=  -- single-line
block-comment       ::=  /* multiple-lines or block [*/]
```

Однострочный комментарий обозначается двумя дефисами. Однострочный комментарий может продолжаться только до конца текущей строки.

Блоки комментариев могут быть многострочными или встраиваться в одну строку. Если отсутствует конечный ограничитель, блок комментариев продолжается до конца ввода. Такая ситуация не рассматривается как ошибка. Новая инструкция SQL может начаться в строке после окончания блока комментариев. Блоки комментариев могут встраиваться в любом месте, где может находиться пробел, в том числе внутри выражений, и в середине других выражений SQL. Блоки комментариев не могут быть вложенными. Однострочные комментарии внутри блока игнорируются.

### ВЫРАЖЕНИЯ

Выражения — это подкоманды внутри других блоков SQL. Ниже описывается допустимый синтаксис для выражений внутри инструкции SQL.

```
expr            ::=  expr binary-op expr |
                     expr [NOT] like-op expr [ESCAPE expr] |
                     unary-op expr |
                     ( expr ) |
                     column-name |
                     table-name.column-name |
                     database-name.table-name.column-name |
                     literal-value |
                     parameter |
                     function-name( expr-list | * ) |
                     expr ISNULL |
                     expr NOTNULL |
                     expr [NOT] BETWEEN expr AND expr |
                     expr [NOT] IN ( value-list ) |
                     expr [NOT] IN ( select-statement ) |
                     expr [NOT] IN [database-name.] table-name |
                     [EXISTS] ( select-statement ) |
                     CASE [expr] ( WHEN expr THEN expr )+ [ELSE expr] END |
                     CAST ( expr AS type ) |
                     expr COLLATE collation-name
like-op         ::=  LIKE | GLOB
binary-op       ::=  see Operators
unary-op        ::=  see Operators
parameter       ::=  :param-name | @param-name | ?
value-list      ::=  literal-value [, literal-value]*
literal-value   ::=  literal-string | literal-number | literal-boolean | literal-blob |
literal-null
literal-string  ::=  'string value'
literal-number  ::=  integer | number
literal-boolean  ::=  true | false
literal-blob  ::=  X'string of hexadecimal data'
literal-null  ::=  NULL
```

Выражением является любая комбинация значений и операторов, которые могут разрешаться в одно значение. Выражения можно разделить на два общих типа в соответствии с тем, разрешаются они в логическое значение (true или false) или в нелогическое значение.

В нескольких распространенных ситуациях (включая предложение WHERE, предложение HAVING, выражение ON в предложении JOIN и выражение CHECK) выражение должно разрешаться в логическое значение. Этому условию отвечают следующие типы выражений:

- ISNULL
- NOTNULL
- IN ()
- EXISTS ()
- LIKE
- GLOB
- Определенные функции
- Определенные операторы (в частности, операторы сравнения)

**Значения литерала**

Числовое значение литерала записывается как целое число или число с плавающей запятой. Поддерживается экспоненциальное представление. В качестве разделителя десятичной дроби всегда используется символ. (точка).

Строковый литерал обозначается заключением строки в одинарные кавычки '. Чтобы включить в строку одинарную кавычку, поставьте две одинарные кавычки подряд, например: ''.

Логический литерал определяется по значению true (истина) или false (ложь). Логические значения литерала используются вместе с логическим типом данных столбца.

Литерал большого двоичного объекта (BLOB) представляет собой строковый литерал, содержащий шестнадцатеричные данные и один символ x или X в начале, например X'53514697465'.

В качестве значения литерала также может использоваться маркер NULL.

**Имя столбца**

В качестве имени столбца могут использоваться любые имена, определенные в инструкции CREATE TABLE, или один из следующих специальных идентификаторов: ROWID, OID или _ROWID_. Все эти специальные идентификаторы описывают уникальный произвольный целочисленный ключ («ключ строки»), связанный с каждой строкой в каждой таблице. Специальные идентификаторы ссылаются на ключ строки, только если в инструкции CREATE TABLE не определен реальный столбец с тем же именем. Ключи строк ведут себя как столбцы, доступные только для чтения. Ключ строки можно использовать везде, где используются обычные столбцы, с тем исключением, что значение ключа строки нельзя изменить в инструкции UPDATE или INSERT. Инструкция таблицы SELECT * FROM не включает ключ строки в набор результатов.

**Инструкция SELECT**

Инструкция SELECT может встречаться в выражении как правый операнд оператора IN, как скалярное количество (одно значение результата) или как операнд оператора EXISTS. При использовании в качестве скалярного количества или операнда оператора IN инструкция SELECT может иметь в результате только один столбец. Допускается использование составной инструкции SELECT (соединенной с помощью таких ключевых слов, как UNION или EXCEPT). При использовании оператора EXISTS столбцы в наборе результатов инструкции SELECT игнорируются и выражение возвращает значение TRUE, если существует одна или несколько строк, или FALSE, если набор результатов пустой. Если в выражении SELECT ни один термин не ссылается на значение в запросе-контейнере, то выражение вычисляется один раз перед любой другой обработкой, при необходимости результат можно использовать повторно. Если выражение SELECT содержит переменные из внешнего запроса, который называется связанным подзапросом, оно повторно вычисляется при каждом использовании.

Когда SELECT является правым операндом оператора IN, оператор IN возвращает значение TRUE, если результат левого операнда равен любому значению в наборе результатов инструкции SELECT. Перед оператором IN может стоять ключевое слово NOT, которое меняет смысл проверки на обратный.

Когда инструкция SELECT встречается в выражении, но не в правой части оператора IN, в выражении используется первая строка результата SELECT. Если инструкция SELECT возвращает несколько строк результатов, все строки, следующие за первой, игнорируются. Если инструкция SELECT не возвращает строк, SELECT имеет значение NULL.

**Выражение CAST**

Выражение CAST меняет тип данных указанного значения на заданный. В качестве типа может быть указано любое непустое имя, которое является действительным для типа в определении столбца в инструкции CREATE TABLE. Дополнительные сведения см. в разделе «Поддержка типов данных».

### Дополнительные элементы выражений

Следующие элементы SQL также можно использовать в выражениях.

- Встроенные функции: агрегатные функции, скалярные функции, функции форматирования даты и времени
- Операторы
- Параметры

## Встроенные функции

Встроенные функции делятся на три основных категории:

- агрегатные функции;
- скалярные функции;
- функции даты и времени.

Помимо этих функций, существует специальная функция RAISE(), которая служит для уведомления об ошибках, возникающих при выполнении триггера. Эту функцию можно использовать только в теле инструкции CREATE TRIGGER. Сведения о функции RAISE() см. в разделе «CREATE TRIGGER > RAISE()».

Как и во всех ключевых словах SQL, в именах функций не учитывается регистр.

### Агрегатные функции

Агрегатные функции выполняют операции со значениями из нескольких строк. Эти функции главным образом используются в инструкциях SELECT вместе с предложением GROUP BY.

| | |
|---|---|
| AVG(X) | Возвращает среднее значение всех ненулевых строк X в группе. Строковые значения и значения больших двоичных объектов (BLOB), которые не похожи на числа, интерпретируются как 0. Результатом функции AVG() всегда является значение с плавающей запятой, даже если все вводные числа являются целыми. |
| COUNT(X)<br>COUNT(*) | Первая форма возвращает количество ненулевых строк X в группе. Вторая форма (с аргументом *) возвращает общее число строк в группе. |
| MAX(X) | Возвращает максимальное из всех значений в группе. Для определения максимального значения используется обычный порядок сортировки. |
| MIN(X) | Возвращает минимальное ненулевое значение из всех значений в группе. Для определения минимального значения используется обычный порядок сортировки. Если все значения в группе нулевые, возвращается значение NULL. |
| SUM(X)<br>TOTAL(X) | Возвращает числовую сумму всех ненулевых значений в группе. Если все значения нулевые, функция SUM() возвращает NULL, а функция TOTAL() возвращает 0.0. Результат функции TOTAL() всегда является значением с плавающей запятой. Результат функции SUM() является целым числом, если все ненулевые значения являются целыми числами. Если любое число, переданное функции SUM(), не является целым и не равно нулю, то SUM() возвращает значение с плавающей запятой. Действительная сумма может быть округлена до приближенного значения. |

В любой предшествующей агрегатной функции, которая принимает один аргумент, перед этим аргументом может стоять ключевое слово DISTINCT. В таком случае повторяющиеся элементы отфильтровываются перед передачей результата следующей агрегатной функции. Например, вызов функции COUNT(DISTINCT x) возвращает число разных значений в столбце X, а не общее число ненулевых значений.

### Скалярные функции

Скалярные функции выполняют операции со значениями только одной строки.

