Разработка приложений ADOBE® AIR®

Разработка приложений Adobe® AIR®

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Последнее обновление 13.5.2010.

Содержание

Глава 1. Знакомство с Adobe AIR

Adobe® AIR® — это поддерживающая множество платформ среда выполнения, позволяющая максимально эффективно использовать возможности разработки для сборки и развертывания мультимедийных интернет-приложений (RIA) на настольных ПК. Для создания приложений AIR можно использовать Adobe® Flash® CS3 Professional, Adobe® Flash® CS4 Professional, Adobe® Flex™, Adobe® ActionScript® 3.0, HTML, JavaScript® и Ajax.

Дополнительные сведения о том, как начать работать с Adobe AIR и как использовать этот продукт, см. на сайте Adobe AIR Developer Connection (http://www.adobe.com/devnet/air/).

AIR позволяет работать в знакомых средах разработки, используя наиболее удобные инструменты и методы. Благодаря поддержке Flash, Flex, HTML, JavaScript и Ajax можно создавать оптимальные условия работы, соответствующие конкретным потребностям.

В частности, при разработке приложений можно пользоваться одной или несколькими из приведенных ниже технологий:

- Flash / Flex / ActionScript
- HTML / JavaScript / CSS / Ajax
- С любым приложением совместим формат PDF

В результате приложения AIR может:

- иметь в своей основе Flash или Flex — ключевое содержимое таких приложений содержится в формате Flash/Flex (SWF);
- основываться на Flash или Flex с HTML или PDF — их ключевое содержимое хранится в формате Flash/Flex (SWF) с добавлением содержимого в формате HTML (HTML, JS, CSS) или PDF;
- иметь в своей основе HTML — ключевым содержимым таких приложений будет HTML, JS и CSS;
- основываться на HTML с добавлением Flash/Flex или PDF — в этом случае ключевым содержимым приложения будет HTML с добавлением форматов Flash/Flex (SWF) или PDF.

С точки зрения пользователя, приложения AIR выглядят точно так же, как приложения для настольных компьютеров. Среда выполнения устанавливается на компьютер пользователя только один раз, после чего приложения AIR устанавливаются и используются, как любые другие программы.

Среда выполнения предлагает надежную платформу, совместимую с разными операционными системами, и инфраструктуру для разработки приложений. Таким образом, благодаря проверенной функциональности и взаимодействию с разными настольными компьютерами, она избавляет вас от необходимости тестирования приложений в множестве разных обозревателей. Вы разрабатываете приложение не под конкретную операционную систему, а для среды выполнения, что имеет ряд очевидных преимуществ:

- Приложения, разработанные для AIR, функционируют во многих операционных системах без дополнительных доработок с вашей стороны. Среда выполнения обеспечивает предсказуемое и надежное представление содержимого и взаимодействие программы с пользователем во всех операционных системах, которые поддерживает AIR.
- Можно ускорить создание приложений благодаря использованию существующих веб-технологий и шаблонов проектирования. Можно переносить веб-приложения на настольные компьютеры, не изучая традиционные технологии разработки для настольных систем или их сложные собственные коды.
- Такой способ разработки проще, чем разработка на низкоуровневых языках типа C и C++. Вам не придется сталкиваться со сложными низкоуровневыми API-интерфейсами каждой из операционных систем.

При разработке приложений для AIR вам доступен богатый выбор инфраструктур и API-интерфейсов:

- API-интерфейсы AIR, предоставляемые средой выполнения и инфраструктурой AIR;
- API-интерфейсы ActionScript, используемые в SWF-файлах и инфраструктурах Flex (и других библиотеках и инфраструктурах на основе ActionScript);
- языки HTML, CSS и JavaScript;
- большинство инфраструктур Ajax

AIR кардинальным образом меняет процесс создания, развертывания и использования приложений. Вам доступно больше творческих возможностей для совершенствования приложений на основе Flash, Flex, HTML и Ajax для настольных компьютеров, и при этом не нужно изучать традиционные технологии разработки для настольных систем.

Что нового в AIR 1.1?

В Adobe AIR 1.1 появились следующие новые возможности:

- Установочные и прочие диалоговые окна были переведены на следующие языки.
  - Португальский (Бразилия)
  - Китайский (традиционный и упрощенный)
  - Французский
  - Немецкий
  - Итальянский
  - Японский
  - Корейский
  - Русский
  - Французский
  - Испанский
- Поддержка при создании приложений для работы в различных странах, включая ввод с клавиатуры для языков с двухбайтовыми кодировками. См. раздел ««Локализация приложений AIR» на странице 144».
  - Поддержка локализации атрибутов name и description в файле дескриптора приложения.
  - Поддержка локализации сообщений об ошибках, например SQLError.detailID и SQLError.detailArguments, в базе данных SQLite.
  - Добавление свойства Capabilities.languages для получения массива предпочтительных языков интерфейса пользователя в соответствии с установками операционной системы.
  - Метки кнопок HTML и меню по умолчанию, например контекстные меню и панель меню Mac, были локализованы для всех поддерживаемых языков.
- Поддержка миграции сертификата из самоподписывающего приложения в приложение с сертификатом, выданным соответствующими органами.
- Поддержка Microsoft Windows XP для карманных ПК и поддержка 64-разрядных версий Windows Vista® Home Premium, Business, Ultimate или Enterprise.

- Добавление прикладного интерфейса программирования `File.spaceAvailable` для освобождения места на диске.
- Добавление свойства `NativeWindow.supportsTransparency` для определения того, можно ли отображать окно как прозрачное в используемой операционной системе.

Дополнительные сведения об Adobe AIR версии 1.1 см. в примечаниях к выпуску Adobe AIR 1.1 по адресу (http://www.adobe.com/go/learn_air_relnotes_ru).

Что нового в AIR 1.5?

В Adobe AIR 1.5 появились следующие новые возможности:

- Поддержка новых функций Adobe Flash Player 10.
  - Настраиваемые фильтры и эффекты
  - Улучшенный прикладной интерфейс программирования для отображения
  - Динамическое создание звуков
  - Векторный тип данных
  - Улучшенные прикладные интерфейсы программирования для загрузки и выгрузки файлов
  - Протокол RTMFP
  - Трехмерные эффекты
  - Расширенная поддержка текста
  - Управление цветом
  - Текстовый процессор
  - Создание динамических потоков
  - Аудиокодек Speex

  Дополнительные сведения об этих возможностях см. по адресу http://www.adobe.com/products/flashplayer/features/.

  Дополнительные сведения об использовании новых прикладных интерфейсов программирования ActionScript 3.0 в Adobe Flash Player 10 см. в книге «Руководство разработчика Adobe ActionScript 3.0» (http://www.adobe.com/go/learn_flex_programmingAS3_ru), а также в справочнике ActionScript® 3.0 для Adobe® Flash® Platform (http://www.adobe.com/go/learn_flex3_aslr_ru).
- Дополнительные языки поддерживаются в программе установки AIR 1.5 и других диалоговых окнах, открывающихся в среде выполнения:
  - Чешский
  - Голландский
  - Шведский
  - Турецкий
  - Польский
- Шифрование базы данных.

Можно зашифровать файлы базы данных в AIR 1.5. Все содержимое базы данных, включая метаданные, может быть зашифровано таким образом, что данные не смогут быть прочитаны вне приложения AIR, зашифровавшего их. Эта возможность позволит разработчику зашифровывать, расшифровывать и перешифровывать файлы данных. См. раздел Зашифрованное локальное хранилище (для разработчиков ActionScript) или Зашифрованное локальное хранилище (для разработчиков HTML).

- Версия пакета WebKit, используемая Adobe AIR, была обновлена и теперь включает поддержку интерпретатора SquirrelFish JavaScript.
- Новый прикладной интерфейс программирования проверки подписи XML, который может использоваться для проверки целостности и идентификации подписчика для данных или информации. Дополнительные сведения см. в разделе Проверка подписи XML в AIR (для разработчиков ActionScript) или Проверка подписи XML в AIR (для разработчиков HTML).

Дополнительные сведения об Adobe AIR версии 1.5 см. в сведениях о выпуске Adobe AIR 1.5 по адресу (http://www.adobe.com/go/learn_air_relnotes_ru).

Что нового в AIR 1.5.1?

В Adobe AIR 1.5.1 появились следующие новые возможности:

Последняя версия надстройки Flash Player

AIR 1.5.1 содержит обновленную версию надстройки Flash Player (10.0.22), которая используется при просмотре SWF-файлов внутри HTML. Дополнительные сведения см. по адресу http://www.adobe.com/support/documentation/en/flashplayer/releasenotes.html.

Новые прикладные программные интерфейсы

`InvokeEvent.reason`

Это новое свойство события InvokeEvent указывает, сам ли пользователь запустил приложение или оно было автоматически загружено при его входе в систему. Класс InvokeEventReason (в пакете flash.desktop) определяет два возможных строчных значения для свойства InvokeEvent.reason. InvokeEventReason.LOGIN определяет вариант при входе в систему; InvokeEventReason.STANDARD определяет стандартный случай.

`Capabilities.cpuArchitecture`

В этом новом свойстве возвращается информация об архитектуре процессора на компьютере, в виде строки (например, «PowerPC» или «x86»).

Если необходимо реализовать преимущества этих новых прикладных программных интерфейсов AIR 1.5.1, обновите свой файл дескриптора приложения, разрешив использование пространства имен 1.5.1:

```
xmlns="http://ns.adobe.com/air/application/1.5.1"
```

Если использование новых прикладных программных интерфейсов не требуется, то обновлять дескриптор приложения не обязательно. Приложение получит возможность выполняться в среде AIR 1.5.1, если пользователь обновит версию среды выполнения, установленной в его системе.

Дополнительные сведения об Adobe AIR версии 1.5.1 см. в сведениях о выпуске Adobe AIR 1.5.1 по адресу (http://www.adobe.com/go/learn_air_relnotes_ru).

Что нового в AIR 1.5.2?

В Adobe AIR 1.5.2 появились следующие новые возможности:

Новые прикладные программные интерфейсы

`Capabilities.supports32BitProcesses` и `Capabilities.supports64BitProcesses`

Эти свойства указывают, будет ли система поддерживать 64-разрядные или 32-разрядные процессы.

`LocalConnection.isPerUser`

Это свойство указывает, будет ли область действия объектов LocalConnection ограничена текущим пользователем (TRUE), или они будут глобально доступны всем пользователям на данном компьютере (FALSE). Это свойство влияет только на содержимое, выполняющееся в Mac OS; другие платформы игнорируют данный параметр. Например, локальные подключения в Windows и Linux всегда выполняются на уровне пользователя. В предыдущих версиях программы все объекты LocalConnection на Mac OS имели глобальную область действия. Из соображений безопасности всегда устанавливайте для этого свойства значение TRUE, за исключением тех случаев, когда необходимо сохранить совместимость с предыдущими версиями. В будущих версиях это свойство скорее всего будет иметь значение по умолчанию TRUE.

`System.disposeXML()`

Это статический метод делает объект ActionScript XML мгновенно доступным для сборки мусора. Данный метод удаляет родительские и дочерние связи между всеми узлами для указанного объекта XML. Этот метод использует один параметр: объект XML, который нужно сделать доступным для сбора мусора. Используйте этот метод, чтобы обеспечить эффективность распределения памяти, связанной с объектами XML.

Обновление файла дескриптора приложения

Обновите файл дескриптора своего приложения, обеспечив поддержку пространства имен версии 1.5.2 для использования новых возможностей и интерфейсов прикладного программирования AIR 1.5.2. Для обновления пространства имен измените атрибут xmlns, установив для него:

```
xmlns="http://ns.adobe.com/air/application/1.5.2"
```

Что нового в версии AIR 2 Beta

Полные сведения о версии AIR 2 beta см. по адресу Adobe Labs AIR 2 Beta.

Глава 2. Установка Adobe AIR

С помощью Adobe® AIR® можно выполнять приложения AIR на настольном компьютере. Его можно установить следующими способами:

- установить только среду выполнения (не устанавливать приложение AIR)
- установить приложение AIR впервые (вам будет предложено установить среду выполнения)
- установить в качестве среды разработки AIR комплект AIR SDK, Adobe® Flex® Builder™ или комплект Adobe Flex® SDK (включающий инструменты разработки командной строки AIR)

  Среда выполнения устанавливается на компьютер один раз.

Системные требования для установки Adobe AIR и выполнения приложений AIR подробно описаны здесь: Adobe AIR: системные требования (http://www.adobe.com/products/air/systemreqs/).

Файлы журналов создаются установщиками среды выполнения и приложения AIR при установке, обновлении или удалении приложений AIR или среды выполнения AIR. Журналы установок позволяют определить причины проблем при установке или обновлении. См. раздел ««Журналы установки» на странице 96».

Установка Adobe AIR

Используйте следующие инструкции, чтобы загрузить и установить версии Adobe AIR для Windows, Mac OS X и Linux.

Для обновления среды выполнения на компьютере необходимы права администратора.

Установка среды выполнения на компьютер Windows

1. Загрузите установочный файл.
2. Дважды щелкните его.
3. Следуйте инструкциям в окне установки для ее выполнения.

Установка среды выполнения на компьютер Mac

1. Загрузите установочный файл.
2. Дважды щелкните его.
3. Следуйте инструкциям в окне установки для ее выполнения.
4. Если установщик отображает окно идентификации, введите свое имя пользователя в системе Mac OS и пароль.

Установка среды выполнения на компьютере с Linux

1. Загрузите установочный файл.
2. Установите права на доступ к файлам таким образом, чтобы могла выполниться программа установки приложения:

Из командной строки можно установить права на доступ к файлу с помощью команды `chmod +x installer.bin`. Некоторые версии Linux позволяют установить права доступа к файлам с помощью диалогового окна «Свойства» (Properties), открываемого из контекстного меню.

3 Запустите программу установки из командной строки или двойным щелчком выполняемого установочного файла.

4 Следуйте инструкциям в окне установки для ее выполнения.

Adobe AIR устанавливается либо как пакет rpm, либо как пакет dpkg, с именем: `adobeairv.n` и `adobecerts`. Для установки необходимо, чтобы был запущен X-сервер. AIR регистрирует тип MIME: `application/vnd.adobe.air-application-installer-package+zip`.

Удаление Adobe AIR

Установленную на компьютер среду выполнения можно удалить. Ниже описано, как это сделать.

Удаление среды выполнения с компьютера под управлением Windows

1 В меню «Пуск» выберите «Настройки» > «Панель управления».

2 Выберите пункт «Установка и удаление программ».

3 Выберите «Adobe AIR», чтобы удалить среду выполнения.

4 Нажмите кнопку «Изменить/Удалить».

Удаление среды выполнения с компьютера под управлением Mac OS

- Дважды щелкните «Adobe AIR Uninstaller» в папке /Applications/Utilities.

Удаление среды выполнения с компьютера под управлением Linux

Выполните одно из следующих действий:

- Выберите команду «Adobe AIR Uninstaller» из меню «Приложения» (Applications).
- Запустите двоичный файл программы установки AIR с параметром `-uninstall`
- Удалите пакеты AIR (`adobeairv.n` и `adobecerts`) с помощью диспетчера пакетов.

Установка и выполнение образцов приложений AIR

Вы можете ознакомиться с образцами некоторых приложений AIR и их характеристиками. Ниже описано, как их загрузить и установить:

1 Загрузите и запустите образцы приложений AIR. Доступны как скомпилированные файлы, так и исходный код.

2 Для загрузки и запуска образца приложения щелкните по кнопке «Установить» рядом с ним. Вам будет предложено установить и запустить приложение.

3 Если вы решите загрузить образцы приложений, но запустить их позднее, выберите только ссылки для загрузки. Приложения AIR можно запустить в любое время.