| | |
|---|---|
| ABS(X) | Возвращает абсолютное значение аргумента X. |
| COALESCE(X, Y, ...) | Возвращает копию первого ненулевого аргумента. Если все аргументы нулевые, возвращается результат NULL. Должно быть не меньше двух аргументов. |

| | |
|---|---|
| GLOB(X, Y) | Эта функция служит для реализации синтаксиса X GLOB Y. |
| IFNULL(X, Y) | Возвращает копию первого ненулевого аргумента. Если оба аргумента нулевые, возвращается результат NULL. Эта функция действует аналогично COALESCE(). |
| HEX(X) | Аргумент интерпретируется как значение типа хранения BLOB. В качестве результата возвращается шестнадцатеричное представление содержимого этого значения. |
| LAST_INSERT_ROWID( ) | Возвращает идентификатор (сгенерированный первичный ключ) последней строки, вставленной в базу данных с помощью текущего экземпляра SQLConnection. Это значение идентично значению, возвращаемому свойством SQLConnection.lastInsertRowID. |
| LENGTH(X) | Возвращает длину строки X (число символов). |
| LIKE(X, Y [, Z]) | Эта функция служит для реализации синтаксиса X LIKE Y [ESCAPE Z] в SQL. Если используется необязательное предложение ESCAPE, то функция вызывается с тремя аргументами. В противном случае она вызывается только с двумя аргументами. |
| LOWER(X) | Возвращает копию строки X, все символы которой преобразованы в нижний регистр. |
| LTRIM(X) LTRIM(X, Y) | Возвращает строку, полученную в результате удаления пробела с левой стороны X. Если указан аргумент Y, функция удаляет все символы в Y, которые находятся с левой стороны от X. |
| MAX(X, Y, ...) | Возвращает аргумент с максимальным значением. В качестве аргументов могут использоваться как числа, так и строки. Максимальное значение определяется с использованием порядка сортировки по умолчанию. Обратите внимание, что MAX() является простой функцией, когда имеет 2 или более аргументов, но если используется только один аргумент, она работает как агрегатная функция. |
| MIN(X, Y, ...) | Возвращает аргумент с минимальным значением. В качестве аргументов могут использоваться как числа, так и строки. Минимальное значение определяется с использованием порядка сортировки по умолчанию. Обратите внимание, что MIN() является простой функцией, когда имеет 2 или более аргументов, но если используется только один аргумент, она работает как агрегатная функция. |
| NULLIF(X, Y) | Возвращает первый аргумент, если аргументы разные. В противном случае возвращает NULL. |
| QUOTE(X) | Эта подпрограмма возвращает строку, которая является значением ее аргумента, пригодного для включения в другую инструкцию SQL. Строки окружены одинарными кавычками. При необходимости для внутренних кавычек применяется маскировка. Классы хранения BLOB кодируются как шестнадцатеричные литералы. Эта функция полезна для обеспечения возможности отмены или повтора действия при создании триггеров. |
| RANDOM(*) | Возвращает псевдопроизвольное целое число из диапазона от 9223372036854775808 до 9223372036854775807. Это произвольное значение не обеспечивает надежное шифрование. |
| RANDOMBLOB(N) | Возвращает большой двоичный объект размером N байт, который содержит псевдопроизвольные байты. В качестве N должно использоваться положительное целое число. Это произвольное значение не обеспечивает надежное шифрование. Если в N имеет отрицательное значение, возвращается один байт. |
| ROUND(X) ROUND(X, Y) | Округляет число X до Y знаков справа от разделителя десятичной дроби. Если аргумент Y опущен, используется значение 0. |
| RTRIM(X) RTRIM(X, Y) | Возвращает строку, полученную в результате удаления пробела с правой стороны X. Если указан аргумент Y, функция удаляет все символы в Y, которые находятся с правой стороны от X. |
| SUBSTR(X, Y, Z) | Возвращает подстроку вводной строки X, которая начинается с Y-го символа и имеет длину Z символов. Крайний левый символ строки X имеет индекс 1. Если Y является отрицательным числом, первый символ подстроки определяется путем отсчета от правого края, а не левого. |
| TRIM(X) TRIM(X, Y) | Возвращает строку, полученную в результате удаления пробела с правой стороны X. Если указан аргумент Y, функция удаляет все символы в Y, которые находятся с правой стороны от X. |
| TYPEOF(X) | Возвращает тип выражения X. Возможные возвращаемые значения: null, integer, real, text и blob. Дополнительные сведения о типах данных см. в разделе «Поддержка типов данных». |
| UPPER(X) | Возвращает копию вводной строки X, преобразованную в верхний регистр. |
| ZEROBLOB(N) | Возвращает большой двоичный объект, содержащий N байт со значением 0x00. |

### Функции форматирования даты и времени

Функции форматирования даты и времени представляют собой группу скалярных функций, которые служат для создания форматированных данных даты и времени. Обратите внимание, что эти функции выполняют операции со значениями, содержащими строки и числа, и возвращают соответствующие результаты. Эти функции не предназначены для использования с типом данных DATE. Если их использовать для данных в столбце, для которого объявлен тип данных DATE, они работают некорректно.

| | |
|---|---|
| DATE(T, ...) | Функция DATE() возвращает строку, содержащую дату в формате YYYY-MM-DD (ГГГГ-ММ-ДД). Первый параметр (T) определяет строку времени в формате, описанном в разделе «Форматы времени». После строки времени можно указать любое число модификаторов. Модификаторы перечислены в разделе «Модификаторы». |
| TIME(T, ...) | Функция TIME() возвращает строку, содержащую время в формате HH:MM:SS (ЧЧ:ММ:СС). Первый параметр (T) определяет строку времени в формате, описанном в разделе «Форматы времени». После строки времени можно указать любое число модификаторов. Модификаторы перечислены в разделе «Модификаторы». |
| DATETIME(T, ...) | Функция DATETIME() возвращает строку, содержащую дату и время в формате YYYY-MM-DD HH:MM:SS (ГГГГ-ММ-ДД ЧЧ:ММ:СС). Первый параметр (T) определяет строку времени в формате, описанном в разделе «Форматы времени». После строки времени можно указать любое число модификаторов. Модификаторы перечислены в разделе «Модификаторы». |
| JULIANDAY(T, ...) | Функция JULIANDAY() возвращает число, обозначающее количество дней, прошедшее с полудня по Гринвичу 24 ноября 4714 года до нашей эры, и заданную дату. Первый параметр (T) определяет строку времени в формате, описанном в разделе «Форматы времени». После строки времени можно указать любое число модификаторов. Модификаторы перечислены в разделе «Модификаторы». |
| STRFTIME(F, T, ...) | Подпрограмма STRFTIME() возвращает дату, отформатированную в соответствии со строкой формата, указанной в качестве первого аргумента F. Строка формата поддерживает следующие подстановки:<br>%d — день месяца<br>%f — дробные секунды SS.SSS<br>%H — час 00-24<br>%j — день года 001-366<br>%J — номер дня по юлианскому календарю<br>%m — месяц 01-12<br>%M — минута 00-59<br>%s — число секунд с 1970-01-01<br>%S — секунды 00-59<br>%w — день недели 0-6 (воскресенье = 0)<br>%W — неделя года 00-53<br>%Y — год 0000-9999<br>%% - %<br>Второй параметр (T) определяет строку времени в формате, описанном в разделе «Форматы времени». После строки времени можно указать любое число модификаторов. Модификаторы перечислены в разделе «Модификаторы». |

### Форматы времени

Строка времени может иметь любой из перечисленных ниже форматов.

| | |
|---|---|
| YYYY-MM-DD | 2007-06-15 |
| YYYY-MM-DD HH:MM | 2007-06-15 07:30 |
| YYYY-MM-DD HH:MM:SS | 2007-06-15 07:30:59 |
| YYYY-MM-DD HH:MM:SS.SSS | 2007-06-15 07:30:59.152 |

| | |
|---|---|
| YYYY-MM-DDTHH:MM | 2007-06-15T07:30 |
| YYYY-MM-DDTHH:MM:SS | 2007-06-15T07:30:59 |
| YYYY-MM-DDTHH:MM:SS.SSS | 2007-06-15T07:30:59.152 |
| HH:MM | 07:30 (дата 2000-01-01) |
| HH:MM:SS | 07:30:59 (дата 2000-01-01) |
| HH:MM:SS.SSS | 07:30:59:152 (дата 2000-01-01) |
| now | Текущая дата и время по универсальному времени (UCT). |
| DDDD.DDDD | Номер дня по юлианскому календарю, представленный в виде числа с плавающей запятой. |

Символ T в этих форматах является символом литерала «T», который разделяет дату и время. Форматы, которые содержат только время, предполагают дату 2001-01-01.

### Модификаторы

После строки времени может следовать ноль или больше модификаторов, которые изменяют дату или ее интерпретацию. Доступные модификаторы перечислены ниже.

| | |
|---|---|
| NNN days | Число дней, которые требуется добавить ко времени. |
| NNN hours | Число часов, которые требуется добавить ко времени. |
| NNN minutes | Число минут, которые требуется добавить ко времени. |
| NNN.NNNN seconds | Число секунд и миллисекунд, которые требуется добавить ко времени |
| NNN months | Число месяцев, которые требуется добавить ко времени. |
| NNN years | Число лет, которые требуется добавить ко времени. |
| start of month | Смещает время назад к началу месяца. |
| start of year | Смещает время назад к началу года. |
| start of day | Смещает время назад к началу дня. |
| weekday N | Смещает время вперед к указанному дню недели. (0 = воскресенье, 1 = понедельник и т. д.) |
| localtime | Преобразует дату в местное время. |
| utc | Преобразует дату в универсальное время (UCT). |

## Операторы

SQL поддерживает большой набор операторов, включая общие операторы, которые используются в большинстве языков программирования, а также несколько операторов, которые применяются только в SQL.

### Общие операторы

В блоках SQL разрешается использовать следующие двоичные операторы, перечисленные в порядке приоритета от самого высокого до самого низкого.

```
*     /     %
+     -
<<  >>  &  |
<  >=    >  >=
=     ==    !=    <>  IN
AND
OR
```

Ниже перечислены поддерживаемые унарные префиксные операторы.

```
 !     ~     NOT
```

Оператор COLLATE можно рассматривать как унарный постфиксный оператор. Оператор COLLATE имеет самый высокий приоритет. Он всегда обеспечивает более крепкую привязку, чем любой другой унарный префиксный оператор или двоичный оператор.

Обратите внимание, что существует два варианта операторов равенства и неравенства. Равенство может быть выражено как = или ==. В качестве оператора неравенства может использоваться != или <>.

Оператор || является оператором сцепления строк, он объединяет две строки операндов.

Оператор % выводит остаток левого операнда по модулю правого операнда.

Результат любого двоичного оператора является числовым значением, за исключением оператора сцепления ||, результатом которого является строка.

**Операторы SQL**

**LIKE**

Оператор LIKE выполняет сравнение для сопоставления с шаблоном.

```
expr     ::=  (column-name | expr) LIKE pattern
pattern  ::=  '[ string | % | _ ]'
```

Операнд справа от оператора LIKE содержит шаблон, а левый операнд содержит строку, которую требуется сравнить с шаблоном. Символ процента (%) в шаблоне является символом подстановки. Он заменяет любую последовательность из нуля или более символов в строке. Символ подчеркивания (_) в шаблоне соответствует любому одному символу в строке. Любой другой символ соответствует самому себе или своему эквиваленту нижнего или верхнего регистра, то есть сопоставление выполняется без учета регистра. (Примечание. Ядро базы данных распознает верхний и нижний регистр только для 7-битовых латинских символов. По этой причине оператор LIKE учитывает регистр для 8-битовых символов iso8859 и для символов UTF-8. Например, выражение 'a' LIKE 'A' возвращает значение TRUE, а 'æ' LIKE 'Æ' возвращает FALSE). Чувствительность к регистру латинских символов можно изменить с помощью свойства SQLConnection.caseSensitiveLike.

Если присутствует необязательное предложение ESCAPE, то выражение, следующее за ключевым словом ESCAPE, должно содержать строку, состоящую из одного символа. Этот символ можно использовать в шаблоне LIKE для сопоставления символов процента или прочерка литерала. Символ маскировки, за которым следует символ процента, прочерка или он сам, соответствует символу процента, прочерка или маскировки литерала в строке.

**GLOB**

Оператор GLOB похож на LIKE, но использует синтаксис глобализации файлов Unix для подстановочных знаков. В отличие от LIKE, оператор GLOB учитывает регистр.

IN

Оператор IN вычисляет, равен ли левый операнд одному из значений в правом операнде (из набора значений в круглых скобках).

```
in-expr         ::=  expr [NOT] IN ( value-list ) |
                     expr [NOT] IN ( select-statement ) |
                     expr [NOT] IN [database-name.] table-name
value-list      ::=  literal-value [, literal-value]*
```

Правый операнд может содержать разделенные запятыми значения литералов или результат инструкции SELECT. Объяснения и ограничения использования инструкции SELECT в качестве правого операнда оператора IN см. в подразделе «Инструкция SELECT» раздела «Выражения».

**BETWEEN...AND**

Оператор BETWEEN...AND равносилен использованию двух выражений с операторами >= и <=. Например, выражение x BETWEEN y AND z эквивалентно выражению x >= y AND x <= z.

**NOT**

Оператор NOT выполняет функцию отрицания. Перед операторами GLOB, LIKE и IN может стоять ключевое слово NOT, которое меняет смысл проверки на обратный (другими словами, требуется проверить, что значение не соответствует указанному шаблону).

## Параметры

Параметр указывает местозаполнитель в выражении для значения литерала, который заполняется во время выполнения путем присвоения значения в ассоциативный массив SQLStatement.parameters. Параметры могут принимать три формы.

| | |
|---|---|
| ? | Знак вопроса обозначает индексированный параметр. Параметрам присваиваются числовые значения индекса (от нуля) в соответствии с их порядком в инструкции. |
| :AAAA | Двоеточие, за которым следует имя идентификатора, сохраняет место для именованного параметра с именем AAAA. Именованные параметры также нумеруются в соответствии с их порядком в инструкции SQL. Во избежание путаницы лучше не смешивать именованные параметры с нумерованными. |
| @AAAA | Символ «собака» равносилен двоеточию. |

## Неподдерживаемые возможности SQL

Ниже приводится список стандартных элементов SQL, которые не поддерживаются в Adobe AIR.

**Ограничения FOREIGN KEY** Ограничения FOREIGN KEY подвергаются синтаксическому анализу, но не применяются.

**Триггеры** Триггеры FOR EACH STATEMENT не поддерживаются (все триггеры должны быть FOR EACH ROW). Триггеры INSTEAD OF не поддерживаются в таблицах (их использование допускается только в видах). Рекурсивные триггеры (то есть те, которые запускают сами себя) не поддерживаются.

**ALTER TABLE** Поддерживаются только варианты RENAME TABLE и ADD COLUMN команды ALTER TABLE. Другие типы операций ALTER TABLE, такие как DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT и т. д., игнорируются.

**Вложенные транзакции** Допускается только одна активная транзакция.

**RIGHT и FULL OUTER JOIN** RIGHT OUTER JOIN и FULL OUTER JOIN не поддерживаются.

**Обновляемый вид** Виды доступны только для чтения. Для видов нельзя выполнять инструкции DELETE, INSERT или UPDATE. Поддерживается триггер INSTEAD OF, который запускается при попытке удаления, вставки или обновления вида (DELETE, INSERT или UPDATE). С его помощью можно обновлять базовые таблицы в теле триггера.

**GRANT и REVOKE** База данных представляет собой обычный файл на диске. К ней можно применять только обычные права доступа к файлу, которые поддерживаются существующей операционной системой. Команды GRANT и REVOKE, которые широко используются в клиентских и серверных RDBMS, не реализуются.

Следующие элементы SQL и возможности SQLite поддерживаются в некоторых версиях SQLite, но не поддерживаются в Adobe AIR. Большинство этих возможностей обеспечиваются с помощью методов класса SQLConnection.

**Элементы SQL, связанные с транзакциями (BEGIN, END, COMMIT, ROLLBACK)** Эти возможности обеспечиваются с помощью методов класса SQLConnection, связанных с транзакциями: SQLConnection.begin(), SQLConnection.commit() и SQLConnection.rollback().

**ANALYZE** Эта функция обеспечивается с помощью метода SQLConnection.analyze().

**ATTACH** Эта функция обеспечивается с помощью метода SQLConnection.attach().

**COPY** Эта инструкция не поддерживается.

**CREATE VIRTUAL TABLE** Эта инструкция не поддерживается.

**DETACH** Эта функция обеспечивается с помощью метода SQLConnection.detach().

**PRAGMA** Эта инструкция не поддерживается.

**VACUUM** Эта функция обеспечивается с помощью метода SQLConnection.compact().

**Доступ к системным таблицам не поддерживается.** Системные таблицы, включая sqlite_master и другие таблицы с префиксом «sqlite_», недоступны в инструкциях SQL. Среда выполнения включает API-интерфейс схемы, который обеспечивает объектно-ориентированный метод доступа к данным схемы. Дополнительные сведения см. в описании метода SQLConnection.loadSchema().

**Функции регулярных выражений (MATCH() и REGEX())** Эти функции недоступны в инструкциях SQL.

Следующие функциональные возможности отличаются во многих версиях SQLite и Adobe AIR.

**Индексированные параметры инструкций** Во многих версиях параметры инструкций индексируются от единицы. Однако в Adobe AIR параметры инструкции индексируются от нуля (то есть первому параметру присваивается индекс 0, второму — 1 и т. д.).