   - В Windows дважды щелкните значок приложения на рабочем столе или выберите его в меню «Пуск».

- В Mac OS дважды щелкните по значку приложения, по умолчанию установленного в папку Applications в каталоге пользователя (например, Macintosh HD/Пользователи/Иван/Applications/).
- В Linux дважды щелкните значок приложения на рабочем столе или выберите его из меню «Приложения» (Applications). Приложения AIR устанавливаются в собственную папку в каталоге `/opt`.

Примечание. *Проверьте, не обновлялись ли эти инструкции, в примечаниях к выпуску: http://www.adobe.com/go/learn_air_relnotes_ru.*

Обновления Adobe AIR

Время от времени компания Adobe выпускает обновления Adobe AIR, содержащие новые функции или исправления незначительных проблем. С помощью функции автоматического уведомления и обновления компания Adobe автоматически уведомляет пользователей о доступности обновленной версии Adobe AIR.

Обновления Adobe AIR обеспечивают правильную работу Adobe AIR и могут содержать важные изменения системы защиты. Компания Adobe рекомендует выполнять обновление до последней версии Adobe AIR при появлении новой версии, особенно если упоминается обновление системы защиты.

По умолчанию при запуске приложения AIR среда выполнения проверяет доступность обновления. Она выполняет эту проверку, если прошло более двух недель с момента последней проверки наличия обновлений. Если обновление доступно, среда AIR загружает обновление в фоновом режиме.

Пользователи могут отключить возможность автоматического обновления с помощью приложения AIR SettingsManager. Приложение AIR SettingsManager можно загрузить с веб-страницы http://airdownload.adobe.com/air/applications/SettingsManager/SettingsManager.air.

Стандартный процесс установки Adobe AIR включает соединение с веб-сайтом http://airinstall.adobe.com для отправки основной информации о среде установки, такой как версия и язык операционной системы. Эта информация передается только один раз при каждой установке и позволяет компании Adobe подтвердить успешное выполнение установки. Персональные данные не собираются и не передаются.

Глава 3. Функции, характерные для Adobe AIR

Adobe® AIR® включает функции, недоступные в SWF-содержимом, выполняемом в проигрывателе Adobe® Flash® Player.

HTML-разработчики в среде AIR

При разработке HTML-приложений в среде AIR полный список прикладных интерфейсов программирования, доступных в JavaScript с помощью файла AIRAliases.js, см. в документе Справочное руководство для HTML-разработчиков по прикладным интерфейсам программирования Adobe AIR.

Классы ActionScript 3.0, характерные для AIR

Следующие выполняемые классы характерны для Adobe AIR. Они недоступны при воспроизведении SWF-содержимого в обозревателе.

Класс	Пакет
AAAARecord	flash.net.dns
ApplicationUpdater	air.update
ApplicationUpdaterUI	air.update
ARecord	flash.net.dns
BrowserInvokeEvent	flash.events
CertificateStatus	flash.security
CompressionAlgorithm	flash.utils
DatagramSocket	flash.net
DatagramSocketDataEvent	flash.events
DNSResolver	flash.net.dns
DNSResolverEvent	flash.events
DockIcon	flash.desktop
DownloadErrorEvent	air.update.events
DRMAuthenticateEvent	flash.events
DRMManagerError	flash.errors
EncryptedLocalStore	flash.data
File	flash.filesystem
FileListEvent	flash.events

Класс	Пакет
FileMode	flash.filesystem
FileStream	flash.filesystem
FocusDirection	flash.display
Geolocation	flash.sensors
GeolocationEvent	flash.events
HTMLHistoryItem	flash.html
HTMLHost	flash.html
HTMLLoader	flash.html
HTMLPDFCapability	flash.html
HTMLUncaughtScriptExceptionEvent	flash.events
HTMLWindowCreateOptions	flash.html
Icon	flash.desktop
InteractiveIcon	flash.desktop
InterfaceAddress	flash.net
InvokeEvent	flash.events
InvokeEventReason	flash.desktop
MXRecord	flash.net.dns
NativeApplication	flash.desktop
NativeDragActions	flash.desktop
NativeDragEvent	flash.events
NativeDragManager	flash.desktop
NativeDragOptions	flash.desktop
NativeMenu	flash.display
NativeMenuItem	flash.display
NativeProcess	flash.desktop
NativeProcessExitEvent	flash.events
NativeProcessStartupInfo	flash.desktop
NativeWindow	flash.display
NativeWindowBoundsEvent	flash.events
NativeWindowDisplayState	flash.display
NativeWindowDisplayStateEvent	flash.events
NativeWindowInitOptions	flash.display
NativeWindowResize	flash.display
NativeWindowSystemChrome	flash.display

Класс	Пакет
NativeWindowType	flash.display
NetworkInfo	flash.net
NetworkInterface	flash.net
NotificationType	flash.desktop
OutputProgressEvent	flash.events
PaperSize	flash.printing
PrintMethod	flash.printing
PrintUIOptions	flash.printing
PTRRecord	flash.net.dns
ReferencesValidationSetting	flash.security
ResourceRecord	flash.net.dns
RevocationCheckSettings	flash.security
Screen	flash.display
ScreenMouseEvent	flash.events
SecureSocket	flash.net
SecureSocketMonitor	air.net
SecureSocketConnectEvent	flash.net
ServiceMonitor	air.net
SignatureStatus	flash.security
SignerTrustSettings	flash.security
SocketMonitor	air.net
SQLCollationType	flash.data
SQLColumnNameStyle	flash.data
SQLColumnSchema	flash.data
SQLConnection	flash.data
SQLError	flash.errors
SQLErrorEvent	flash.events
SQLErrorOperation	flash.errors
SQLEvent	flash.events
SQLIndexSchema	flash.data
SQLMode	flash.data
SQLResult	flash.data
SQLSchema	flash.data
SQLSchemaResult	flash.data

Класс	Пакет
SQLStatement	flash.data
SQLTableSchema	flash.data
SQLTransactionLockType	flash.data
SQLTriggerSchema	flash.data
SQLUpdateEvent	flash.events
SQLViewSchema	flash.data
SRVRecord	flash.net.dns
StageAspectRatio	flash.display
StageOrientation	flash.display
StageOrientationEvent	flash.events
StatusFileUpdateErrorEvent	air.update.events
StatusFileUpdateEvent	air.update.events
StatusUpdateErrorEvent	air.update.events
StatusUpdateEvent	air.update.events
StorageVolume	flash.filesystem
StorageVolumeChangeEvent	flash.events
StorageVolumeInfo	flash.filesystem
SystemTrayIcon	flash.desktop
UpdateEvent	air.update.events
Updater	flash.desktop
URLFilePromise	air.desktop
URLMonitor	air.net
URLRequestDefaults	flash.net
XMLSignatureValidator	flash.security

Кроме того, существуют два предназначенных только для среды AIR интерфейса:

- flash.desktop.IFilePromise
- flash.security.IURIDereferencer

Классы проигрывателя Flash Player с функциями, характерными для Adobe AIR

Ниже перечислены классы, которые могут использоваться при воспроизведении в обозревателе SWF-содержимого и для которых AIR предлагает дополнительные методы и свойства:

Класс	Свойство или метод
Capabilities	`languages`
Clipboard	`supportsFilePromise`
ClipboardFormats	`BITMAP_FORMAT` FILE_LIST_FORMAT FILE_PROMISE_LIST_FORMAT URL_FORMAT
Event	`DISPLAYING` EXITING HTML_BOUNDS_CHANGE HTML_DOM_INITIALIZE HTML_RENDER LOCATION_CHANGE NETWORK_CHANGE USER_IDLE USER_PRESENT
FileReference	`uploadUnencoded()`

HTTPStatusEvent	`HTTP_RESPONSE_STATUS`
	`responseURL`
	`responseHeaders`

KeyboardEvent	`commandKey` `controlKey`
LoaderContext	`allowLoadBytesCodeExecution`
LoaderInfo	`parentSandboxBridge` `childSandboxBridge`
NetStream	`resetDRMVouchers()` `setDRMAuthenticationCredentials()`

Класс	Свойство или метод

PrintJob	`active`
	`copies`
	`firstPage`
	`isColor`
	`jobName`
	`lastPage`
	`maxPixelsPerInch`
	`paperArea`
	`printableArea`
	`printer`
	`printers`
	`supportsPageSetupDialog`
	maxPixelsPerInch

URLRequest	`followRedirects`
	`manageCookies`
	`shouldAuthenticate`
	`shouldCacheResponse`
	`userAgent`
	`userCache`
	`selectPaperSize()`
	`showPageSetupDialog()`
	`start2()`
	`terminate()`

PrintJobOptions	`resolution`
URLStream	`httpResponseStatus event`
Stage	`nativeWindow`
Security	`APPLICATION`

Большинство перечисленных новых свойств и методов доступно только для содержимого в изолированной программной среде AIR. Тем не менее новые члены классов URLRequest также доступны для воспроизведения содержимого в других изолированных программных средах.

Методы `ByteArray.compress()` и `ByteArray.uncompress()` имеют новый параметр `algorithm`, позволяющий выбрать метод сжатия deflate или zlib. Этот параметр доступен только для содержимого, воспроизводимого в Adobe AIR.

Компоненты Flex, характерные для Adobe AIR

Следующие компоненты Adobe® Flex™ MX доступны при разработке содержимого для Adobe AIR:

- FileEvent
- FileSystemComboBox
- FileSystemDataGrid
- FileSystemEnumerationMode
- FileSystemHistoryButton
- FileSystemList
- FileSystemSizeDisplayMode
- FileSystemTree
- FlexNativeMenu
- HTML
- Window
- WindowedApplication
- WindowedSystemManager

Помимо этого, Flex 4 включает следующие компоненты Spark среды AIR:

- Window
- WindowedApplication

Дополнительные сведения о компонентах AIR Flex см. в руководстве Использование компонентов Flex AIR.

Глава 4. Инструменты Adobe Flash Platform для разработки приложений AIR

Приложения AIR можно создавать с помощью следующих инструментов платформы Adobe Flash Platform.

Для разработчиков ActionScript 3.0 (Flash и Flex):

- Adobe Flash Professional CS3, CS4, CS5
- SDK Adobe Flex 3.x и 4.0
- Adobe Flex Builder 3.x
- Adobe Flash Builder 4.0

Для разработчиков HTML и Ajax:

- SDK Adobe AIR
- Adobe Dreamweaver CS3, CS4, CS5

Настройка Flash CS3 для Adobe AIR

Обновление Adobe® AIR® Update для Adobe® Flash® CS3 Professional расширяет среду разработки Adobe Flash элементами, которые позволяют создавать приложения AIR с помощью Adobe Flash. Оно позволяет создавать, тестировать и отлаживать файлы приложений AIR в Adobe Flash.

Adobe® Flash® CS4 Professional имеет встроенные средства для создания приложений AIR. Дополнительные сведения см. в разделе «Публикация для Adobe AIR» в руководстве по использованию Flash.

Примечание. *Обновление Adobe AIR для Flash CS3 поддерживает AIR 1.0, AIR 1.1 и Flash Player 9.x. Flash CS4 требуется для разработки приложения с помощью AIR 1.5 и Flash Player 10.*

Системные требования обновления Adobe AIR для Adobe Flash CS3

Чтобы использовать Flash CS3 для разработки и выполнения приложений AIR, необходимо установить следующее программное обеспечение.

- Flash CS3 Professional

  Если отсутствует копия Flash CS3 Professional, это программное обеспечение можно купить на веб-сайте Adobe по адресу: http://www.adobe.com/products/flash/

- Adobe AIR

  Сведения по установке Adobe AIR см. в разделе ««Установка Adobe AIR» на странице 6».

- Обновление Adobe AIR для Flash CS3

Установка обновления Adobe AIR 1.0 для Flash CS3

Прежде чем устанавливать обновление Adobe AIR для Flash CS3, выйдите из программы Adobe Flash и любого другого запущенного браузера.

- Загрузите обновление Adobe AIR 1.0 для Flash CS3 (http://www.adobe.com/support/flash/downloads.html#flashCS3)
- После того как обновление загружено, дважды щелкните файл исправления, чтобы установить его.

Установка обновления Adobe AIR 1.1 для Flash CS3

Чтобы установить это обновление, необходимо сначала установить исходное обновление Adobe AIR 1.0 для Adobe Flash CS3 Professional. Без установки предыдущего обновления данное обновление установить невозможно.

1. Найдите папку с именем AIK:

   Windows: C:/Program Files/Adobe/Adobe Flash CS3

   Mac OS X: жесткий диск Macintosh: Программы: Adobe Flash CS3
2. Удалите все файлы из папки AIK. Саму папку не удаляйте.
3. Загрузите соответствующую версию AIRSDKIntegrationKit:

   Windows: http://www.adobe.com/go/getaikwin_ru

   Mac OS X: http://www.adobe.com/go/getaikmac_ru
4. Распакуйте содержимое AIRSDKIntegrationKit в папку AIK.
5. Загрузите среду выполнения Adobe AIR 1.1 с веб-страницы http://get.adobe.com/air/.
6. Установите среду выполнения на компьютер.

Удаление обновления Adobe AIR для Flash CS3

Чтобы удалить обновление Adobe AIR для Flash CS3, выполните следующие действия.

1. Удалите следующую папку:

   (Windows) HD:\Program Files\Adobe\Adobe Flash CS3\AIK

   (Mac) HD:/Applications/Adobe Flash CS3/AIK
2. Перейдите в следующую папку:

   (Windows) HD:\Program Files\Adobe\Adobe Flash CS3\<язык>\First Run\Commands\

   (Mac) HD:/Applications/Adobe Flash CS3/First Run/Commands

   и удалите следующие файлы и папки:

   - папка AIR
   - AIR - Application and Installer Settings.jsfl
   - AIR - Create AIR File.jsfl
3. Удалите следующий файл:

   (Windows) HD:\Program Files\Adobe\Adobe Flash CS3\<язык>\Configuration\External Libraries\FLAir.dll

   (Mac) HD:/Applications/Adobe Flash CS3/Configuration/External Libraries/FLAir.bundle.

4 Удалите следующий файл:

(Windows) HD:\Program Files\Adobe\Adobe Flash CS3\<язык>\Configuration\Players\AdobeAIR1_0.xml

(Mac) HD:/Applications/Adobe Flash CS3/Configuration/Players/ AdobeAIR1_0.xml

5 Перейдите в следующую папку:

(Windows) HD:\Document and Settings\<имя_пользователя>\Local Settings\Application Data\Adobe\Flash CS3\<язык>\Configuration\Commands\

(Mac) HD:/Users/<имя_пользователя>/Library/Application Support/Adobe/Flash CS3/<язык>/Configuration/Commands/

и удалите следующие файлы и папки:

- папка AIR
- AIR - Application and Installer Settings.jsfl
- AIR - Create AIR File.jsfl

Примечание. *Если указанная папка не видна в Windows, установите флажок «Показать скрытые файлы и папки» в параметрах папок.*

Разработка приложений AIR в программе Flex Builder 3

В комплект Adobe® Flex™ Builder™ 3 входят инструменты для создания приложений Adobe® AIR®. В состав Flex Builder входят компоненты Flex для приложений Adobe AIR. Алгоритм разработки приложений AIR в программе Flex Builder аналогичен алгоритму разработки для большинства приложений Flex.

Создание проектов AIR в программе Flex Builder

Установите AIR and Flex Builder 3, если они не были установлены.