**Определения столбцов INTEGER PRIMARY KEY** Во многих версиях в качестве действительного столбца первичного ключа таблицы используются только столбцы, которые явно определены как INTEGER PRIMARY KEY. В таких версиях указание других типов данных, которые являются синонимами INTEGER (например, int), не приводит к тому, что столбец используется в качестве внутреннего первичного ключа. Однако в Adobe AIR тип данных int (или другие синонимы INTEGER) рассматриваются как точное соответствие INTEGER. По этой причине столбец, определенный как int PRIMARY KEY, используется в качестве внутреннего первичного ключа таблицы. Дополнительные сведения см. в разделах «CREATE TABLE» и «Сходство столбцов».

## Дополнительные возможности SQL

Следующие типы сходства столбцов по умолчанию в SQLite не поддерживаются, но поддерживаются в Adobe AIR. (Обратите внимание, что, как и во всех ключевых словах в SQL, в этих именах типов данных не учитывается регистр).

**Boolean** соответствует классу Boolean.

**Date** соответствует классу Date.

**int** соответствует классу int (эквивалентно сходству столбца INTEGER).

**Number** соответствует классу Number (эквивалентно сходству столбца REAL).

**Object** соответствует классу Object или любому подклассу, сериализацию и десериализацию которого можно выполнить с помощью AMF3. (Сюда относится большинство классов, в том числе пользовательских, кроме некоторых классов, которые включают экранные объекты и объекты и объекты, свойства которых содержат экранные объекты.)

**String** соответствует классу String (эквивалентно сходству столбца TEXT).

**XML** соответствует классу XML в ActionScript (E4X).

**XMLList** соответствует классу XMLList в ActionScript (E4X).

Следующие значения литералов не поддерживаются по умолчанию в SQLite, но поддерживаются в Adobe AIR.

**true** используется для представления логического значения литерала true, предназначено для работы со столбцами BOOLEAN.

**false** используется для представления логического значения литерала false, предназначено для работы со столбцами BOOLEAN.

# Поддержка типов данных

В отличие от большинства баз данных SQL, модуль базы данных SQL в Adobe AIR не требует, чтобы столбцы таблиц содержали значения определенного типа. Вместо этого среда выполнения использует для управления типами данных две концепции: классы хранения и сходство столбцов. В этом разделе описываются классы хранения и сходство столбцов, а также способы компенсации различий между типами данных в различных ситуациях.


- «Классы хранения» на странице 1218
- «Сходство столбцов» на странице 1219
- «Типы данных и операторы сравнения» на странице 1221
- «Типы данных и математические операторы» на странице 1222
- «Типы данных и сортировка» на странице 1222
- «Типы данных и группировка» на странице 1222
- «Типы данных и составные инструкции SELECT» на странице 1223


## Классы хранения

Классы хранения представляют действительные типы данных, которые служат для хранения значений в базе данных. База данных использует следующие классы хранения.

**NULL** Значение — NULL.

**INTEGER** Значение — целое число со знаком.

**REAL** Значение — число с плавающей запятой.

**TEXT** Значение — текстовая строка (размер до 256 МБ).

**BLOB** Это значение представляет собой большой двоичный объект (англ. Binary Large Object, BLOB); другими словами, это необработанные двоичные данные (максимальный размер — 256 МБ).

Все значения, которые передаются в базу данных как литералы, встроенные в инструкцию SQL, или значения, связанные с подготовленной инструкцией SQL с помощью параметров, попадают в классы хранения до выполнения инструкции SQL.

Литералы, которые входят в состав инструкции SQL, относятся к классу хранения TEXT, если они заключены в одинарные или двойные кавычки, к классу INTEGER, если литерал определен как число без десятичного разделителя или экспоненты, не заключенное в кавычки, к классу REAL, если литерал является числом с десятичным разделителем или экспонентой, заключенным в кавычки, или к классу NULL, если литерал имеет значение NULL. Литералы с классом хранения BLOB определяются с использованием формата записи X'ABCD'. Дополнительные сведения см. в подразделе «Значения литералов» раздела «Выражения».

Значения, передаваемые как параметры с помощью ассоциативного массива SQLStatement.parameters, относятся к классу хранения, который ближе всего соответствует назначенному им собственному типу данных. Например, значения int относятся к классу хранения INTEGER, значения Number — к классу хранения REAL, значения String — к классу хранения TEXT, а объекты ByteArray — к классу хранения BLOB.

## Сходство столбцов

*Сходство* столбца — это рекомендуемый тип для данных в таком столбце. Когда значение, сохраненное в столбце (с помощью инструкции INSERT или UPDATE), среда выполнения пытается преобразовать это значение из текущего типа данных в заданное сходство. Например, если значение Date (экземпляр Date в ActionScript или JavaScript) вставляется в столбец со сходством TEXT, перед сохранением в базе данных значение Date преобразуется в строковое представление String (что равносильно вызову метода объекта toString()). Если значение не удается преобразовать к заданному сходству, происходит ошибка и операция не выполняется. Когда значение получается из базы данных с помощью инструкции SELECT, оно возвращается как экземпляр класса, соответствующего сходству, независимо от того, было ли оно преобразовано из другого типа данных при сохранении.

Если столбец принимает значения NULL, нулевое значение ActionScript или JavaScript (null) может использоваться в качестве значения параметра для сохранения NULL в столбце. Когда значение класса хранения NULL извлекается из инструкции SELECT, оно всегда возвращается как нулевое значение ActionScript или JavaScript (null) независимо от сходства столбца. Если столбец принимает значения NULL, следует всегда проверять значения, извлекаемые из этого столбца, чтобы определить, равны ли они нулю, прежде чем выполнить попытку приведения значения к ненулевому типу (такому как Number или Boolean).

Каждому столбцу в базе данных назначается один из следующих сходств типа:

- TEXT (или String)
- NUMERIC
- INTEGER (или int)
- REAL (или Number)
- Boolean
- Date
- XML
- XMLLIST
- Object
- NONE

**TEXT (или String)**

Столбец со сходством TEXT или String хранит все данные, используя классы хранения NULL, TEXT или BLOB. Если числовые данные вставляются в столбец со сходством TEXT, перед сохранением они преобразуются в текст.

**NUMERIC**

Столбец со сходством NUMERIC содержит значения, используя классы хранения NULL, REAL или INTEGER. Когда текстовые данные вставляются в столбец NUMERIC, перед сохранением предпринимается попытка преобразовать их в целое или действительное число. В случае успешного преобразования значение сохраняется в классе хранения INTEGER или REAL (например, значение '10.05' перед сохранением преобразуется в класс хранения REAL). Если преобразование выполнить не удается, возникает ошибка. Если значение — NULL, попытка преобразования не предпринимается. Значение, полученное из столбца NUMERIC, возвращается как экземпляр самого точного числового типа, к которому можно отнести значение. Другими словами, если значение является положительным целым числом или 0, возвращается экземпляр uint. Если значение является отрицательным целым числом, возвращается экземпляр int. А если значение имеет часть с плавающей запятой (то есть не является целым числом), возвращается экземпляр Number.

**INTEGER (или int)**

По поведению столбец со сходством INTEGER напоминает столбец со сходством NUMERIC, но с одним исключением. Если сохраняемое значение является действительным числом (таким как экземпляр Number) без части после плавающей запятой или текстом, который можно преобразовать в действительное число без плавающей запятой, то оно преобразуется в целое число и сохраняется с использованием класса хранения INTEGER. При попытке сохранить действительное число с частью после плавающей запятой возникает ошибка.

**REAL (или Number)**

Столбец со сходством REAL или NUMBER ведет себя так же, как столбец с родственным типом NUMERIC. Однако здесь целочисленные значения принудительно преобразуются в представление с плавающей запятой. Значение в столбце REAL всегда возвращается из базы данных как экземпляр Number.

**Boolean**

Столбец со сходством Boolean хранит значения true (истина) или false (ложь). Столбец Boolean принимает значение, которое является экземпляром Boolean ActionScript или JavaScript. Если код пытается сохранить значение String, объект String с длиной больше нуля приравнивается значению true, а пустой объект String — значению false. Если код пытается сохранить числовые данные, любое ненулевое значение сохраняется как true, а 0 — как false. Когда логическое значение (Boolean) извлекается с помощью инструкции SELECT, оно возвращается как экземпляр Boolean. Ненулевые значения хранятся с использованием класса хранения INTEGER (0 — false и 1 — true) и преобразуются в объекты Boolean при извлечении данных.

**Date**

Столбец со сходством Date хранит значения даты и времени. Столбец Date может принимать значения, которые являются экземплярами Date в ActionScript или JavaScript. При попытке сохранить значение String в столбце Date среда выполнения преобразует его в дату по юлианскому календарю. Если выполнить преобразование не удается, выдается сообщение об ошибке. Если код пытается сохранить значение Number, int или uint, проверка данных не выполняется, а значение рассматривается как действительная дата по юлианскому календарю. Значение Date, полученное с помощью инструкции SELECT, автоматически преобразуется в экземпляр Date. Значения Date сохраняются как значения даты по юлианскому календарю с использованием класса хранения REAL, поэтому операции сравнения и сортировки выполняются в обычном порядке.

**XML или XMLList**

Столбец со сходством XML или XMLList содержит структуры XML. Когда код пытается сохранить данные в столбце XML при помощи параметра SQLStatement, среда выполнения пытается преобразовать и проверить значение с помощью функции ActionScript XML() или XMLList(). Если значение невозможно преобразовать в допустимый объект XML, происходит ошибка. При попытке сохранить данные с использованием текстового значения SQL литерала (например, INSERT INTO (col1) VALUES ('Invalid XML (no closing tag)'), синтаксический анализ или проверка значения не выполняется, поскольку его формат считается правильным. Если сохранить недопустимое значение, то при извлечении оно возвращает пустой объект XML. Значение Data XML и XMLList сохраняется с использованием класса хранения TEXT или NULL.

**Object**

Столбец со сходством Object содержит сложные объекты ActionScript или JavaScript, включая экземпляры класса Object, а также экземпляры подклассов Object, такие как экземпляры Array и даже экземпляры пользовательских классов. Сериализация данных столбца Object выполняется в формате AMF3, и они сохраняются с использованием класса хранения BLOB. При извлечении значения выполняется его десериализация из AMF3, и оно возвращается в виде экземпляра класса в том виде, в котором оно было

сохранено. Обратите внимание, что некоторые классы ActionScript, в особенности объекты представления, можно десериализовать как экземпляры их исходного типа данных. Прежде чем сохранять экземпляр пользовательского класса, необходимо зарегистрировать псевдоним класса с помощью метода flash.net.registerClassAlias() (а в Flex можно добавить в объявление класса метаданные [RemoteObject]). Кроме того, перед извлечением таких данных необходимо зарегистрировать такой же псевдоним для класса. Любые данные, для которых нельзя правильно выполнить десериализацию по той причине, что класс изначально не поддается десериализации или псевдоним класса отсутствует или несогласован, возвращаются как анонимный объект (экземпляр класса Object) со свойствами и значениями, соответствующими изначально сохраненному экземпляру.

**NONE**

Столбец со сходством NONE не различает классы хранения. Попытка преобразования данных перед вставкой не предпринимается.

**Определение сходства**

Сходство типа столбца определяется по типу столбца, объявленному в инструкции CREATE TABLE. При определении типа действуют следующие правила (без учета регистра).

- Если тип данных столбца содержит любые строки из CHAR, CLOB, STRI или TEXT, то столбец имеет сходство TEXT/String. Обратите внимание, что тип VARCHAR содержит строку CHAR и ему назначается сходство TEXT.
- Если тип данных столбца содержит строку BLOB или тип данных не задан, то столбец имеет сходство NONE.
- Если тип данных столбца содержит строку XMLL, то столбец имеет сходство XMLList.
- Если тип данных столбца содержит строку XML, то столбец имеет сходство XML.
- Если тип данных содержит строку OBJE, то столбец имеет сходство Object.
- Если тип данных содержит строку BOOL, то столбец имеет сходство Boolean.
- Если тип данных содержит строку DATE, то столбец имеет сходство Date.
- Если тип данных содержит строку INT (включая UINT), ему назначается сходство INTEGER/int.
- Если тип данных столбца содержит любые строки из REAL, NUMB, FLOA или DOUB, то столбец имеет сходство REAL/Number.
- В противном случае сходством будет NUMERIC.
- Если таблица создана с помощью инструкции CREATE TABLE t AS SELECT..., то тип данных для всех столбов не задается, и им назначается сходство NONE.

## Типы данных и операторы сравнения

Поддерживаются операторы двоичного сравнения =, <, <=, >= и !=, а также операция проверки заданного членства, оператор IN и оператор троичного сравнения BETWEEN. Дополнительные сведения об этих операторах см. в разделе «Операторы».

Результат сравнения зависит от классов хранения двух сравниваемых значений. При сравнении двух значений действуют следующие правила.

- Значение с классом хранения NULL считается меньшим по отношению к любому другому значению (включая другое значение с классом хранения NULL).

- Значение INTEGER или REAL меньше любого значения TEXT или BLOB. При сравнении INTEGER или REAL с другим значением INTEGER или REAL выполняется сравнение чисел.
- Значение TEXT меньше значения BLOB. При сравнении двух значений TEXT выполняется двоичное сравнение.
- При сравнении двух значений BLOB результат всегда определяется путем двоичного сравнения.

Троичный оператор BETWEEN всегда преобразуется в эквивалентное двоичное выражение. Например, a BETWEEN b AND c преобразуется в a >= b AND a <= c, даже если это означает, что к a применяются разные сходства при каждом сравнении, необходимом для вычисления выражения.

Выражения типа a IN (SELECT b ....) следуют трем правилам, перечисленным ранее для двоичного сравнения, то есть аналогичны a = b. Например, если b — это значение столбца, а a — выражение, то сходство b применяется к a перед любым сравнением. Выражение a IN (x, y, z) преобразуется к a = +x OR a = +y OR a = +z. Значения справа от оператора IN (в приведенном примере это значения x, y и z) считаются выражениями, даже если они являются значениями столбца. Если значение слева от оператора IN является столбцом, то используется сходство этого столбца. Если значение является выражением, то преобразование не выполняется.

На порядок выполнения сравнения может также влиять предложение COLLATE. Дополнительные сведения см. в разделе «COLLATE».

## Типы данных и математические операторы

Для каждого из поддерживаемых математических операторов *, /, %, + и - перед вычислением выражения к каждому операнду применяется сходство NUMERIC. Если какой-либо операнд не удается успешно преобразовать в класс хранения NUMERIC, результатом выражения будет NULL.

При использовании оператора сцепления перед вычислением выражения || каждый операнд преобразуется в класс хранения TEXT. Если какой-либо операнд не удается преобразовать в класс хранения TEXT, результатом выражения будет NULL. Значение невозможно преобразовать в двух случаях: если значение операнда — NULL, или если это объект BLOB с классом хранения, отличным от TEXT.

## Типы данных и сортировка

При сортировке значений при помощи предложения ORDER BY на первое место ставится класс хранения NULL. Затем следуют значения INTEGER и REAL в числовом порядке, далее значения TEXT в двоичном порядке или в зависимости от заданной сортировки (BINARY или NOCASE). Завершают список значения BLOB в двоичном порядке. Перед сортировкой преобразования классов хранения не выполняется.

## Типы данных и группировка

При группировке значений с помощью предложения GROUP BY значения с различными классами хранения считаются разными. Исключением являются значения INTEGER и REAL, которые считаются равными, если их числа эквивалентны. При использовании предложения GROUP BY к значениям не применяется сходство.

## Типы данных и составные инструкции SELECT

Составные операторы SELECT (UNION, INTERSECT и EXCEPT) выполняют неявное сравнение значений. Перед выполнением этих сравнений к каждому значению может применяться сходство. Ко всем значениям, которые могут быть возвращены в одном столбце набора результатов составной инструкции SELECT, применяется одно и то же сходство, если имеется. Применяемое сходство является сходством столбца, который возвращается первым компонентом инструкции SELECT и имеет значение столбца (а не какой-либо другой тип выражения) в этом положении. Если для заданного столбца составной инструкции SELECT ни один из компонентов инструкции SELECT не возвращает значение столбца, то к значениям этого столбца не применяется сходство перед сравнением.

# Глава 67. Сообщения со сведениями об ошибках SQL, идентификаторы и аргументы

Класс SQLError представляет различные ошибки, которые могут произойти при работе с локальной базой данных SQL Adobe AIR. Для каждого исключения экземпляр SQLError имеет свойство `details`, содержащее сообщение об ошибке на английском языке. Кроме того, каждое сообщение об ошибке имеет соответствующий уникальный идентификатор, который содержится в свойстве `detailID` объекта SQLError. С помощью свойства `detailID` приложение может идентифицировать конкретное сообщение об ошибке details. Приложение может предоставить альтернативный текст для конечного пользователя в его локали. Значения аргумента в массиве `detailArguments` могут быть заменены в соответствующей позиции в строке сообщения об ошибке. Эту возможность целесообразно использовать в приложениях, которые отображают сообщения об ошибках свойства `details` напрямую конечному пользователю, находящемуся в конкретной локали.

Следующая таблица содержит список значений `detailID` и связанный текст сообщения об ошибке на английском языке. Текст метки-заполнителя в сообщениях указывает, где во время выполнения заменяются значения `detailArguments`. Этот список может использоваться в качестве источника для локализации сообщений об ошибках, которые могут произойти в операциях базы данных SQL.