- Откройте Flex Builder 3.
- Выберите «Файл» > «Создать» > «Проект Flex».
- Введите имя проекта.
- Приложения AIR считаются типом приложений во Flex. Существует два типа. Интернет-приложение Flex, запускаемое в проигрывателе Adobe® Flash® Player. Настольное приложение AIR, запускаемое в среде Adobe AIR. В качестве типа приложения выберите настольное приложение.
- Выберите технологию сервера (если требуется) для использования с пользовательским приложением AIR. Если технология сервера не используется, нажмите «Отсутствует», а затем «Далее».
- Выберите каталог для хранения пользовательского приложения. По умолчанию таким каталогом является bin. Нажмите кнопку «Далее».
- Измените требуемым образом исходный путь и путь к библиотеке и нажмите «Готово» для создания проекта AIR.

Отладка приложений AIR в программе Flex Builder

Flex Builder предоставляет полную поддержку по отладке для приложений AIR. Дополнительную информацию по возможностям отладки во Flex Builder см. в справке Flex Builder.

1 Откройте исходный файл приложения (например, MXML-файл) во Flex Builder.

2 Нажмите кнопку «Отладка» на главной панели инструментов.

Также можно выбрать меню «Выполнить» >«Отладка».

Приложение запускается и выполняется в приложении ADL (AIR Debugger Launcher). Отладчик Flex Builder отслеживает любые точки прерывания или ошибки среды выполнения, после чего можно отладить это приложение так же, как и любое другое приложение Flex.

Также можно отладить приложение из командной строки с помощью инструмента ADL командной строки. Дополнительные сведения см. в разделе ««Использование средства AIR Debug Launcher (ADL)» на странице 51».

Упаковка приложений AIR в программе Flex Builder

Когда приложение завершено и подготовлено к выпуску (или проведению теста с рабочего стола), оно упаковывается в файл AIR. Процесс упаковки включает следующие шаги:

- выбор приложения AIR для публикации;
- решение, смогут ли пользователи просматривать исходный код, и определение файлов приложения для включения (дополнительно);
- цифровая подпись приложения AIR с использованием сертификата подписи коммерческого кодекса или посредством создания и применения собственноручно выполненной подписи;
- создание промежуточного файла AIR, который будет подписан позднее (дополнительно).

Упаковка приложения AIR

1 Откройте проект и убедитесь, что приложение не содержит ошибок компиляции и выполняется надлежащим образом.

2 Выберите «Проект» > «Экспорт сборки выпуска».

3 Если в программе Flex Builder открыто несколько проектов и приложений, выберите проект AIR, который необходимо упаковать.

4 (Дополнительно) Если необходимо разрешить пользователям просматривать исходный код при работе с приложением, выберите «Включить представление "Код"». Отдельные файлы можно исключить, выбрав пункт «Выбрать исходные файлы». Все исходные файлы выбраны по умолчанию. Дополнительные сведения о публикации исходных файлов во Flex Builder см. в справке Flex Builder.

5 Также по желанию можно изменить имя создаваемого файла AIR. Для продолжения нажмите «Далее». При этом для приложения создается цифровая подпись.

Цифровая подпись для приложений AIR

Перед продолжением работы с версией выпуска для экспорта следует выбрать способ цифровой подписи для пользовательского приложения AIR. Для этого существует несколько вариантов. Можно подписать приложение с использованием сертификата подписи коммерческого кодекса или посредством создания и применения собственноручно выполненной подписи, а также можно упаковать приложение и подписать его позднее.

Цифровые сертификаты, выпускаемые органами по сертификации, такими как VeriSign, Thawte, GlobalSign и ChosenSecurity, гарантируют пользователям подлинность личности разработчика как издателя и удостоверяют, что файл установки не был изменен со времени его подписания разработчиком. Собственноручно выполненные цифровые сертификаты преследуют ту же цель, но без гарантий от третьей стороны.

Кроме того, предоставляется возможность упаковки приложения AIR без цифровой подписи и создания промежуточного файла AIR (.airi). Промежуточный файл AIR установить невозможно, и поэтому он является недействительным файлом. Он используется разработчиком для тестирования и может запускаться с помощью инструмента командной строки AIR ADT. AIR обеспечивает такую возможность, поскольку в некоторых средах разработки подписи выполняются отдельным разработчиком или группой. Такой подход обеспечивает дополнительную безопасность при управлении цифровыми сертификатами.

Дополнительные сведения о подписи приложений см. в разделе ««Цифровая подпись файлов AIR» на странице 96».

Цифровая подпись пользовательских приложений AIR

1. Приложение AIR можно снабдить цифровой подписью, выбрав существующий цифровой сертификат или создав новый собственноручно подписанный сертификат. Выберите «Экспортировать и подписать файл AIR» с функцией «Цифровой сертификат».
2. Если имеется существующий цифровой сертификат, нажмите «Обзор», чтобы перейти к нему и выбрать его.
3. Для создания нового собственноручно подписанного сертификата нажмите «Создать».
4. Введите необходимую информацию и нажмите «ОК».
5. Чтобы выбрать файлы для исключения из экспортируемого файла AIR, нажмите «Далее» (дополнительно). Все файлы выбраны по умолчанию.
6. Для создания файла AIR нажмите «Готово».

Создание промежуточного файла AIR

- Выберите пункт «Экспортировать промежуточный файл AIRI путем экспорта позднее». Для создания промежуточного файла нажмите «Готово».

После создания промежуточного файла AIR его можно подписать с помощью инструмента ADT командной строки (см. раздел ««Подписание промежуточного файла AIR с помощью ADT» на странице 68»).

Создать проект библиотеки AIR

Для создания библиотеки кодов AIR для проектов AIR следует создать проект библиотеки AIR с помощью стандартного мастера создания проектов библиотеки Flex.

1. Выберите пункты меню «Файл» > «Создать» > «Проект библиотеки Flex».
2. Укажите имя проекта.
3. Выберите «Добавить библиотеки Adobe AIR» и нажмите кнопку «Далее».

   Примечание. *Выбранная версия Flex SDK должна поддерживать AIR. Flex 2.0.1 SDK не поддерживает AIR.*
4. Измените требуемый путь сборки и нажмите «Готово». Дополнительные сведения о создании проектов библиотек см. в разделе «О проектах библиотек» в справке по Flex Builder.

Установка расширения AIR для Dreamweaver

С помощью расширения AIR для Dreamweaver можно создавать мультимедийные приложения для настольного компьютера. Например, это может быть набор веб-страниц, которые взаимодействуют друг с другом для отображения XML-данных. Расширение Adobe AIR для Dreamweaver можно использовать для упаковки этого набора страниц в небольшое приложение, пригодное для установки на компьютер пользователя. Когда пользователь запускает на компьютере это приложение, оно отображает веб-сайт в собственном окне, не используя обозреватель. Таким образом, пользователь может просматривать веб-сайт локально без подключения к Интернету.

Adobe AIR не поддерживает динамические страницы, например страницы Adobe® ColdFusion® и PHP. Среда выполнения способна обрабатывать только страницы HTML и JavaScript. Однако JavaScript можно использовать на страницах для вызова веб-служб Интернета, включая службы на основе ColdFusion или PHP, методами Ajax, например XMLHTTPRequest или с помощью различных API Adobe AIR.

Системные требования

Для использования расширения Adobe AIR для Dreamweaver необходимо установить и правильно настроить на компьютере следующее ПО.

- Dreamweaver CS3 или Dreamweaver CS4
- Adobe® Extension Manager CS3
- Java JRE 1.4 или более поздней версии (требуется для создания файла Adobe AIR). Java JRE можно загрузить с веб-сайта http://java.sun.com/.

  Предыдущие требования распространяются только для создания и просмотра приложений Adobe AIR в Dreamweaver. Чтобы установить и запустить приложение Adobe AIR на компьютере, необходимо также установить на компьютер Adobe AIR. Загрузить среду выполнения можно с веб-страницы www.adobe.com/go/air_ru.

Установка расширения Adobe AIR для Dreamweaver

1. Загрузить расширение Adobe AIR для Dreamweaver можно с веб-страницы http://www.adobe.com/products/air/tools/ajax/.
2. Дважды нажмите файл расширения .mxp в проводнике Windows (в ОС Windows) или в программе Finder (в ОС Macintosh).
3. Для установки следуйте инструкциям на экране.
4. По окончании установки переустановите Dreamweaver.

   Сведения об использовании расширения Adobe AIR для Dreamweaver см. в разделе «Использование расширения AIR для Dreamweaver».

Установка AIR SDK

В состав SDK Adobe AIR входят следующие инструменты командной строки для запуска и упаковки приложений.

AIR Debug Launcher (ADL) Позволяет запускать приложения AIR, не устанавливая их. См. раздел ««Использование средства AIR Debug Launcher (ADL)» на странице 51».

AIR Development Tool (ADT) Предназначен для упаковки приложений AIR в развертываемые установочные пакеты. См. раздел ««Упаковка установочного файла AIR с помощью AIR Developer Tool (ADT)» на странице 54».

Для работы инструментов командной строки AIR требуется Java. Можно использовать виртуальную машину Java из комплекта JRE или JDK (версии 1.4 или более поздней версии). Java JRE и Java JDK можно загрузить с веб-сайта http://java.sun.com/.

Примечание. *Для запуска приложений AIR конечным пользователям устанавливать Java не требуется.*

Загрузка и установка AIR SDK

Ниже описано, как загрузить и установить AIR SDK:

Установка AIR SDK в ОС Windows

- Загрузите установочный файл AIR SDK.
- AIR SDK распространяется в виде стандартного файла архива. Чтобы установить AIR, извлеките содержимое SDK в папку на компьютере (например, в C:\Program Files\Adobe\AIRSDK или C:\AIRSDK).
- Инструменты ADL и ADT содержатся в папке bin комплекта AIR SDK; добавьте этот путь в переменную среды PATH.

Установка AIR SDK в ОС Mac OS X

- Загрузите установочный файл AIR SDK.
- AIR SDK распространяется в виде стандартного файла архива. Чтобы установить AIR, извлеките содержимое SDK в папку на компьютере (например, в: /Users/<имя_пользователя>/Applications/AIRSDK).
- Инструменты ADL и ADT содержатся в папке bin комплекта AIR SDK; добавьте этот путь в переменную среды PATH.

Установка AIR SDK в ОС Linux

- Пакет SDK доступен в формате tbz2.
- Чтобы установить SDK, создайте папку и распакуйте в нее содержимое SDK, используя команду tar -jxvf <путь к файлу AIR-SDK.tbz2>

Сведения о начале работы с инструментами AIR SDK см. в разделе «Создание приложения AIR с помощью инструментов командной строки».

Состав пакета AIR SDK

В таблице ниже приводится описание файлов пакета AIR SDK:

Папка SDK	Описание файлов/инструментов
BIN	adl.exe — AIR Debug Launcher (ADL), позволяет запустить приложение без предварительной упаковки и установки. Дополнительные сведения см. в разделе ««Использование средства AIR Debug Launcher (ADL)» на странице 51». adt.bat — AIR Developer Tool (ADT), упаковывает приложение в AIR-файл для распространения. Дополнительные сведения см. в разделе ««Упаковка установочного файла AIR с помощью AIR Developer Tool (ADT)» на странице 54».
FRAMEWORKS	AIRAliases.js — содержит псевдонимы определений для доступа к классам среды выполнения ActionScript. Дополнительные сведения об использовании файла псевдонимов см. в разделе «Использование файла AIRAliases.js». fileservicemonitor.swf дает приложениям AIR возможность реагировать на изменения сетевого подключения к указанному компьютеру посредствам событий. Сведения об использовании этой среды см. в разделе Изменения сетевого соединения (для разработчиков ActionScript) или Изменения сетевого соединения (для разработчиков HTML).
LIB	adt.jar — исполняемый файл adt, вызываемый файлом adt.bat. Descriptor.1.0.xsd — файл схемы приложения.
RUNTIME	Среда выполнения AIR — используется ADL для запуска приложений AIR до их распаковки или установки.
SAMPLES	В этой папке содержится пример файла дескриптора приложения, пример функции незаметной установки (badge.swf), а также значки приложения AIR по умолчанию; см. раздел ««Распространение, установка и запуск приложений AIR» на странице 86».
SRC	В этой папке содержатся исходные файлы, используемые для демонстрации незаметной установки.
TEMPLATES	descriptor-template.xml — шаблон файла дескриптора приложения, необходимого для каждого приложения AIR. Дополнительные сведения о файле дескриптора приложения см. в разделе ««Задание свойств приложения AIR» на странице 71».

Настройка Flex SDK

Создать приложение Adobe® AIR® с помощью Adobe® Flex™ можно одним из следующих способов.

- Можно загрузить и установить пакет Adobe® Flash® Builder™, в котором содержатся инструменты для создания проектов Adobe AIR, их проверки, отладки и упаковки приложений AIR. См. раздел ««Создание первого приложения Flex AIR во Flash Builder или Flex Builder» на странице 26».
- Можно загрузить Adobe® Flex™ SDK и разрабатывать приложения Flex AIR с помощью привычного текстового редактора и инструментов командной строки.

Сведения об инструментах командной строки AIR в пакете Flex SDK

Используемые для создания приложения AIR инструменты командной строки вызывают соответствующие инструменты, используемые для создания приложений Flex.

- amxmlc вызывает mxmlc для компиляции классов приложения
- acompc вызывает compc для компиляции классов библиотек и компонентов
- aasdoc вызывает asdoc для создания файлов документации из комментариев в исходном коде

Единственное отличие между версиями служебных программ Flex и AIR заключается в том, что версии AIR загружают параметры конфигурации из файла air-config.xml, на не flex-config.xml.

Полное описание инструментово Flex SDK и их параметров командной строки содержится в руководстве «Создание и развертывание приложений Flex» в библиотеке документации по Flex. Здесь содержится лишь краткое описание инструментово Flex SDK, необходимое для понимания разницы между созданием приложений Flex и приложений AIR.

Дополнительные сведения см. в разделе «Создание приложения AIR с помощью инструментов командной строки».

Установка Flex SDK

Для создания приложений AIR с помощью инструментов командной строки необходимо, чтобы на компьютере была установлена Java. Можно использовать виртуальную машину Java из комплекта JRE или JDK (версии 1.5 или более поздней версии). Java JRE и Java JDK можно загрузить с веб-сайта http://java.sun.com/.

Примечание. *Для запуска приложений AIR конечным пользователям устанавливать Java не требуется.*

В состав Flex SDK входят инструменты командной строки для упаковки, компиляции и отладки приложений AIR.

1. Если эта задача еще не выполнялась, загрузите и установите Flex SDK с веб-страницы http://opensource.adobe.com/wiki/display/flexsdk/Downloads.
2. Распакуйте содержимое SDK в папку (например, во Flex SDK).
3. Инструменты командной строки находятся в папке bin.

Дополнительные сведения см. в разделе ««Создание первого приложения AIR с использованием пакета Flex SDK» на странице 31».

Настройка компилятора

Обычно параметры компиляции вводятся в командной строке и одном из файлов конфигурации. В глобальном файле конфигурации Flex SDK обычно хранятся значения по умолчанию, используемые каждый раз при запуске компиляторов. Чтобы настроить Flex SDK под собственную среду разработки, внесите изменения в этот файл. Существует два глобальных файла конфигурации Flex, расположенные в папке frameworks в каталоге установки Flex SDK. air-config.xml — используется при запуске компилятора amxmlc. Этот файл настраивает компилятор для AIR путем включения библиотек AIR. flex-config.xml file — используется при запуске компилятора mxmlc.

Значения по умолчанию можно оставить на время ознакомления с Flex и AIR, однако при работе над полномасштабным проектом рекомендуется подробно изучить различные варианты их настройки. В локальном файле конфигурации можно указать значения параметров компилятора для данного проекта, который имеет приоритет над глобальными значениями для данного проекта. Полный список параметров компиляции и описание синтаксиса файлов конфигурации содержится в руководстве «Создание и развертывание приложений Flex» в библиотеке документации по Flex.