| SQLError detailID | Сообщение с подробной информацией об ошибке и ее параметрах на английском языке |
|---|---|
| 1001 | Соединение закрыто. |
| 1102 | Для выполнения этой операции база данных должна быть открыта. |
| 1003 | Имя (имена) параметров %s [,\|и %s] найдены в свойстве параметров, но не в указанном SQL. |
| 1004 | Несоответствие количества параметров. В указанном SQL найдено %d и значение (значения) %d установлены в свойстве параметров.Ожидались значения для %s [,\|и %s]. |
| 1005 | Не удалось включить автоматическое сжатие. |
| 1006 | Не удалось установить значение pageSize. |
| 1007 | Объект схемы с именем «%s» типа «%s» не найден в базе данных «%s». |
| 1008 | Объект схемы с именем «%s» в базе данных «%s» не найден. |
| 1009 | Объекты схемы с типом «%s» не найдены в базе данных «%s». |
| 1010 | Объекты схемы не найдены в базе данных «%s». |
| 2001 | Parser stack overflow. (Переполнение стека синтаксического анализатора.) |
| 2002 | Too many arguments on function '%s' (Слишком много аргументов для функции «%s».) |
| 2003 | Рядом с «%s»: синтаксическая ошибка |
| 2004 | Таблица или индекс с этим именем уже существует: «%s» |
| 2005 | PRAGMA в SQL недопустима. |
| 2006 | Not a writable directory. (Этот каталог защищен от записи.) |
| 2007 | Unknown or unsupported join type: '%s %s %s' (Неизвестный или неподдерживаемый тип соединения: «%s %s %s».) |
| 2008 | Объединения RIGHT и FULL OUTER не поддерживаются настоящее время. |
| 2009 | A NATURAL join may not have an ON or USING clause. (Соединение типа NATURAL не должно содержать предложения ON или USING.) |
| 2010 | Cannot have both ON and USING clauses in the same join. (Одно соединение не может содержать либо предложение ON, либо USING, но не оба сразу.) |
| 2011 | Cannot join using column '%s' - column not present in both tables. (Невозможно выполнить соединение с использованием столбца «%s», так как он отсутствует в одной из таблиц.) |

| | |
|---|---|
| 2012 | Only a single result allowed for a SELECT that is part of an expression. (Разрешается только один результат для предложения SELECT, которое входит в состав выражения.) |
| 2013 | No such table: '[%s.]%s' (Таблица не существует: «[%s.]%s».) |
| 2014 | No tables specified. (Таблицы не указаны.) |
| 2015 | Too many columns in result set\|too many columns on '%s'. (Слишком много столбцов в наборе результатов\|слишком много столбцов в «%s».) |
| 2016 | Термин %s ORDER\|GROUP BY вне диапазона, должен быть между 1 и %d |
| 2017 | Слишком много терминов в предложении ORDER BY. (Слишком много терминов в предложении ORDER BY.) |
| 2018 | %s ORDER BY term out of range - should be between 1 and %d. (Термин %s в ORDER BY выходит за рамки диапазона 1 – %d.) |
| 2019 | %r ORDER BY term does not match any column in the result set. (Термин %r в ORDER BY не соответствует ни одному столбцу в наборе результатов.) |
| 2020 | ORDER BY clause should come after '%s' not before. (Предложение ORDER BY должно стоять после «%s», а не перед.) |
| 2021 | LIMIT clause should come after '%s' not before. (Предложение LIMIT должно стоять после «%s», а не перед.) |
| 2022 | SELECT слева и справа от «%s» не имеют одинаковое количество столбцов результата. |
| 2023 | A GROUP BY clause is required before HAVING. (Перед HAVING должно быть предложение GROUP BY.) |
| 2024 | Aggregate functions are not allowed in the GROUP BY clause. (В предложении GROUP BY запрещено использовать агрегатные функции.) |
| 2025 | DISTINCT в составном элементе должен сопровождаться выражением. |
| 2026 | Too many terms in compound SELECT (Слишком много терминов в составном предложении SELECT.) |
| 2027 | Too many terms in ORDER\|GROUP BY clause (Слишком много терминов в предложении ORDER\|GROUP BY.) |
| 2028 | Temporary trigger may not have qualified name (Временный триггер не может иметь квалифицированного имени.) |
| 2030 | Trigger '%s' already exists (Триггер «%s» уже существует.) |
| 2032 | Cannot create BEFORE\|AFTER trigger on view: '%s'. (Невозможно создать триггер BEFORE\|AFTER для вида: «%s».) |
| 2033 | Cannot create INSTEAD OF trigger on table: '%s'. (Невозможно создать триггер INSTEAD OF для таблицы: «%s».) |
| 2034 | No such trigger: '%s' (Триггер не существует: «%s».) |
| 2035 | Recursive triggers not supported ('%s'). (Рекурсивные триггеры не поддерживаются (%s).) |
| 2036 | No such column: %s[.%s[.%s]] (Столбец не существует: %s[.%s[.%s]]) |
| 2037 | VACUUM недопустим из SQL. |
| 2043 | Table '%s': indexing function returned an invalid plan. (Таблица «%s»: функция индексирования вернула недействительный план.) |
| 2044 | At most %d tables in a join. (В соединении может быть не более %d таблиц.) |
| 2046 | Не удалось добавить столбец PRIMARY KEY. |
| 2047 | Не удалось добавить столбец UNIQUE. |
| 2048 | Не удалось добавить столбец NOT NULL со значением по умолчанию NULL. |
| 2049 | Не удалось добавить столбец со значением по умолчанию, не являющимся константой. |
| 2050 | Не удалось добавить столбец к представлению. |
| 2051 | ANALYZE недопустим в SQL. |
| 2052 | Invalid name: '%s' (Недопустимое имя: «%s».) |
| 2053 | ATTACH недопустим из SQL. |
| 2054 | %s «%s» не может сослаться на объекты в базе данных «%s» |
| 2055 | Access to '[%s.]%s.%s' is prohibited. (Доступ к «[%s.]%s.%s» запрещен.) |
| 2056 | Not authorized. (Не выполнена авторизация.) |
| 2058 | No such view: '[%s.]%s' (Вид не существует: [%s.]%s.) |
| 2060 | Temporary table name must be unqualified. (Временное имя таблицы должно быть неквалифицированным.) |
| 2061 | Table '%s' already exists. (Таблица «%s» уже существует.) |

| | |
|---|---|
| 2062 | There is already an index named: '%s' (Индекс с таким именем уже существует: «%s».) |
| 2064 | Duplicate column name: '%s' (Повторяющееся имя столбца: «%s».) |
| 2065 | Table '%s' has more than one primary key. (Таблица «%s» содержит несколько первичных ключей.) |
| 2066 | AUTOINCREMENT разрешен только для INTEGER PRIMARY KEY |
| 2067 | No such collation sequence: '%s' (Последовательность сортировки не существует: «%s».) |
| 2068 | Parameters are not allowed in views. (В видах не допускается использование параметров.) |
| 2069 | View '%s' is circularly defined. (Вид «%s» определен в циркулярном порядке.) |
| 2070 | Table '%s' may not be dropped. (Невозможно удалить таблицу «%s».) |
| 2071 | Use DROP VIEW to delete view '%s' (Используйте DROP VIEW для удаления вида «%s».) |
| 2072 | Use DROP TABLE to delete table '%s' (Используйте DROP TABLE для удаления таблицы «%s».) |
| 2073 | Foreign key on '%s' should reference only one column of table '%s' (Внешний ключ в «%s» может ссылаться только на один столбец таблицы «%s».) |
| 2074 | Number of columns in foreign key does not match the number of columns in the referenced table. (Число столбцов во внешнем ключе не соответствует числу столбцов в упоминаемой таблице.) |
| 2075 | Unknown column '%s' in foreign key definition. (Неизвестный столбец «%s» в определении внешнего ключа.) |
| 2076 | Table '%s' may not be indexed. (Невозможно проиндексировать таблицу «%s».) |
| 2077 | Views may not be indexed. (Невозможно проиндексировать виды.) |
| 2080 | Conflicting ON CONFLICT clauses specified. (Указаны конфликтующие предложения ON CONFLICT.) |
| 2081 | No such index: '%s' (Индекс не существует: «%s».) |
| 2082 | Index associated with UNIQUE or PRIMARY KEY constraint cannot be dropped. (Невозможно пропустить индекс, связанный с ограничением UNIQUE или PRIMARY KEY.) |
| 2083 | BEGIN в SQL недопустим. |
| 2084 | COMMIT в SQL недопустим. |
| 2085 | ROLLBACK в SQL недопустим. |
| 2086 | Unable to open a temporary database file for storing temporary tables. (Невозможно открыть временный файл базы данных для хранения временных таблиц.) |
| 2087 | Unable to identify the object to be reindexed. (Невозможно определить объект для повторного индексирования.) |
| 2088 | Table '%s' may not be modified. (Невозможно изменить таблицу «%s».) |
| 2089 | Cannot modify '%s' because it is a view. (Невозможно изменить «%s», так как это вид.) |
| 2090 | Variable number must be between ?0 and ?%d< (Переменное число должно находиться в диапазоне от ?0 до ?%d<.) |
| 2092 | Misuse of aliased aggregate '%s' (Недопустимое использование агрегата с псевдонимом «%s».) |
| 2093 | Ambiguous column name: '[%s.[%s.]]%s' (Неоднозначное имя столбца: [%s.[%s.]]%s.) |
| 2094 | No such function: '%s' (Функция не существует: «%s».) |
| 2095 | Wrong number of arguments to function '%s' (Неверное число аргументов, переданных функции «%s».) |
| 2096 | Subqueries prohibited in CHECK constraints. (В ограничениях CHECK не допускается использование вложенных запросов.) |
| 2097 | Parameters prohibited in CHECK constraints. (В ограничениях CHECK не допускается использование параметров.) |
| 2098 | Дерево выражений слишком велико (максимальная глубина %d) |
| 2099 | RAISE() может использоваться только внутри программы-триггера |
| 2100 | Table '%s' has %d columns but %d values were supplied (В таблице «%s» число столбцов (%d) не соответствует числу предоставленных значений (%d).) |
| 2101 | Database schema is locked: '%s' (Схема базы данных заблокирована: «%s».) |
| 2102 | Statement too long. (Слишком длинная инструкция.) |

| | |
|---|---|
| 2103 | Не удалось удалить/изменить последовательность сравнений вследствие наличия активных операторов |
| 2104 | Too many attached databases - max %d (Слишком много присоединенных баз данных. Должно быть не более %d.) |
| 2105 | Cannot ATTACH database within transaction. (Невозможно присоединить базу данных с использованием ATTACH в рамках транзакции.) |
| 2106 | Database '%s' is already in use. (База данных «%s» уже используется.) |
| 2108 | Attached databases must use the same text encoding as main database. (Присоединенные базы данных должны использовать ту же кодировку текста, что и главная база данных.) |
| 2200 | Недостаточно памяти. |
| 2201 | Unable to open database. (Невозможно открыть базу данных.) |
| 2202 | Cannot DETACH database within transaction. (Невозможно отключить базу данных с использованием DETACH в рамках транзакции.) |
| 2203 | Cannot detach database: '%s' (Невозможно отключить базу данных: «%s») |
| 2204 | Database '%s' is locked. (База данных «%s» заблокирована.) |
| 2205 | Не удалось приобрести блокировку считывания для базы данных. |
| 2206 | [столбец\|столбцы] "%s"[,"%s"] не [уникальны\|не] уникален. |
| 2207 | Malformed database schema. (Неправильный формат схемы базы данных.) |
| 2208 | Unsupported file format. (Неподдерживаемый формат файла.) |
| 2209 | Unrecognized token: '%s' (Неопознанная метка: «%s».) |
| 2300 | Could not convert text value to numeric value. (Невозможно преобразовать текстовое значение в числовое.) |
| 2301 | Could not convert string value to date. (Невозможно преобразовать строковое значение в дату.) |
| 2302 | Could not convert floating point value to integer without loss of data. (Невозможно преобразовать значение с плавающей запятой в целое число без потери данных.) |
| 2303 | Cannot rollback transaction - SQL statements in progress. (Невозможно выполнить откат транзакции, так как выполняются инструкции SQL.) |
| 2304 | Cannot commit transaction - SQL statements in progress. (Невозможно выполнить подтверждение транзакции, так как выполняются инструкции SQL.) |
| 2305 | Database table is locked: '%s' (Таблица базы данных заблокирована: «%s».) |
| 2306 | Read-only table. (Таблица доступна только для чтения.) |
| 2307 | String or blob too big. (Слишком большой размер строки или большого двоичного объекта.) |
| 2309 | Cannot open indexed column for writing. (Невозможно открыть проиндексированный столбец для записи.) |
| 2400 | Cannot open value of type %s. (Невозможно открыть значение типа %s.) |
| 2401 | No such rowid: %s< (Идентификатор строки не существует: %s<.) |
| 2402 | Object name reserved for internal use: '%s' (Имя объекта зарезервировано для внутреннего использования: «%s».) |
| 2403 | View '%s' may not be altered. (Невозможно изменить вид «%s».) |
| 2404 | Default value of column '%s' is not constant. (Значение по умолчанию столбца «%s» не является константой.) |
| 2405 | Not authorized to use function '%s' (Нет разрешения на использование функции «%s».) |
| 2406 | Misuse of aggregate function '%s' (Недопустимое использование агрегатной функции «%s».) |
| 2407 | Misuse of aggregate: '%s' (Недопустимое использование агрегата: «%s».) |
| 2408 | No such database: '%s' (База данных не существует: «%s».) |
| 2409 | Table '%s' has no column named '%s' (В таблице «%s» нет столбца с именем «%s».) |
| 2501 | No such module: '%s' (Модуль не существует: «%s».) |
| 2508 | No such savepoint: '%s' (Точка сохранения не существует: «%s».) |
| 2510 | Cannot rollback - no transaction is active. Невозможно выполнить откат, так как нет активных транзакций.) |
| 2511 | Cannot commit - no transaction is active. (Невозможно выполнить подтверждение, так как нет активных транзакций.) |

# Глава 68. Язык AGAL (Adobe Graphics Assembly Language)

Язык AGAL (Adobe Graphics Assembly Language) — это язык программирования шейдеров для определения вершинных и фрагментных программ визуализации. Программы AGAL необходимо загружать в контекст визуализации в формате двоичного байт-кода, описанном в данном документе.

## Формат байт-кода AGAL

Для байт-кода AGAL должен использоваться формат Endian.LITTLE_ENDIAN.

### Заголовок байт-кода

Байт-код AGAL должен начинаться с 7-байтового заголовка:

```
A0 01000000 A1 00 -- for a vertex program
A0 01000000 A1 01 -- for a fragment program
```

| Смещение (в байтах) | Размер (в байтах) | Имя | Описание |
|---|---|---|---|
| 0 | 1 | magic | должен быть 0xa0 |
| 1 | 4 | version | должен быть 1 |
| 5 | 1 | shader type ID | должен быть 0xa1 |
| 6 | 1 | shader type | 0 для вершинной программы; 1 для фрагментной программы |

### Метки

Сразу после заголовка следует любое количество меток. Размер каждой метки составляет 192 бита (24 байта), они всегда имеют формат:

```
[opcode][destination][source1][source2 or sampler]
```

Не для каждого кода операции используются все эти поля. Неиспользованные поля должны задаваться как 0.

### Коды операций

Поле [opcode] имеет размер 32 бита и может принимать одно из следующих значений.