Примечание. *Для приложений AIR не существует специальных параметров компиляции, однако при компиляции приложения AIR необходимы библиотеки AIR. Обычно ссылки на эти библиотеки размещаются в файле конфигурации проекта, в файле инструмента для создания, например Ant, или непосредственно в командной строке.*

Настройка отладчика

AIR поддерживает непосредственную отладку, что исключает необходимость в использовании отладочной версии среды выполнения (как, например, в ситуации с проигрывателем Adobe® Flash® Player). Для отладки с использованием командной строки воспользуйтесь отладчиком Flash Debugger или AIR Debug Launcher (ADL).

Отладчик Flash Debugger содержится в папке Flex SDK. Собственные версии, например fdb.exe для ОС Windows, находятся во вложенной папке bin. Java-версия находится во вложенной папке lib. AIR Debug Launcher, adl.exe, находится в папке bin в каталоге установки Flex SDK. (Отдельной Java-версии нет).

Примечание. *Запустить приложение AIR непосредственно с помощью fdb нельзя, так как fdb пытается осуществить его запуск с помощью проигрывателя Flash Player. Вместо этого позвольте приложению AIR подключиться к активному сеансу fdb.*

Дополнительные сведения см. в разделе ««Использование средства AIR Debug Launcher (ADL)» на странице 51».

Настройка упаковщика приложения

AIR Developer Tool (ADT) для упаковки приложений в установочный AIR-файл представляет собой программу Java. Для работы с этой служебной программой не требуется выполнять других настроек, кроме настройки используемой среды.

В SDK входит файл сценария (находится в папке bin) для выполнения ADT в качестве команды. ADT также можно запустить в качестве программы Java, что удобно при использовании таких инструментов, как Apache Ant.

Дополнительные сведения см. в разделе ««Упаковка установочного файла AIR с помощью AIR Developer Tool (ADT)» на странице 54».

Глава 5. Создание первого приложения Flex AIR во Flash Builder или Flex Builder

Чтобы быстро на практике продемонстрировать, как работает Adobe® AIR®, используйте эти инструкции по созданию и упаковке простейшего SWF-приложения AIR «Hello World» с помощью Adobe® Flash® Builder.

Загрузите и установите Flex Builder 3, если это еще не сделано. Дополнительные сведения см. в разделе ««Настройка Flex SDK» на странице 23».

Создание проекта AIR

Flash Builder включает инструменты, необходимые для разработки и упаковки приложений AIR.

В начале для создания приложений AIR во Flash Builder или Flex Builder выполняется тот же шаг, что и при создании других проектов приложений Flex: определяется новый проект.

Примечание. *Эти инструкции относятся к среде Flash Builder. (Они также применимы для среды Flex Builder 3.)*

1. Запустите Flash Builder.
2. Выберите «Файл» > «Создать» > «Проект Flex».
3. Введите AIRHelloWorld в качестве имени проекта.
4. Приложения AIR считаются типом приложений во Flex. Существует два типа.
   - Интернет-приложение Flex, запускаемое в проигрывателе Adobe® Flash® Player.
   - Настольное приложение AIR, запускаемое в среде Adobe AIR.

   В качестве типа приложения выберите настольное приложение.
5. Поскольку серверная технология использоваться не будет, необходимо выбрать «Нет», а затем нажать кнопку «Далее».
6. Выберите каталог для хранения откомпилированного приложения. По умолчанию таким каталогом является bin. Для создания проекта нажмите «Готово».

Проекты AIR изначально состоят из двух файлов: основного файла MXML и файла приложения XML (также называется файлом дескриптора приложения). В последнем файле указаны параметры для идентификации, установки и запуска приложений AIR. Могут возникнуть ситуации, когда потребуется отредактировать этот файл вручную. Пока учтите, что такой файл существует.

Дополнительные сведения см. в разделах Разработка приложений AIR в среде Flex Builder и Разработка приложений AIR в среде Flash Builder.

Написание кода приложения AIR

Для написания кода приложения «Hello World» выполняется правка файла приложения MXML (AIRHelloWorld.mxml), который открывается в редакторе. Если файл не открыт, используйте навигатор проекта, чтобы открыть его.

Приложения Flex AIR содержатся внутри тега WindowedApplication в MXML. Тег WindowedApplication в MXML создает простое окно, включающее простейшие элементы управления окна, такие как строка заголовка и кнопка закрытия окна.

1 Добавьте атрибут `title` к компоненту WindowedApplication и назначьте ему значение `"Hello World"`:

```
?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"
                       title="Hello World">
</s:WindowedApplication>
```

2 Добавьте в приложение компонент Label (поместите его внутрь тега WindowedApplication). Задайте для свойства `text` компонента Label значение `"Hello AIR"` и установите ограничения макета, чтобы обеспечить центрирование элемента, как показано на следующем рисунке:

```
<?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"
                       title="Hello World">

    <s:Label text="Hello AIR" horizontalCenter="0" verticalCenter="0"/>
</s:WindowedApplication>
```

3 Добавьте следующий блок стиля сразу после открытия тега WindowedApplication и перед тегом только что введенного компонента Label.

```
<fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    s|WindowedApplication
    {

skinClass:ClassReference("spark.skins.spark.SparkChromeWindowedApplicationSkin");
        background-color:#999999;
        background-alpha:"0.7";
    }
</fx:Style>
```

Эти настройки стиля применяются ко всему приложению и визуализируют фон окна в немного прозрачный серый цвет.

Код приложения теперь выглядит следующим образом:

```
<?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"
                       title="Hello World">

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        s|WindowedApplication
        {

skinClass:ClassReference("spark.skins.spark.SparkChromeWindowedApplicationSkin");
            background-color:#999999;
            background-alpha:"0.7";
        }
    </fx:Style>

    <s:Label text="Hello AIR" horizontalCenter="0" verticalCenter="0"/>
</s:WindowedApplication>
```

Далее будут изменены некоторые настройки, позволяющие обеспечить прозрачность в приложении:

1. На панели навигатора Flex найдите файл дескриптора приложения в исходном каталоге проекта. Если проект назван AIRHelloWorld, этот файл будет называться AIRHelloWorld-app.xml.
2. Дважды щелкните файл дескриптора приложения, чтобы отредактировать его в среде Flash Builder.
3. В коде XML найдите закомментированные строки для свойств `systemChrome` и `transparent` (в свойстве `initialWindow`). Удалите комментарии. (Удалите разграничители комментариев `"<!--"` и `"-->"`.)
4. Задайте для свойства `systemChrome` текстовое значение `none`, как в следующем примере:

   ```
   <systemChrome>none</systemChrome>
   ```

5. Задайте для свойства `transparent` текстовое значение `true`, как в следующем примере:

   ```
   <transparent>true</transparent>
   ```

6. Сохраните файл.

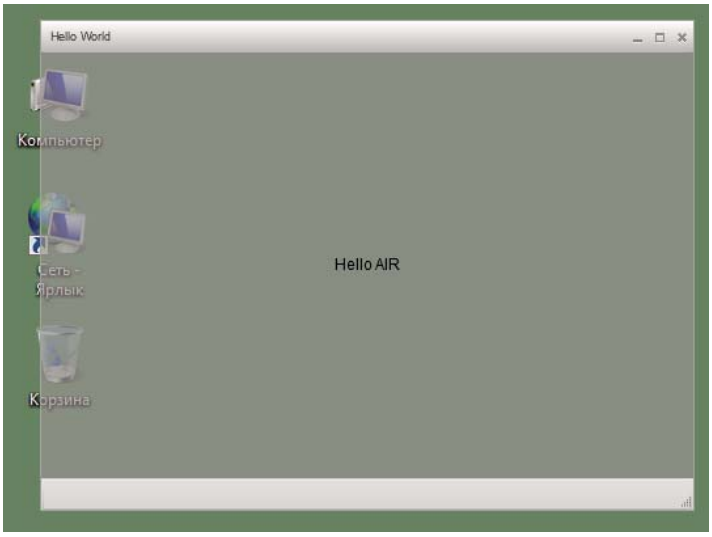
Проверка приложения AIR

Чтобы проверить написанный код приложения, выполните его в режиме отладки.

1 Нажмите кнопку «Отладка» на главной панели инструментов

Также можно выбрать в меню «Запуск» > «Отладка» > AIRHelloWorld.

Получившееся приложение AIR должно выглядеть как в следующем примере (зеленый фон — это рабочий стол):



2 С использованием свойств `horizontalCenter` и `verrticalCenter` элемента управления Label текст размещается в центре окна. Перемещайте или изменяйте размер окна также, как в любом другом приложении для настольного компьютера.

Примечание. *Если приложение не компилируется, наудите и исправьте ошибки в синтаксисе или написании введенного кода. Ошибки и предупреждения отображаются в среде Flash Builder с помощью представления «Проблемы».*

Упаковка, подписывание и выполнение приложения AIR

Теперь можно упаковать приложение «Hello World» в файл AIR для распространения. Файл AIR — это архивный файл, содержащий файлы приложения, к которым относятся все файлы в папке bin данного проекта. В этом простом примере такими файлами являются SWF- и XML-файлы приложения. Пакет AIR распространяется среди пользователей, которые затем используют его для установки приложения. Обязательным шагом в этом процессе является цифровая подпись приложения.

1 Убедитесь, что приложение правильно скомпилировано и работает как предполагалось.

2 Выберите «Проект» > «Экспорт сборки выпуска».

3 Если открыты несколько проектов и приложений, выберите проект AIR, который необходимо упаковать. Затем нажмите кнопку «Далее»

4 Выберите «Экспортировать и подписать файл AIR» с функцией «Цифровой сертификат».

5 Если имеется существующий цифровой сертификат, нажмите «Обзор», чтобы перейти к нему и выбрать его.

6 Если необходимо создать новый самозаверяющий цифровой сертификат, выберите «Создать».

7 Введите необходимую информацию и нажмите «ОК».

8 Нажмите кнопку «Готово», чтобы создать пакет AIR с названием AIRHelloWorld.air.

Теперь можно запустить приложение из навигатора проектов в среде Flash Builder или из файловой системы, дважды щелкнув соответствующий файл AIR.

Глава 6. Создание первого приложения AIR с использованием пакета Flex SDK

Для быстрого и наглядного представления принципов работы Adobe® AIR® используйте эти инструкции по созданию простого SWF-приложения AIR «Hello World» с помощью пакета Flex SDK. В этом пособии показаны принципы компиляции, тестирования и упаковки приложения AIR с использованием инструментов командной строки, доступных в пакете SDK.

Для начала необходимо установить среду выполнения и настроить Adobe® Flex™. В этом пособии показано использование компилятора *AMXMLC*, средства запуска *AIR Debug Launcher* (ADL) и инструмента *AIR Developer Tool* (ADT). Эти программы находятся в каталоге `bin` среды Flex SDK (см. раздел ««Настройка Flex SDK» на странице 23»).

Создание файла дескриптора приложения AIR

В этом разделе описаны принципы создания дескриптора приложения, представляющего собой XML-файл со следующей структурой:

```
<application>
    <id>...</id>
    <version>...</version>
    <filename>…</filename>
    <initialWindow>
        <content>…</content>
        <visible>…</visible>
        <systemChrome>…</systemChrome>
        <transparent>…</transparent>
        <width>…</width>
        <height>…</height>
    </initialWindow>
</application>
```

1 Создайте XML-файл `HelloWorld-app.xml` и сохраните его в каталоге проекта.

2 Добавьте элемент `<application>`, включая атрибут пространства имен AIR:

<application xmlns="http://ns.adobe.com/air/application/2.0"> Последний сегмент пространства имен (2.0) задает версию среды выполнения, которая требуется для приложения.

3 Добавьте элемент `<id>`:

<id>samples.flex.HelloWorld</id> Идентификатор приложения однозначно идентифицирует приложение в сочетании с идентификатором издателя (который AIR извлекает из сертификата, используемого для подписи пакета приложения). Рекомендуемым форматом является обратная запись строки DNS с использованием точки в качестве разделителя, например `"com.company.AppName"`. Идентификатор приложения используется для установки, обращения к частному каталогу хранилища файловой системы приложения, доступа к частному зашифрованному хранилищу и взаимодействия между приложениями.

4 Добавьте элемент `<version>`:

<version>1.0</version> Помогает пользователям определить, какую версию приложения они устанавливают.

5 Добавьте элемент `<filename>`:

<filename>HelloWorld</filename> Имя, используемое для выполняемого файла приложения, каталога установки и аналогичных ссылок на приложение в операционной системе.

6 Добавьте элемент `<initialWindow>`, содержащий следующие дочерние элементы, чтобы указать свойства исходного окна приложения:

<content>HelloWorld.swf</content> Определяет корневой HTML-файл приложения AIR для загрузки.

<visible>true</visible> Сразу делает окно видимым.

<width>400</width> Задает ширину окна (в пикселах).

<height>200</height> Задает высоту окна.

7 Сохраните файл. По завершении файл дескриптора приложения должен иметь следующий вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://ns.adobe.com/air/application/2.0">
    <id>samples.flex.HelloWorld</id>
    <version>0.1</version>
    <filename>HelloWorld</filename>
    <initialWindow>
        <content>HelloWorld.swf</content>
        <visible>true</visible>
        <systemChrome>none</systemChrome>
        <transparent>true</transparent>
        <width>400</width>
        <height>200</height>
    </initialWindow>
</application>
```

В этом примере устанавливается всего несколько возможных свойств приложения. Полный набор свойств приложения, которые позволяют указывать такие аспекты, как Chrome окна, размер окна, прозрачность, каталог установки по умолчанию, связанные типы файлов и значки приложения, см. в разделе ««Задание свойств приложения AIR» на странице 71»

Написание кода приложения

Примечание. *В SWF-приложениях AIR может использоваться основной класс, заданный с использованием MXML или Adobe® ActionScript® 3.0. В этом примере MXML-файл используется для определения основного класса. Процесс создания приложения AIR с использованием основного класса ActionScript аналогичен. Вместо компиляции файла MXML в SWF-файл выполняется компиляция файла класса ActionScript. При использовании ActionScript основной класс должен расширять flash.display.Sprite.*

Как и во всех приложениях Flex, в приложениях AIR, созданных в среде Flex, содержится основной файл MXML. Однако в приложениях AIR в качестве корневого элемента используется компонент *WindowedApplication*, а не компонент Application. Компонент WindowedApplication предоставляет свойства, методы и события для управления приложением и его начальным окном.

Следующая процедура позволяет создать приложение «Hello World»:

1 В текстовом редакторе создайте файл `HelloWorld.mxml` и добавьте следующий код MXML:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute" title="Hello World">
</mx:WindowedApplication>
```

2 Далее добавьте компонент Label в приложение (поместите его внутрь тега WindowedApplication).

3 Задайте для свойства `text` компонента Label значение `"Hello AIR"`.

4 Установите ограничения макета, чтобы он всегда располагался по центру.

В следующем примере показан код на данном этапе:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
    title="Hello World">
        <mx:Label text="Hello World" horizontalCenter="0" verticalCenter="0"/>
</mx:WindowedApplication>
```

5 Добавьте следующий блок стилей:

```
<mx:Style>
    WindowedApplication
    {
        background-color:"0x999999";
        background-alpha:"0.5";
    }
</mx:Style>
```

Эти стили применяются ко всему приложению и задают слегка прозрачный серый цвет фона окна.

Полный код приложения теперь имеет следующий вид:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
title="Hello World">
    <mx:Style>
        WindowedApplication
        {
            background-color:"0x999999";
            background-alpha:"0.5";
        }
    </mx:Style>
    <mx:Label text="Hello World" horizontalCenter="0" verticalCenter="0"/>
</mx:WindowedApplication>
```

Компиляция приложения

Перед выполнением и отладкой приложения скомпилируйте код MXML в SWF-файл с использованием компилятора amxmlc. Компилятор amxmlc находится в каталоге `bin` пакета Flex SDK. При необходимости можно включить каталог bin пакета Flex SDK в переменную среды для пути на компьютере. Настройка пути делает выполнение утилит в командной строке более простым.