| Имя | Код операции | Операция | Описание |
|---|---|---|---|
| mov | 0x00 | move | покомпонентно перемещает данные из source1 в destination. |
| add | 0x01 | add | destination = source1 + source2, покомпонентно |
| sub | 0x02 | subtract | destination = source1 - source2, покомпонентно |
| mul | 0x03 | multiply | destination = source1 * source2, покомпонентно |

| Имя | Код операции | Операция | Описание |
|---|---|---|---|
| div | 0x04 | divide | destination = source1 / source2, покомпонентно |
| rcp | 0x05 | reciprocal | destination = 1/source1, покомпонентно |
| min | 0x06 | minimum | destination = minimum(source1,source2), покомпонентно |
| max | 0x07 | maximum | destination = maximum(source1,source2), покомпонентно |
| frc | 0x08 | fractional | destination = source1 - (float)floor(source1), покомпонентно |
| sqt | 0x09 | square root | destination = sqrt(source1), покомпонентно |
| rsq | 0x0a | reciprocal root | destination = 1/sqrt(source1), покомпонентно |
| pow | 0x0b | power | destination = pow(source1,source2), покомпонентно |
| log | 0x0c | logarithm | destination = log_2(source1), покомпонентно |
| exp | 0x0d | exponential | destination = 2^source1, покомпонентно |
| nrm | 0x0e | normalize | destination = normalize(source1), покомпонентно (дает только трехкомпонентный результат, целевой элемент должен быть замаскирован как .xyz или меньше) |
| sin | 0x0f | sine | destination = sin(source1), покомпонентно |
| cos | 0x10 | cosine | destination = cos(source1), покомпонентно |
| crs | 0x11 | cross product | destination.x = source1.y * source2.z - source1.z * source2.y<br>destination.y = source1.z * source2.x - source1.x * source2.z<br>destination.z = source1.x * source2.y - source1.y * source2.x<br>(дает только трехкомпонентный результат, целевой элемент должен быть замаскирован как .xyz или меньше) |
| dp3 | 0x12 | dot product | destination = source1.x*source2.x + source1.y*source2.y + source1.z*source2.z |
| dp4 | 0x13 | dot product | destination = source1.x*source2.x + source1.y*source2.y + source1.z*source2.z + source1.w*source2.w |
| abs | 0x14 | absolute | destination = abs(source1), покомпонентно |
| neg | 0x15 | negate | destination = -source1, покомпонентно |
| sat | 0x16 | saturate | destination = maximum(minimum(source1,1),0), покомпонентно |
| m33 | 0x17 | multiply matrix 3x3 | destination.x = (source1.x * source2[0].x) + (source1.y * source2[0].y) + (source1.z * source2[0].z)<br>destination.y = (source1.x * source2[1].x) + (source1.y * source2[1].y) + (source1.z * source2[1].z)<br>destination.z = (source1.x * source2[2].x) + (source1.y * source2[2].y) + (source1.z * source2[2].z)<br>(дает только трехкомпонентный результат, целевой элемент должен быть замаскирован как .xyz или меньше) |

| Имя | Код операции | Операция | Описание |
|---|---|---|---|
| m44 | 0x18 | multiply matrix 4x4 | destination.x = (source1.x * source2[0].x) + (source1.y * source2[0].y) + (source1.z * source2[0].z) + (source1.w * source2[0].w)<br>destination.y = (source1.x * source2[1].x) + (source1.y * source2[1].y) + (source1.z * source2[1].z) + (source1.w * source2[1].w)<br>destination.z = (source1.x * source2[2].x) + (source1.y * source2[2].y) + (source1.z * source2[2].z) + (source1.w * source2[2].w)<br>destination.w = (source1.x * source2[3].x) + (source1.y * source2[3].y) + (source1.z * source2[3].z) + (source1.w * source2[3].w) |
| m34 | 0x19 | multiply matrix 3x4 | destination.x = (source1.x * source2[0].x) + (source1.y * source2[0].y) + (source1.z * source2[0].z) + (source1.w * source2[0].w)<br>destination.y = (source1.x * source2[1].x) + (source1.y * source2[1].y) + (source1.z * source2[1].z) + (source1.w * source2[1].w)<br>destination.z = (source1.x * source2[2].x) + (source1.y * source2[2].y) + (source1.z * source2[2].z) + (source1.w * source2[2].w)<br>(дает только трехкомпонентный результат, целевой элемент должен быть замаскирован как .xyz или меньше) |
| kil | 0x27 | kill/discard (только для фрагментного шейдера) | Если единственный скалярный исходный компонент меньше нуля, фрагмент игнорируется и не отрисовывается в буфере кадра. (Для целевого реестра должны быть заданы все нули) |
| tex | 0x28 | texture sample (только для фрагментного шейдера) | destination выравнивает нагрузку от текстуры source2 с координатами source1. В этом случае source2 должен быть в формате образца. |
| sge | 0x29 | set-if-greater-equal | destination = source1 >= source2 ? 1 : 0, покомпонентно |
| slt | 0x2a | set-if-less-than | destination = source1 < source2 ? 1 : 0, покомпонентно |
| seq | 0x2c | set-if-equal | destination = source1 == source2 ? 1 : 0, покомпонентно |
| sne | 0x2d | set-if-not-equal | destination = source1 != source2 ? 1 : 0, покомпонентно |

В AGAL2 добавлены следующие коды операций:

| Имя | Код операции | Операция | Описание |
|---|---|---|---|
| ddx | 0x1a | частная производная на оси X | Загрузить частную производную на оси X для source1 в место назначения. |
| ddy | 1x0b | частная производная на оси Y | Загрузить частную производную на оси Y для source1 в место назначения. |
| ife | 0x1c | если равно | Перейти, если source1 равно source2. |
| ine | 0x1d | если не равно | Перейти, если source1 не равно source2. |
| ifg | 1x0e | если больше | Перейти, если source1 больше или равно source2. |

| Имя | Код операции | Операция | Описание |
|---|---|---|---|
| ifl | 1x0f | если меньше | Перейти, если source1 меньше, чем source2. |
| els | 0x20 | else | Блок else |
| eif | 0x21 | Endif | Закрыть блок if или else. |

**Формат целевого поля**

Поле [destination] имеет размер 32 бита:

```
31.............................0
----TTTT----MMMMNNNNNNNNNNNNNNNN
```

T = тип регистра (4 бита)

M = маска записи (4 бита)

N = номер регистра (16 бит)

- = не определен, должен быть 0

**Формат исходного поля**

Поле [source] имеет размер 64 бита:

```
63..............................................................0
D-------------QQ----IIII----TTTTSSSSSSSSOOOOOOOONNNNNNNNNNNNNNNN
```

D = прямой=0/непрямой=1, для прямого Q и I игнорируются, 1 бит

Q = выбор компонента регистра индекса (2 бита)

I = тип регистра индекса (4 бита)

T = тип регистра (4 бита)

S = преобразование (8 бит, 2 бита на компонент)

O = косвенное смещение (8 бит)

N = номер регистра (16 бит)

- = не определен, должен быть 0

**Формат поля образца**

Второе исходное поле для кода операции с текстурами должно иметь формат [sampler], который имеет размер 64 бита.

```
63..............................................................0
FFFFMMMMWWWWSSSSDDDD--------TTTT--------BBBBBBBBNNNNNNNNNNNNNNNN
```

N = номер регистра образца (16 бит)

B = смещение уровня детализации текстуры, целое число со знаком, масштаб x8. Используемое значение с плавающей запятой: b/8,0 (8 бит)

T = тип регистра, должен быть 5, образец (4 бита)

F = фильтр (0=ближайшие,1=линейные) (4 бита)

M = множественное отображение (0=выкл.,1=ближайшие, 2=линейные)

W = перенос (0=зажим,1=повтор)

S = биты специальных флагов (должно быть 0)

D = размерность (0=2D, 1=куб)

**Реестры программ**

Число используемых реестров зависит от используемого профиля Context3D. Число реестров и их использование определяются в следующей таблице:

| **Имя**<br><br>**AGAL2**<br><br>**AGAL3**<br><br>**Применение** | **Значение** | **AGAL** | |
|---|---|---|---|
| Число в фрагментной программе<br>Число в вершинной программе<br>Число в фрагментной программе<br>Число в вершинной программе | | Число в фрагментной программе | Число в вершинной программе |
| Поддержка профилей Context3D<br>Стандарт<br>Расширенный стандарт | | Ниже стандарта | |
| Версия SWF<br>25<br>28 и больше | | Ниже 25 | |

| **Имя**<br><br>**AGAL2**<br><br>**AGAL3**<br><br>**Применение** | **Значение** | **AGAL** | |
|---|---|---|---|
| Атрибут<br>Н/Д<br>8<br>Н/Д<br>16<br>Ввод вершинного шейдера; считывается из буфера вершин, заданного с помощью Context3D.setVertexBufferAt(). | 0 | Н/Д | 8 |
| Константа<br>64<br>250<br>200<br>250<br>Ввод шейдера; задается с помощью семейства функций Context3D.setProgramConstants(). | 1 | 28 | 128 |

| **Имя**<br><br>**AGAL2**<br><br>**AGAL3**<br><br>**Применение** | **Значение** | **AGAL** | |
|---|---|---|---|
| Временный<br>26<br>26<br>26<br>26<br>Временный реестр для вычислений; недоступен за пределами программы. | 2 | 8 | 8 |
| Вывод<br>1<br>1<br>1<br>1<br>Вывод шейдера: в вершинной программе выводом является положение пространстве обрезки, а во фрагментной — цвет. | 3 | 1 | 1 |

| **Имя**<br><br>**AGAL2**<br><br>**AGAL3**<br><br>**Применение** | **Значение** | **AGAL** | |
|---|---|---|---|
| Переменная<br>10<br>10<br>10<br>10<br>Передача интерполированных данных между вершинными и фрагментными шейдерами. Переменные регистры из вершинной программы применяются в качестве ввода в фрагментную программу. Значения интерполируются в соответствии с расстоянием от вершин треугольников. | 4 | 8 | 8 |

| **Имя**<br><br>**AGAL2**<br><br>**AGAL3**<br><br>**Применение** | **Значение** | **AGAL** | |
|---|---|---|---|
| Образец<br>16<br>Н/Д<br>16<br>Н/Д<br>Ввод фрагментного шейдера; считывается из текстуры, заданной с помощью Context3D.setTextureAt(). | 5 | 8 | Н/Д |
| Реестр фрагментов<br>1<br>Н/Д<br>1<br>Н/Д<br>Доступен только для записи и служит для перезаписи значения z (глубины), записанного в вершинном шейдере. | 6 | Н/Д | Н/Д |

| **Имя**<br><br>**AGAL2**<br><br>**AGAL3**<br><br>**Применение** | **Значение** | **AGAL** |
|---|---|---|
| Метки<br>1024<br>2048 | | 200 |

Последнюю версию AGAL Mini Assembler можно найти здесь.