1 Откройте командную среду или терминал команд и перейдите к папке проекта приложения AIR.

2 Введите следующую команду:

```
amxmlc HelloWorld.mxml
```

При выполнении компилятора `amxmlc` создается файл `HelloWorld.swf`, содержащий скомпилированный код приложения.

Примечание. *Если не удается скомпилировать приложение, исправьте синтаксические или орфографические ошибки. Ошибки и предупреждения отображаются в окне консоли, используемом для выполнения компилятора amxmlc.*

Дополнительные сведения см. в разделе ««Компиляция исходных файлов MXML и ActionScript для AIR» на странице 48».

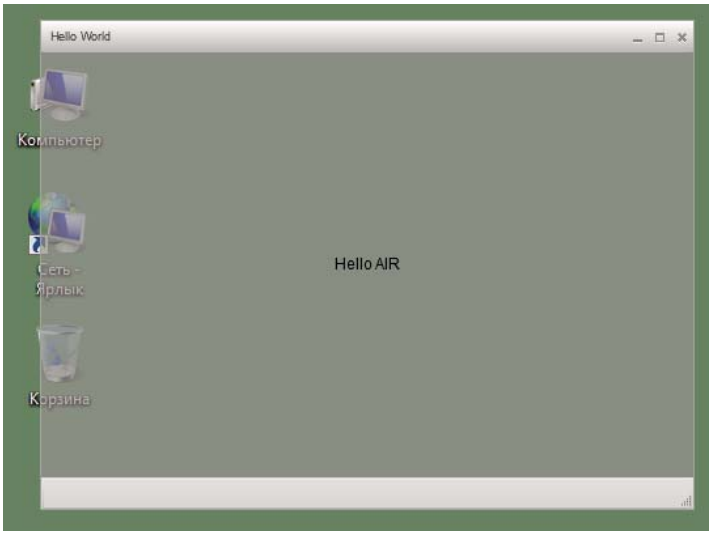
Проверка приложения

Чтобы выполнить и протестировать приложение из командной строки, используйте инструмент AIR Debug Launcher (ADL) для запуска приложения с помощью соответствующего файла дескриптора приложения. (Инструмент ADL находится в каталоге bin пакета Flex SDK.)

❖ В командной строке введите следующую команду:

```
adl HelloWorld-app.xml
```

Получившееся в результате приложение AIR имеет вид, аналогичный показанному на рисунке (зеленый фон представляет рабочий стол пользователя):



С использованием свойств horizontalCenter и verticalCenter элемента управления Label текст размещается в центре окна. Перемещайте или изменяйте размер окна также, как в любом другом приложении для настольного компьютера.

Дополнительные сведения см. в разделе ««Использование средства AIR Debug Launcher (ADL)» на странице 51».

Создание файла установки AIR

Если приложение успешно запущено, можно использовать утилиту ADT, чтобы упаковать приложение в файл установки AIR. Файл установки AIR — это файл архива, в котором содержаться все файлы приложения, которые можно передавать другим пользователям. Перед установкой такого упакованного файла AIR необходимо установить Adobe AIR.

Чтобы обеспечить защиту приложения, все файлы установки AIR должны содержать цифровую подпись. С помощью ADT или другого инструмента генерации сертификатов можно создать простой, самозаверяющий сертификат, используемый в целях разработки. Можно также приобрести коммерческий сертификат подписи кода в коммерческом центре сертификации. Если пользователь устанавливает файл AIR с самозаверяющим сертификатом, то в ходе процесса установки показывается «неизвестный» издатель. Это объясняется тем, что самозаверяющие сертификаты гарантируют лишь то, что файл AIR не изменялся с момента своего создания. Ничто не сможет помешать злоумышленнику сделать файл AIR с самозаверяющим сертификатом и представить его как ваше приложение. Поэтому в выпускаемых для широкого использования файлах AIR рекомендуется использовать обеспечивающие возможность проверки коммерческие сертификаты. Краткие сведения о проблемах системы защиты AIR см. в разделе Система защиты AIR (для разработчиков ActionScript) или Система защиты AIR (для разработчиков HTML).

Создание самозаверяющих подписывающих сертификатов и пары ключей

❖ В командной строке введите следующую команду (выполняемый файл ADT находится в каталоге `bin` пакета Flex SDK):

```
adt –certificate -cn SelfSigned 1024-RSA sampleCert.pfxsamplePassword
```

В этом примере используется минимальный набор атрибутов, которые могут быть установлены для сертификата. Можно использовать любые значения для параметров, обозначенных *курсивом*. Ключ должен иметь тип *1024-RSA* или *2048-RSA* (см. раздел ««Цифровая подпись файлов AIR» на странице 96»).

Создание файла установки AIR

❖ Из командной строки выполните следующую команду (вводится одной строкой):

```
adt -package -storetype pkcs12 -keystore sampleCert.pfx HelloWorld.air
HelloWorld-app.xml HelloWorld.swf
```

Будет выдано предложение ввести пароль для файла ключей.

Аргумент HelloWorld.air — это файл AIR, создаваемый ADT. HelloWorld-app.xml — это файл дескриптора приложения. Последующими аргументами являются файлы, используемые приложением. В этом примере используются только три файла, но можно включить любое число файлов и каталогов.

После создания пакета AIR можно установить и выполнить приложение, дважды щелкнув этот упакованный файл. Также можно ввести имя файла AIR, используя команду в оболочке ОС или консоль управления.

Дополнительные сведения см. в разделе ««Упаковка установочного файла AIR с помощью AIR Developer Tool (ADT)» на странице 54».

Глава 7. Создание первого приложения AIR с помощью Flash Professional

Для быстрого практического знакомства с работой Adobe® AIR® следуйте инструкциям в этом разделе, создав и упаковав простейшее приложение AIR «Hello World» с помощью приложения Adobe® Flash® Professional.

Создание приложения «Hello World» в Adobe Flash

Создание приложения Adobe AIR в Adobe Flash во многом напоминает процесс создания любого другого FLA-файла. Создание файла Flash (Adobe AIR) начинается с экрана приветствия. Затем выполняется создание приложения, определяются параметры установки и осуществляется установка приложения AIR. В следующей процедуре описаны шаги, позволяющие создать простейшее приложение Hello World с помощью Flash Professional.

Создание приложения «Hello World»

1. Запустите приложение Flash.
2. В окне приветствия щелкните Flash-файл (Adobe AIR), чтобы создать пустой FLA-файл с параметрами публикации Adobe AIR.
3. Выберите инструмент «Текст» на панели «Инструменты» и создайте статическое текстовое поле (по умолчанию) в центре рабочей области. Сделайте поле достаточно широким, чтобы в нем помещалось 15-20 символов.
4. Введите текст «Hello World» в это текстовое поле.
5. Сохраните файл, переименовав его (например, в helloAIR).

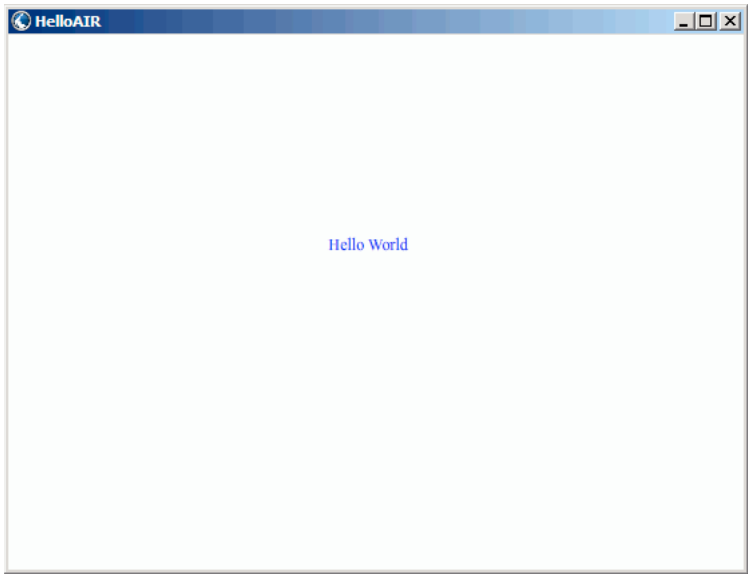
Проверка приложения

1. Нажмите клавиши Ctrl + Enter или выберите «Управление» ->«Тестировать ролик», чтобы проверить приложение в Adobe AIR.
2. Для использования функции «Отладка ролика» вначале добавьте к приложению программный код ActionScript. Можно попробовать сделать это быстро, добавив инструкцию трассировки, как показано в следующем примере.

   ```
   trace("Running AIR application using Debug Movie");
   ```

3. Нажмите клавиши Ctrl + Shift + Enter или выберите «Управление» -> «Отладка ролика», чтобы запустить приложение в режиме отладки ролика.

Приложение «Hello World» выглядит, как показано на следующем рисунке.



Упаковка приложения

1 Выберите «Файл» > «Опубликовать».

2 Подпишите пакет Adobe AIR с использованием существующего цифрового сертификата или создайте самозаверенный сертификат, выполнив следующие действия:

   a Нажмите кнопку «Создать...», чтобы открылось диалоговое окно «Создание цифрового сертификата для автоматического подписывания»

   b Введите сведения в поля «Имя владельца», «Подразделение», «Название организации», «Электронная почта», «Страна», «Пароль» и «Подтверждение пароля».

   c Укажите тип сертификата. Тип сертификата определяет уровень безопасности: 1024-RSA использует 1024-разрядный ключ (менее защищенный), а 2048-RSA использует 2048-разрядный ключ (более защищенный).

   d Сохраните информацию в файле сертификата, выбрав строчку «Сохранить как», или нажмите кнопку «Обзор...», чтобы найти требуемую папку. (Например, *C:/Temp/mycert.pfx*.) Когда закончите, нажмите кнопку «ОК».

   e Adobe Flash открывает диалоговое окно «Цифровая подпись». Имя файла и путь к созданному цифровому сертификату для автоматического подписывания отображается в текстовом поле «Сертификат». Если этой информации нет, введите имя файла и путь к нему или нажмите кнопку «Обзор», чтобы найти файл и выбрать его.

   f В текстовом поле «Пароль» диалогового окна «Цифровая подпись» введите пароль, совпадающий с паролем, назначенным на шаге c. Дополнительные сведения о создании подписей приложений Adobe AIR см. в разделе ««Цифровая подпись файлов AIR» на странице 96».

3 Чтобы создать приложение и файл программы установки, нажмите кнопку «Опубликовать файл». (В среде Flash CS4 нажмите кнопку «ОК».) Необходимо выполнить команды «Тестировать ролик» или «Отладка ролика», чтобы создать SWF-файл и xml-файлы приложения перед созданием файла Adobe AIR.

4 Чтобы установить приложение, дважды щелкните AIR-файл (*application*.air) в той папке, где сохранено приложение.

5 Нажмите кнопку «Установить» в диалоговом окне «Установка приложения».

6 Просмотрите настройки установки и параметры расположения и убедитесь, что установлен флажок «Запуск приложения после установки». Нажмите кнопку «Продолжить».

7 Нажмите кнопку «Готово», когда появится сообщение о завершении установки приложения.

Глава 8. Создание первого HTML-приложения AIR с помощью Dreamweaver

Чтобы быстро на практике продемонстрировать как работает Adobe® AIR®, используйте эти инструкции по созданию и упаковке простейшего HTML-приложения AIR «Hello World», используя расширение Adobe® AIR® Extension для Dreamweaver®.

Если это еще не сделано, загрузите и установите пакет Adobe AIR, расположенный по следующему адресу: www.adobe.com/go/air_ru.

Инструкции по установке расширения Adobe AIR для Dreamweaver см. в разделе Установка расширения AIR для Dreamweaver.

Обзор расширения, включая системные требования, см. в разделе Расширение AIR для Dreamweaver.

Подготовка файлов приложений

Начальная страница приложение Adobe AIR и все относящиеся к нему страницы должны быть определены на сайте Dreamweaver:

1. Запустите Dreamweaver и убедитесь, что был определен сайт.
2. Откройте новую страницу HTML, выбрав «Файл» > «Создать», затем выберите HTML в столбце «Тип страницы», выберите «Нет» в столбце «Макет», щелкните «Создать».
3. На новой странице введите **Hello World!**

   Этот пример исключительно прост, но, если потребуется, можно добавить к тексту стиль по своему выбору, добавить дополнительное содержимое к странице, связать другие страницы с начальной и т.п.
4. Сохраните страницы («Файл» > «Сохранить») как файл hello_world.html. Убедитесь, что файл сохранен на сайте Dreamweaver.

Дополнительные сведения о сайте Dreamweaver см. в справке Dreamweaver.

Создание приложения Adobe AIR

1. Убедитесь, что страница hello_world.html открыта в окне «Документ» в Dreamweaver. (См. инструкции по ее созданию в предыдущем разделе.)
2. Выберите «Сайт» > «Настройки приложения Air».

   Большинство требуемых настроек в диалоговом окне «AIR — приложение и настройки» заполняются автоматически. Но требуется выбрать исходное содержимое (или начальную страницу) для приложения.
3. Нажмите кнопку «Обзор», рядом с пунктом «Исходное содержимое», чтобы перейти к странице hello_world.html и выбрать ее.

4 Рядом с пунктом «Цифровая подпись» нажмите кнопку «Задать».

Цифровая подпись позволяет гарантировать, что код приложения не был изменен или поврежден с момента его создания автором программного обеспечения. Такая подпись необходима для всех приложений Adobe AIR.

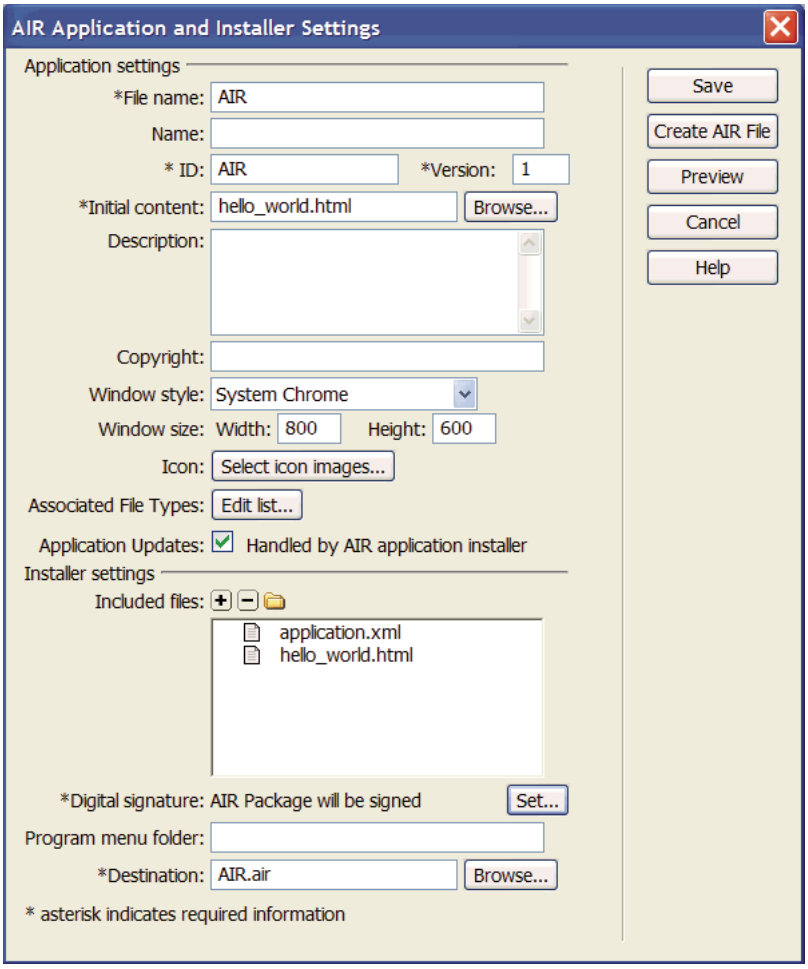
5 В диалоговом окне «Цифровая подпись» выберите «Подписать пакет AIR цифровым сертификатом», а затем нажмите кнопку «Создать». (Если доступ к цифровому сертификату уже есть, можно нажать кнопку «Обзор» чтобы выбрать его.)

6 Заполните обязательные поля в диалоговом окне «Самозаверяющий цифровой сертификат». Необходимо указать свое имя, ввести пароль и подтвердить его, а затем ввести имя для файла цифрового сертификата. Dreamweaver сохраняет цифровой сертификат в корневом каталоге сайта.

7 Нажмите «ОК», чтобы вернуться в диалоговое окно «Цифровая подпись».

8 В диалоговом окне «Цифровая подпись» введите пароль, указанный для цифрового сертификата, и нажмите «ОК».

Заполненное диалоговое окно «AIR — настройки приложения и установщика» может выглядеть следующим образом.



Дополнительные объяснения обо всех возможностях данного диалогового окна и порядке их изменения см. в разделе «Создание приложения AIR в Dreamweaver».

9 Нажмите кнопку «Создать файл AIR».

Dreamweaver создает файл приложения Adobe AIR и сохраняет его в корневую папку сайта. Dreamweaver также создает файл application.xml и сохраняет его в той же папке. Этот файл выполняет роль манифеста, определяя различные свойства приложения.

Установка приложения на настольном компьютере

Теперь, когда создан файл приложения, можно установить его на любом настольном компьютере.

1 Переместите файл приложения Adobe AIR с сайта Dreamweaver на персональный компьютер.

Этот шаг не обязателен. Если потребуется, приложение можно установить на свой компьютер непосредственно из каталога на сайте Dreamweaver.

2 Дважды щелкните исполняемый файл приложения (файл .air), чтобы установить это приложение.

Предварительный просмотр приложения Adobe AIR

В любое время можно выполнить предварительный просмотр страниц, которые затем станут частью приложений AIR. Поэтому нет необходимости упаковывать приложение для того, чтобы посмотреть, как оно будет выглядеть после установки.

1 Убедитесь, что страница hello_world.html открыта в окне «Документ» в Dreamweaver.

2 На панели инструментов «Документ» нажмите кнопку «Просмотр и отладка в браузере», а затем выберите «Просмотр в AIR».

Также можно нажать комбинацию клавиш Ctrl+Shift+F12 (Windows) или Cmd+Shift+F12 (Macintosh).

При выполнении предварительного просмотра данной страницы отображается то, что пользователи увидят на начальной странице приложения после его установки на настольный компьютер.

Глава 9. Создание первого HTML-приложения AIR с помощью комплекта AIR SDK

Чтобы быстро на практике продемонстрировать как работает Adobe® AIR®, используйте эти инструкции по созданию и упаковке простейшего HTML-приложения AIR «Hello World».

Для начала необходимо установить среду выполнения и настроить AIR SDK. Используются инструменты *AIR Debug Launcher* (ADL) и *AIR Developer Tool* (ADT), описанные в данном руководстве. ADL и ADT — это служебные программы, запускаемые из командной строки, которые можно найти в каталоге `bin` в пакете AIR SDK (см. раздел ««Установка AIR SDK» на странице 21»). Предполагается, что читатель данного руководства уже знаком с тем, как запускать программы из командной строки и знает как настроить необходимые переменные пути в операционной системе.

Примечание. *Если используется продукт Adobe® Dreamweaver®, ознакомьтесь с разделом ««Создание первого HTML-приложения AIR с помощью Dreamweaver» на странице 39».*

Создание файлов проекта

В каждом HTML-приложении AIR содержатся следующие два файла: файл дескриптора приложения, определяющий метаданные приложения, и страница HTML верхнего уровня. В дополнение к этим обязательным файлам данный проект содержит файл с кодом JavaScript, `AIRAliases.js`, который определяет подходящие псевдонимы-переменные для классов прикладного интерфейса программирования AIR.

1. Создайте каталог с именем `HelloWorld`, в котором будут содержаться файлы проекта.
2. Создайте файл XML с именем `HelloWorld-app.xml`.
3. Создайте файл HTML с именем `HelloWorld.html`.
4. Скопируйте файл `AIRAliases.js` из папки frameworks в AIR SDK в каталог проекта.

Создание файла дескриптора приложения AIR

Чтобы приступить к разработке приложения AIR, создайте XML-файл дескриптора приложения со следующей структурой:

```
<application>
    <id>…</id>
    <version>…</version>
    <filename>…</filename>
    <initialWindow>
        <content>…</content>
        <visible>…</visible>
        <width>…</width>
        <height>…</height>
    </initialWindow>
</application>
```

1 Откройте файл `HelloWorld-app.xml` для редактирования.

2 Добавьте корневой элемент `<application>`, включая атрибут пространства имен AIR:

<application xmlns="http://ns.adobe.com/air/application/2.0"> Последний сегмент пространства имен (2.0) указывает версию среды выполнения, которая требуется для приложения.

3 Добавьте элемент `<id>`:

<id>examples.html.HelloWorld</id> Идентификатор приложения однозначно идентифицирует приложение в сочетании с идентификатором издателя (который AIR извлекает из сертификата, используемого для подписи пакета приложения). Идентификатор приложения используется для установки, обращения к частному каталогу хранилища файловой системы приложения, доступа к частному зашифрованному хранилищу и взаимодействия между приложениями.

4 Добавьте элемент `<version>`:

<version>0.1</version> Помогает пользователям определить, какую версию приложения они устанавливают.

5 Добавьте элемент `<filename>`:

<filename>HelloWorld</filename> Имя, используемое для выполняемого файла приложения, каталога установки и других ссылок на приложение в операционной системе.

6 Добавьте элемент `<initialWindow>`, содержащий следующие дочерние элементы, чтобы указать свойства исходного окна приложения:

<content>HelloWorld.html</content> Указывает корневой HTML-файл приложения AIR для загрузки.

<visible>true</visible> Сразу делает окно видимым.

<width>400</width> Задает ширину окна (в пикселах).

<height>200</height> Задает высоту окна.

7 Сохраните файл. Завершенный файл дескриптора приложения должен выглядеть следующим образом.

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://ns.adobe.com/air/application/2.0">
    <id>examples.html.HelloWorld</id>
    <version>0.1</version>
    <filename>HelloWorld</filename>
    <initialWindow>
        <content>HelloWorld.html</content>
        <visible>true</visible>
        <width>400</width>
        <height>200</height>
    </initialWindow>
</application>
```

В этом примере устанавливается всего несколько возможных свойств приложения. Полный набор свойств приложения, которые позволяют указывать такие аспекты как Chrome окна, размер окна, прозрачность, каталог установки по умолчанию, связанные типы файлов и значки приложения, см. в разделе ««Задание свойств приложения AIR» на странице 71».

Создание HTML-страницы приложения

Теперь необходимо создать простую HTML-страницу, которая будет выполнять роль основного файла приложения AIR.

1 Откройте файл `HelloWorld.html` для редактирования. Добавьте следующий код HTML:

```
<html>
<head>
    <title>Hello World</title>
</head>
<body onLoad="appLoad()">
    <h1>Hello World</h1>
</body>
</html>
```

2 В разделе `<head>` HTML, импортируйте файл `AIRAliases.js`:

```
<script src="AIRAliases.js" type="text/javascript"></script>
```

AIR определяет свойство с именем `runtime` для объекта окна HTML. Свойство runtime обеспечивает доступ к встроенным классам AIR, используя полностью уточненное имя пакета для класса. Например, чтобы создать объект файла AIR, можно добавить следующую инструкцию в JavaScript:

```
var textFile = new runtime.flash.filesystem.File("app:/textfile.txt");
```

Файл `AIRAliases.js` определяет подходящие псевдонимы для наиболее полезных прикладных программных интерфейсов AIR. Используя `AIRAliases.js`, можно укоротить ссылку на класс File следующим образом:

```
var textFile = new air.File("app:/textfile.txt");
```

3 Ниже приведен тег скрипта AIRAliases, добавьте другой тег скрипта, содержащий функцию JavaScript для обработки события `onLoad`:

```
<script type="text/javascript">
function appLoad(){
    air.trace("Hello World");
}
</script>
```

Функция `appLoad()` просто вызывает функцию `air.trace()`. При запуске приложения с помощью ADL сообщение трассировки вводится на консоль управления. Сообщения трассировки могут быть очень полезны для отладки.

4 Сохраните файл.

Файл `HelloWorld.html` должен выглядеть следующим образом:

```
<html>
<head>
    <title>Hello World</title>
    <script type="text/javascript" src="AIRAliases.js"></script>
    <script type="text/javascript">
        function appLoad(){
            air.trace("Hello World");
        }
    </script>
</head>
<body onLoad="appLoad()">
    <h1>Hello World</h1>
</body>
</html>
```

Проверка приложения

Чтобы запустить и проверить приложение из командной строки, используйте утилиту AIR Debug Launcher (ADL). Исполняемый файл ADL можно найти в каталоге `bin` пакета AIR SDK. Если настройка AIR SDK еще не выполнена, см. раздел ««Установка AIR SDK» на странице 21».

1 Откройте консоль управления или оболочку ОС. Перейдите в каталог, который создан для этого проекта.

2 Выполните следующую команду:

```
adl HelloWorld-app.xml
```

Открывается окно AIR, в котором отображается приложение. Также в окне консоли отображается сообщение, полученное в результате вызова `air.trace()`.

Дополнительные сведения см. в разделе ««Задание свойств приложения AIR» на странице 71».

Создание файла установки AIR

Если приложение успешно запущено, можно использовать утилиту ADT, чтобы упаковать приложение в файл установки AIR. Файл установки AIR — это файл архива, в котором содержаться все файлы приложения, которые можно передавать другим пользователям. Перед установкой такого упакованного файла AIR необходимо установить Adobe AIR.

Чтобы обеспечить защиту приложения, все файлы установки AIR должны содержать цифровую подпись. С помощью ADT или другого инструмента генерации сертификатов можно создать простой, самозаверяющий сертификат, используемый в целях разработки. Также можно приобрести коммерческий сертификат для подписывания кодов, выпущенный специальными органами сертификации, например компаниями VeriSign или Thawte. Если пользователь устанавливает файл AIR с самозаверяющим сертификатом, то в ходе процесса установки показывается «неизвестный» издатель. Это объясняется тем, что самозаверяющие сертификаты гарантируют лишь то, что файл AIR не изменялся с момента своего создания. Ничто не сможет помешать злоумышленнику сделать файл AIR с самозаверяющим сертификатом и представить его как ваше приложение. Поэтому в выпускаемых для широкого использования файлах AIR рекомендуется использовать обеспечивающие возможность проверки коммерческие сертификаты. Краткие сведения о проблемах системы защиты AIR см. в разделе Система защиты AIR (для разработчиков ActionScript) или Система защиты AIR (для разработчиков HTML).

Создание самозаверяющих подписывающих сертификатов и пары ключей

❖ Из командной строки выполните следующую команду (выполняемый файл ADT находится в каталоге `bin` пакета AIR SDK):

```
adt -certificate -cn SelfSigned 1024-RSA sampleCert.pfx samplePassword
```

ADT создает файл хранения ключа с именем *sampleCert.pfx*, содержащий сертификат и связанный личный ключ.

В этом примере используется минимальный набор атрибутов, которые могут быть установлены для сертификата. Можно использовать любые значения для параметров, обозначенных *курсивом*. Ключ должен иметь тип *1024-RSA* или *2048-RSA* (см. раздел ««Цифровая подпись файлов AIR» на странице 96»).

Создание файла установки AIR

❖ Из командной строки выполните следующую команду (вводится одной строкой):

```
adt -package -storetype pkcs12 -keystore sampleCert.pfx HelloWorld.air
HelloWorld-app.xml HelloWorld.html AIRAliases.js
```

Будет выдано предложение ввести пароль для файла ключей.

Аргумент HelloWorld.air — это файл AIR, создаваемый ADT. HelloWorld-app.xml — это файл дескриптора приложения. Последующими аргументами являются файлы, используемые приложением. В этом примере используется только два файла, но можно включить в проект любое число файлов и каталогов.

После создания пакета AIR можно установить и выполнить приложение, дважды щелкнув этот упакованный файл. Также можно ввести имя файла AIR, используя команду в оболочке ОС или консоль управления.

Следующие шаги

В приложении AIR коды HTML и JavaScript обычно выполняются так же, как и в стандартном веб-обозревателе. (Действительно, AIR использует тот же механизм визуализации WebKit, который применяется веб-обозревателем Safari.) Однако существует несколько важных отличий, которые необходимо учитывать при разработке HTML-приложения в AIR. Дополнительные сведения об этих различиях, а также другие полезные сведения см. в разделе Программирование HTML и JavaScript.

Глава 10. Создание приложения AIR с помощью инструментов командной строки

Инструменты для работы с командной строкой Adobe® AIR® позволяют откомпилировать, протестировать, собрать в пакет и подписать приложения Adobe AIR. Инструменты для работы с командной строкой AIR входят в состав продуктов Adobe® Flex™ и AIR SDK (http://www.adobe.com/go/learn_air_download_AIRSDK_ru).

AIR и Flex SDK предлагают следующие инструменты для работы с командной строкой, позволяющие разрабатывать приложения AIR.

Компиляция Чтобы откомпилировать файлы источника Flex MXML и ActionScript, используйте компиляторы *amxmlc* или *acompc*. Для проектов Flash Professional компиляция проекта выполняется при публикации видеоролика. HTML-приложения AIR не требуют компиляции.

Примечание. *Инструменты amxmlc и acompc включены только во Flex SDK, но не входят в AIR SDK.*

Тестирование и отладка Используйте программу *AIR Debug Launcher (ADL)* для тестирования и отладки приложений AIR. Используя ADL, можно запускать проектируемые приложения AIR, без их упаковки и установки. ADL распечатывает выходные данные с сообщениями трассировки и ошибками в консоли командной строки. Также ADL можно использовать для подключения к инструменту *Flash Debugger (FDB)* для отладки более сложных проблем.

Примечание. *Инструмент FDB поставляется только с Flex SDK, но не входит в AIR SDK. Пакет Flex SDK доступен для загрузки с сервера http://opensource.adobe.com.*

Упаковка Используйте инструмент *AIR Developer Tool (ADT)* для упаковки готового приложения AIR в файл программы установки AIR (.air), который можно установить в любой поддерживаемой вычислительной системе, а также в файл стандартной программы установки, который можно установить только на компьютерах такого же типа, что и компьютер на котором запущен ADT.

Подписывание Используйте ADT для подписывания приложения AIR с помощью сертификата публикации. Можно подписать приложение при создании программного пакета. Также можно разделить создание пакета на два шага (что особенно полезно, если доступ к сертификатам и частным ключам, используемым для подписи кода, строго контролируется). Все приложения AIR должны быть подписаны.

Примечание. *Большинство интегрированных сред разработки, включая Adobe Flash Builder, Adobe Flash Professional и Aptana Studio, могут компилировать, отлаживать и создавать пакеты приложений AIR. Если применяется подобная среда разработки, то использование инструментов командной строки для выполнения этих общих задач как правило не требуется. Но, возможно, что инструменты командной строки все равно потребуются для реализации функций, не поддерживаемых интегрированной средой разработки. Кроме того, можно также использовать инструменты командной строки в качестве составной части процесса автоматизированного создания.*

Компиляция исходных файлов MXML и ActionScript для AIR

Для компиляции файлов Adobe® ActionScript® 3.0 и MXML для приложения AIR можно использовать компилятор командной строки MXML (amxmlc). (Компиляция для HTML-приложений не требуется. Для компиляции SWF-файла во Flash Professional просто опубликуйте видеоролик в SWF-файл.)

Базовый шаблон командной строки для использования инструмента amxmlc имеет следующий вид:

```
amxmlc [compiler options] -- MyAIRApp.mxml
```

где *[compiler options]* задает параметры командной строки, используемые для компиляции приложения AIR.

Команда amxmlc вызывает стандартный компилятор Flex mxmlc с дополнительным параметром `+configname=air`. Этот параметр сообщает компилятору, что нужно использовать файл air-config.xml, а не flex-config.xml. В остальном amxmlc идентичен mxmlc. Компилятор mxmlc и формат файла конфигурации описаны в руководстве Создание и развертывание приложений Flex 3 в библиотеке документации по Flex 3.

Компилятор загружает файл конфигурации air-config.xml, в котором задаются библиотеки AIR и Flex, в большинстве случаев необходимые для компиляции приложения AIR. Также для переопределения или дополнения глобальных параметров конфигурации можно воспользоваться локальным файлом конфигурации на уровне проекта. Как правило, проще всего создать файл локальной конфигурации, скопировав и отредактировав его глобальную версию. Локальный файл можно загрузить с помощью `-load-config`:

-load-config=project-config.xml переопределяет глобальные параметры.

-load-config+=project-config.xml добавляет значения к тем глобальным параметрам, у которых их более одного, таким как `-library-path`. Глобальные параметры, принимающие только одно значение, переопределяются.

Если для файла локальной конфигурации используется специальное соглашение об именах, компилятор amxmlc автоматически загружает локальный файл. Например, если основной MXML-файл — `RunningMan.mxml`, то файл локальной конфигурации будет называться `RunningMan-config.xml`. Теперь для компиляции приложения нужно ввести всего лишь следующее:

```
amxmlc RunningMan.mxml
```

`RunningMan-config.xml` загружается автоматически, потому что его имя соответствует имени скомпилированного MXML-файла.

Примеры amxmlc

Ниже приводятся примеры использования компилятора amxmlc. (В приложении компилируются только ресурсы ActionScript и MXML).

Компиляция MXML-файла AIR:

```
amxmlc myApp.mxml
```

Компиляция и задание имени вывода:

```
amxmlc –output anApp.swf -- myApp.mxml
```

Компиляция MXML-файла ActionScript:

```
amxmlc myApp.as
```

Указание файла конфигурации компилятора:

```
amxmlc –load-config config.xml -- myApp.mxml
```

Добавление параметров из другого файла конфигурации:

```
amxmlc –load-config+=moreConfig.xml -- myApp.mxml
```

Добавление библиотек в командную строку (помимо библиотек, уже содержащихся в файле конфигурации):

```
amxmlc –library-path+=/libs/libOne.swc,/libs/libTwo.swc  -- myApp.mxml
```

Компиляция MXML-файла AIR без файла конфигурации (Win):

```
mxmlc -library-path [AIR SDK]/frameworks/libs/air/airframework.swc, ^
[AIR SDK]/frameworks/libs/air/airframework.swc, ^
-library-path [Flex 3 SDK]/frameworks/libs/framework.swc ^
-- myApp.mxml
```

Компиляция MXML-файла AIR без файла конфигурации (Mac OS X или Linux):

```
mxmlc -library-path [AIR SDK]/frameworks/libs/air/airframework.swc, \
[AIR SDK]/frameworks/libs/air/airframework.swc, \
-library-path [Flex 3 SDK]/frameworks/libs/framework.swc \
-- myApp.mxml
```

Компиляция MXML-файла AIR для использования библиотеки в общем доступе с средой выполнения:

```
amxmlc -external-library-path+=../lib/myLib.swc -runtime-shared-libraries=myrsl.swf --
myApp.mxml
```

Компиляция из Java (путь к классу содержит `mxmlc.jar`):

```
java flex2.tools.Compiler +flexlib [Flex SDK 3]/frameworks +configname=air [additional
compiler options] -- myApp.mxml
```

Параметр flexlib определяет расположение каталога frameworks комплекта Flex SDK, благодаря чему компилятор находит файл flex_config.xml.

Компиляция из Java (без задания пути к классу):

```
java -jar [Flex SDK 2]/lib/mxmlc.jar +flexlib [Flex SDK 3]/frameworks +configname=air
[additional compiler options] -- myApp.mxml
```

Порядок вызова компилятора, используя Apache Ant (пример использует задачу Java для запуска mxmlc.jar):

```
<property name="SDK_HOME" value="C:/Flex3SDK"/>
<property name="MAIN_SOURCE_FILE" value="src/myApp.mxml"/>
<property name="DEBUG" value="true"/>
<target name="compile">
    <java jar="${MXMLC.JAR}" fork="true" failonerror="true">
        <arg value="-debug=${DEBUG}"/>
        <arg value="+flexlib=${SDK_HOME}/frameworks"/>
        <arg value="+configname=air"/>
        <arg value="-file-specs=${MAIN_SOURCE_FILE}"/>
    </java>
</target>
```

Компиляция компонента AIR или библиотеки кодов (Flex)

Используйте компилятор acompc для компиляции библиотек AIR и независимых компонентов. Компилятор компонента acompc во многом похож на amxmlc, но есть и ряд отличий:

- Необходимо указать, какие классы в коде необходимо включить в библиотеку или компонент.
- Команда acompc не ищет файл локальной конфигурации автоматически. Для использования файла конфигурации проекта необходимо задать параметр –load-config.

Команда acompc вызывает стандартный для Flex компилятор компонента compc, но загружает его параметры конфигурации из файла `air-config.xml` (не из `flex-config.xml`).

Файл конфигурации компилятора компонента

Используйте файл локальной конфигурации, чтобы не вводить вручную (и, возможно, с ошибками) в командную строку исходный путь и имена классов. Добавьте параметр -load-config к командной строке acompc, чтобы загрузить файл локальной конфигурации.

В примере ниже показана конфигурация для создания библиотеки с помощью двух классов: ParticleManager и Particle, оба включены в пакет `com.adobe.samples.particles`. Файлы классов расположены в папке `source/com/adobe/samples/particles`.

```
<flex-config>
    <compiler>
        <source-path>
            <path-element>source</path-element>
        </source-path>
    </compiler>
    <include-classes>
        <class>com.adobe.samples.particles.ParticleManager</class>
        <class>com.adobe.samples.particles.Particle</class>
    </include-classes>
</flex-config>
```

Для компиляции библиотеки с помощью файла конфигурации `ParticleLib-config.xml` введите следующее:

```
acompc -load-config ParticleLib-config.xml -output ParticleLib.swc
```

Для выполнения того же действия целиком из командной строки введите следующее:

```
acompc -source-path source -include-classes com.adobe.samples.particles.Particle
com.adobe.samples.particles.ParticleManager -output ParticleLib.swc
```

(Введите всю команду в одну строку или воспользуйтесь символом продолжения строки).

Примеры acompc

Предполагается, что используется файл конфигурации `myLib-config.xml`.

Компиляция компонента или библиотеки AIR:

```
acompc -load-config myLib-config.xml -output lib/myLib.swc
```

Компиляция библиотеки в совместном доступе с средой выполнения:

```
acompc -load-config myLib-config.xml -directory -output lib
```

(Перед тем как вызывать команду, убедитесь, что папка lib существует и пуста).

Использование средства AIR Debug Launcher (ADL)

Средство AIR Debug Launcher (ADL) рекомендуется использовать для запуска SWF- и HTML-приложений во время разработки. С помощью ADL можно запустить приложение без предварительной упаковки и установки. По умолчанию, ADL использует среду выполнения, включенную в комплект SDK, т. е. для работы с ADL не потребуется отдельно устанавливать среду выполнения.

ADL печатает трассировочные инструкции и ошибки рабочей среды в виде стандартного вывода, но не поддерживает контрольные точки и иные функции отладки. Можно использовать программу отладки Flash Debugger (или интегрированные среды разработки, такие как Flash Builder или Aptana Studio) для решения сложных проблем с отладкой.

Запуск приложения с помощью ADL

Для выполнения приложения с помощью ADL, используйте следующий шаблон:

```
adl application.xml
```

Здесь application.xml является файлом дескриптора приложения.

Полный синтаксис для ADL имеет следующий вид:

```
adl [-runtime runtime-directory] [-pubid publisher-id] [-nodebug] [-profile profileName]
application.xml [root-directory] [-- arguments]
```

-runtime runtime-directory задает каталог, содержащий используемую среду выполнения. Если каталог среды выполнения не указан отдельно, используется каталог из того же комплекта SDK, что и для ADL. Если переместить утилиту ADL из папки ее SDK, необходимо будет задать каталог среды выполнения. В Windows и Linux необходимо задать каталог, содержащий каталог `Adobe AIR`. В Mac OS X задайте каталог, содержащий `Adobe AIR.framework`.

-pubid publisher-id присваивает определенное значение в качестве ID издателя приложения AIR для данного запуска. Указание временного ID издателя позволяет протестировать функции приложения AIR (такие как связь по локальной сети), использующие ID издателя для уникального определения приложения. Начиная с версии AIR 1.5.3, можно также указать идентификатор издателя в файле дескриптора приложения (и не использовать этот параметр).

Примечание. *Начиная с версии AIR 1.5.3, идентификатор издателя больше автоматически не вычисляется и не назначается приложению AIR. Идентификатор издателя можно указать при создании обновления существующего приложения AIR, однако для новых приложений идентификатор издателя не требуется и не нужно указывать его.*

-nodebug отключает поддержку отладки. В этом случае процесс приложения не сможет подключиться к программе отладки Flash, а диалоги, связанные с необработанными исключениями, будут заблокированы. (Несмотря на это трассировочные инструкции по-прежнему печатаются в окне консоли.) Отключение отладки ускоряет работу приложения и более точно эмулирует режим работы установленного приложения.

-profile profileName — ADL выполняет отладку приложения с помощью указанного профиля. *`profileName`* может принимать значения `desktop`, `extendedDesktop`, `mobileDevice` и `extendedMobileDevice`. Дополнительные сведения см. в разделах ««Ограничение профилей целевого приложения» на странице 77» и ««Профили приложения» на странице 83».

application.xml — это файл дескриптора приложения. См. раздел ««Задание свойств приложения AIR» на странице 71». Дескриптор приложения — это единственный параметр, необходимый ADL и, в большинстве случаев, единственный требуемый параметр.

root-directory задает корневой каталог запускаемого приложения. Если он не задан, используется каталог, в котором находится файл дескриптора.

-- arguments любые строки символов, стоящие после «--», передаются приложению в качестве аргументов командной строки.

Примечание. *При запуске уже работающего приложения AIR новый экземпляр приложения не запускается. Вместо этого работающему экземпляру отправляется событие* `invoke`.

Печать трассировочных инструкций

Для печати трассировочных инструкций в консоли, используемой для запуска ADL, добавьте трассировочные инструкции в код с помощью функции `trace`.

Пример JavaScript:

```
//ActionScript
trace("debug message");
```

Пример ActionScript:

```
//JavaScript
air.trace("debug message");
```

В коде JavaScript можно использовать функции `alert()` и `confirm()` для отображения сообщений отладки из своего приложения. Кроме того, в консоль печатаются номера строк с синтаксическими ошибками и все необработанные исключения JavaScript.

Примечание. *Чтобы применить префикс air, показанный в примере JavaScript, необходимо импортировать на страницу файл AIRAliases.js. Это файл находится внутри каталога frameworks в пакете AIR SDK.*

Примеры ADL

Запуск приложения в текущем каталоге:

```
adl myApp-app.xml
```

Запуск приложения в подкаталоге текущего каталога:

```
adl source/myApp-app.xml release
```

Запуск приложения и передача двух аргументов командной строки: «tick» и «tock»:

```
adl myApp-app.xml -- tick tock
```

Запуск приложения в определенной среде выполнения:

```
adl -runtime /AIRSDK/runtime myApp-app.xml
```

Запуск приложения без поддержки отладки:

```
adl myApp-app.xml -nodebug
```

Запустите приложение с помощью Apache Ant:

```
<property name="SDK_HOME" value="C:/AIRSDK"/>
<property name="ADL" value="${SDK_HOME}/bin/adl.exe"/>
<property name="APP_DESCRIPTOR" value="$src/myApp-app.xml"/>

<target name="test">
    <exec executable="${ADL}">
        <arg value="${APP_DESCRIPTOR}"/>
    </exec>
</target>
```

Подключение к отладчику Flash Debugger (FDB)

Для отладки приложений AIR с помощью Flash Debugger запустите сеанс FDB и затем запустите приложение, используя ADL.

Примечание. *В SWF-приложениях AIR файлы источника ActionScript должны быть скомпилированы с флагом* `-debug`*. (Во Flash Professional, установите флажок «Разрешить отладку» в диалоговом окне «Параметры публикации».)*

1. Запустите FDB. Программу FDB можно найти в каталоге `bin` пакета Flex SDK.

   В консоли выводится запрос FDB: `<fdb>`

2. Выполните команду `run`: `<fdb>run [Enter]`
3. Запустите отладочную версию приложения в другой командной строке или оболочке:

   ```
   adl myApp.xml
   ```

4. С помощью команд FDB задайте контрольные точки.
5. Введите: `continue [Enter]`

Если приложение AIR выполнено в виде SWF-файла, то отладчик управляет только выполнением кода ActionScript. Если приложение AIR выполнено в виде HTML-файла, то отладчик управляет только выполнением кода JavaScript.

Чтобы выполнить ADL без подключения к отладчику, включите параметр `-nodebug`:

```
adl myApp.xml -nodebug
```

Чтобы получить краткие сведения о командах FDB, выполните команду `help`:

```
<fdb>help [Enter]
```

Дополнительные сведения о командах FDB см. на странице Использование команд отладчика из командной строки в документации по Adobe Flex.

Завершение работы ADL и коды ошибок

Ниже перечислены коды выхода, выводимые ADL.

Код выхода	Описание
0	Успешный запуск. ADL завершает работу после выхода из приложения AIR.
1	Успешный вызов уже работающего приложения AIR. ADL завершает работу немедленно.
2	Ошибка использования. ADL переданы неверные аргументы.
3	Не удается найти среду выполнения.

Код выхода	Описание
4	Не удается запустить среду выполнения. Это часто происходит из-за того, что версия уровня исправлений, указанная в приложении, не совпадает с версией или уровнем исправлений среды выполнения.
5	Произошла ошибка, причины которой неизвестны.
6	Не удается найти файл дескриптора приложения.
7	Недопустимое содержимое дескриптора приложения. Обычно эта ошибка говорит о неправильной сборке XML-файла.
8	Не удается найти основной файл содержимого приложения (заданный элементом <content> файла дескриптора).
9	Основной файл содержимого приложения не является допустимым SWF- или HTML-файлом.

Упаковка установочного файла AIR с помощью AIR Developer Tool (ADT)

Установочный файл AIR создается как для SWF-, так и для HTML-приложения AIR с помощью средства AIR Developer Tool (ADT).

ADT — это Java-приложение, которое можно запускать из командной строки или средства сборки, например Ant. В комплект SDK входят сценарии командных строк, автоматизирующие выполнение программ Java.

Если для создания приложения используется Flash Builder, можно также применять мастер экспорта Flash Builder, для создания файла пакета AIR. Сведения см. в руководстве Разработка приложений AIR в программе Flash Builder.

В среде Flash Professional CS5 используйте команду «Файл» > «Опубликовать» для создания пакета AIR. Дополнительные сведения см. в разделе Публикация для Adobe AIR руководства «Использование Flash (CS5)». В среде Adobe Flash CS3 или CS4 Professional для создания пакета AIR в меню «Команды» выберите команду «Создать файл AIR». Дополнительные сведения см. в разделе Публикация для Adobe AIR руководства «Использование Flash (CS4)».

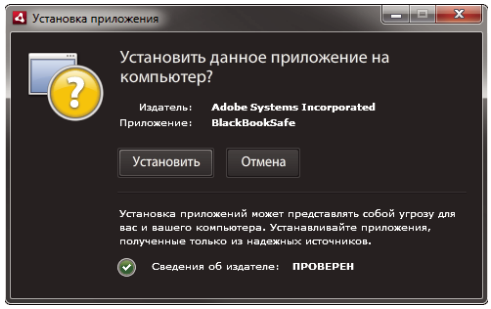
Если для создания приложения используется расширение Adobe® AIR® для Dreamweaver®, для подготовки пакета можно также использовать команду «Создать файл AIR». Эта команда доступна в диалоговом окне «AIR — настройки приложения и установщика».

Упаковка установочного файла AIR

Каждое приложение AIR должно как минимум иметь файл дескриптора и основной SWF- или HTML-файл. Прочие активы, устанавливаемые вместе с приложением, необходимо также упаковать в AIR-файл.

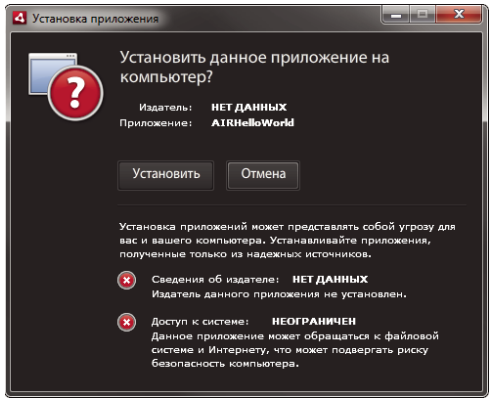
Все установочные файлы AIR необходимо подписывать цифровым сертификатом. Программа установки AIR использует подпись для гарантии того, что файл приложения не изменялся с момента подписания. Можно использовать сертификат для подписания кода, выданный уполномоченным центром сертификации, или применять самозаверяющий сертификат.

Сертификат, выданный сертифицирующим органом, вызовет у пользователей большее доверие к вам как к издателю. Во время установки отобразится диалоговое окно со сведениями о том, что удостоверение издателя проверено сертифицирующим органом.



Диалоговое окно подтверждения для приложения, подписанного доверенным сертификатом

При использовании самоподписанного сертификата пользователи не смогут проверить ваше удостоверение издателя. Кроме этого, пользователь не сможет быть уверен в том, что содержимое пакета не было изменено. (Исходный установочный файл может быть подменен до того, как конечный пользователь откроет его.) Во время установки отобразится диалоговое окно со сведениями о том, что удостоверение издателя не может быть проверено. Устанавливая такие приложения пользователи подвергают систему безопасности большему риску:



Диалоговое окно подтверждения для приложения с самоподписанным сертификатом

Файл AIR можно упаковать и подписать в один этап с помощью команды `-package` средства ADT. Также можно создать промежуточный, неподписанный пакет с помощью команды `-prepare` и отдельно подписать промежуточный пакет с помощью команды `-sign`.

Примечание. *Java 1.5 и более поздние версии не позволяют использовать в паролях для защиты файлов сертификатов PKCS12 нестандартные символы ASCII. При создании или экспорте файла сертификата кода используйте в пароле только стандартные символы ASCII.*

При подписании установочного пакета ADT автоматически связывается с сервером органа, поставившего временную отметку, для ее проверки. Информация о временной отметке включена в файл AIR. Файл AIR, содержащий проверенную временную отметку, в будущем может быть установлен в любой момент. Если средство ADT не может связаться с сервером временных отметок, упаковка не производится. Эту функцию можно отменить, но без временной отметки приложение AIR нельзя будет установить после того, как истечет срок действия сертификата подписи.

При создании пакета для обновления существующего приложения AIR он должен быть подписан тем же сертификатом, что и исходное приложение. Если исходный сертификат за последние 180 дней обновлялся или истекал, или при необходимости замены сертификата на новый можно использовать подпись переноса. При использовании подписи переноса AIR-файл приложения подписывается как старым, так и новым сертификатом. Воспользуйтесь командой `-migrate`, чтобы применить подпись переноса, как это описано в разделе «Подпись файла AIR для изменения сертификата приложения» на странице 68.

Важная информация. *По окончании срока действия сертификата и по прошествии 180-дневного льготного периода применить подпись переноса невозможно. Если не выполнить перенос сертификата, пользователи должны будут удалить текущую версию приложения перед установкой новой. Льготный период применяется только к приложениям, для которых в пространстве имен дескриптора приложения указана версия AIR 1.5.3 или более поздняя версия. Обратите внимание, что льготный период не предусмотрен в более ранних версиях среды выполнения AIR.*

В версиях, предшествующих AIR 1.1, поддержка подписей переноса отсутствует. Для применения подписи переноса необходимо создать пакет приложения с помощью SDK версии 1.1 или более поздней версии.

Приложения, развертываемые с помощью файлов AIR, называются приложениями с профилем настольного компьютера. Нельзя использовать ADT для упаковки исходного установщика для приложения AIR, если файл дескриптора приложения не поддерживает профиль настольного компьютера. Этот профиль можно запретить путем использования элемента supportedProfiles в файле дескриптора приложения. См. раздел ««Ограничение профилей целевого приложения» на странице 77».

Примечание. *Параметры в файле дескриптора определяют идентификационные данные приложения AIR и путь установки по умолчанию. См. раздел ««Структура файла дескриптора приложения» на странице 71».*

Идентификаторы издателя

Начиная с версии AIR 1.5.3, идентификатор издателя не используется. В новых приложениях (изначально опубликованных с помощью AIR 1.5.3 или более поздней версии) не требуется и не должен использоваться идентификатор издателя.

При обновлении приложений, опубликованных с помощью более ранних версий AIR, необходимо указать исходный идентификатор издателя в файле дескриптора приложения. В противном случае установленная версия приложения и его обновленная версия будут считаться различными приложениями. Если используется другой идентификатор издателя (или если он не используется), пользователи должны будут удалить текущую версию приложения перед установкой новой.

Чтобы определить исходный идентификатор издателя, найдите файл `publisherid` в подкаталоге META-INF/AIR, в котором установлено исходное приложение. Строка в этом файле является идентификатором издателя. Чтобы указать идентификатор издателя вручную, в дескрипторе приложения должна быть указана среда выполнения AIR 1.5.3 (или более поздней версии) в объявлении пространства имен файла дескриптора приложения.

Для приложений, опубликованных с помощью версии AIR, предшествующей 1.5.3, или опубликованных с помощью AIR 1.5.3 SDK, при указании в пространстве имен дескриптора приложения более ранней версии AIR — идентификатор издателя вычисляется на основе сертификата подписи. Этот идентификатор используется совместно с идентификатором приложения для определения подлинности приложения. При наличии идентификатора издателя он используется в следующих целях:

- Проверка того, является ли устанавливаемый файл AIR обновлением или новым приложением
- как часть ключа шифрования для зашифрованного локального хранилища;
- как часть пути к каталогу хранения приложения;
- как часть строки для локальных соединений;

- как часть строки учетных данных, используемой для вызова приложения в API-интерфейсе обозревателя AIR;
- как часть идентификатора OSID (используется при создании пользовательских программ установки/удаления).

В версиях, предшествующих AIR 1.5.3, идентификатор издателя приложения мог изменяться, если обновление для приложения было подписано подписью переноса с использованием нового или обновленного сертификата. При изменении идентификатора издателя также изменяется поведение функций AIR, зависящих от идентификатора. Например, данные в существующем зашифрованном локальном хранилище становятся недоступными, и в любых экземплярах Flash или AIR, создающих локальное соединение с приложением, в строке подключения должен использоваться новый идентификатор.

В AIR 1.5.3 или более поздней версии идентификатор разработчика не связан с сертификатом подписи и присваивается только в том случае, если в дескрипторе приложения содержится тег publisherID. Приложение невозможно обновить, если указанный в пакете обновления AIR идентификатор издателя не совпадает с текущим идентификатором издателя.

Упаковка и подпись файла AIR в один этап

❖ Используйте команду `-package` со следующим синтаксисом (в единой командной строке):

```
adt -package SIGNING_OPTIONS air_file app_xml [file_or_dir | -C dir file_or_dir | -e file dir ...] ...
```

SIGNING_OPTIONS. Параметры подписи определяют хранилище ключей, в котором содержатся закрытый ключ и сертификат подписи файла AIR. Для подписи приложения AIR самозаверяющим сертификатом, созданным ADT, используйте следующие параметры:

```
-storetype pkcs12 -keystore certificate.p12
```

В этом примере *certificate.p12* — это имя файла-хранилища ключей. (Так как пароль не включается в командную строку, ADT запрашивает его.) Способы подписи описаны в разделе «Способы подписи командной строки с помощью ADT» на странице 63.

air_file — имя создаваемого файла AIR.

app_xml — путь к файлу дескриптора приложения. Можно задать абсолютный путь или путь относительно текущей директории. (Файл дескриптора приложения переименовывается в файле AIR в application.xml.)

file_or_dir — файлы и каталоги для упаковки в файл AIR. Можно задать любое количество файлов и каталогов, разделив их пробелом. Если указать каталог, то все файлы и подкаталоги в нем, кроме скрытых, будут добавлены в пакет. (Кроме того, если задан файл дескриптора приложения — напрямую, с помощью знака подстановки или подстановки каталогов, — он игнорируется и не добавляется в пакет повторно.) Заданные файлы и каталоги должны быть в текущем каталоге или одном из его подкаталогов. Используйте параметр `-C` для изменения текущего каталога.

Важная информация. *Знаки подстановки нельзя использовать в аргументах* `file_or_dir`*, следующих за* `-C`*. (Перед передачей аргументов ADT командные оболочки подставляют значения вместо всех знаков подстановки, поэтому ADT будет искать файлы не в том месте.) Тем не менее для обозначения текущего каталога по-прежнему можно использовать знак точки «.». Например, команда* `"-C assets."` *копирует все из каталога assets, включая подкаталоги, в корневой каталог пакета приложения.*

`-C dir` изменяет рабочий каталог на *dir*, прежде чем обрабатывать последующие файлы и каталоги, добавленные в пакет приложения. Файлы или каталоги добавляются в корневой каталог пакета приложения. Параметр `-C` можно использовать многократно для добавления файлов из разных точек файловой системы. Если для `dir` задан относительный путь, за основу всегда берется рабочий каталог.

В процессе обработки файлов и каталогов пакета средством ADT сохраняются относительные пути от текущего каталога к целевым файлам. При установке пакета эти пути подставляются в структуру каталогов приложения. Таким образом, параметр `-C release/bin lib/feature.swf` поместит файл `release/bin/lib/feature.swf` в подкаталог `lib` корневой папки приложения.

`-e file dir` помещает указанный файл в определенный каталог пакета.

Примечание. *Элемент* `<content>` *файла дескриптора приложения должен указывать итоговое расположение основного файла приложения в дереве каталогов пакета приложения.*

Также можно упаковать и распространить приложение AIR с помощью собственного установщика (например, EXE-файл в ОС Windows или DMG-файл в ОС Mac OS). Дополнительные сведения см. в разделе ««Упаковка приложения AIR в собственной программе установки» на странице 65».

Примеры использования команды ADT -package

Упаковка файлов приложения в текущей папке для приложения AIR на основе SWF:

```
adt –package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf components.swc
```

Упаковка файлов приложения в текущей папке для приложения AIR на основе HTML:

```
adt –package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.html AIRAliases.js
image.gif
```

Упакуйте все файлы и подкаталоги в текущем рабочем каталоге:

```
adt –package -storetype pkcs12 -keystore ../cert.p12 myApp.air myApp.xml .
```

Примечание. *Файл-хранилище ключей содержит закрытый ключ, которым подписано приложение. Ни в коем случае не упаковывайте сертификат подписи в пакет AIR! Если в командах ADT используются знаки подстановки, поместите файл-хранилище ключей в другую папку, чтобы он не попал в пакет. В этом примере файл-хранилище ключей cert.p12 находится в родительском каталоге.*

Включайте в пакет только основные файлы и подкаталог images:

```
adt –package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf images
```

Включайте в пакет HTML-приложение и все файлы в подкаталогах HTML, scripts и images:

```
adt –package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml index.html AIRALiases.js
html scripts images
```

Включайте в пакет файл application.xml и основной SWF-файл в рабочем каталоге (release/bin):

```
adt –package -storetype pkcs12 -keystore cert.p12 myApp.air release/bin/myApp.xml –C
release/bin myApp.swf
```

Включайте в пакет ресурсы более чем из одного места в файловой системе сборки. В этом примере ресурсы приложения до упаковки находятся в следующих папках:

```
/devRoot
    /myApp
        /release
            /bin
                myApp.xml
                myApp.swf or myApp.html
    /artwork
        /myApp
            /images
                image-1.png
                ...
                image-n.png
    /libraries
        /release
            /libs
                lib-1.swf
                lib-2.swf
                lib-a.js
                AIRAliases.js
```

Выполнение команды ADT из каталога /devRoot/myApp:

```
adt –package -storetype pkcs12 -keystore cert.p12 myApp.air release/bin/myApp.xml
    –C release/bin myApp.swf (or myApp.html)
    –C ../artwork/myApp images
    –C ../libraries/release libs
```

создает следующую структуру пакета:

```
/myAppRoot
    /META-INF
        /AIR
            application.xml
            hash
    myApp.swf
    mimetype
    /images
        image-1.png
        ...
        image-n.png
    /libs
        lib-1.swf
        lib-2.swf
        lib-a.js
        AIRAliases.js
```

Запустите ADT в качестве программы Java для простого приложения на основе SWF (без указания пути к классам):

```
java –jar {AIRSDK}/lib/ADT.jar –package -storetype pkcs12 -keystore cert.p12 myApp.air
myApp.xml myApp.swf
```

Запустите ADT в качестве программы Java для простого приложения на основе HTML (без указания пути к классам):

```
java –jar {AIRSDK}/lib/ADT.jar –package -storetype pkcs12 -keystore cert.p12 myApp.air
myApp.xml myApp.html AIRAliases.js
```

Запустите ADT как программу Java (путь к классу Java должен включать пакет ADT.jar):

```
java com.adobe.air.ADT –package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml
myApp.swf
```

Запустите ADT в качестве задания Java в Apache Ant:

```
<property name="SDK_HOME" value="C:/AIRSDK"/>
<property name="ADT.JAR" value="${SDK_HOME}/lib/adt.jar"/>

target name="package">
    <java jar="${ADT.JAR}" fork="true" failonerror="true">
        <arg value="-package"/>
        <arg value="-storetype"/>
        <arg value="pkcs12"/>
        <arg value="-keystore"/>
        <arg value="../../ExampleCert.p12"/>
        <arg value="myApp.air"/>
        <arg value="myApp.xml"/>
        <arg value="myApp.xml"/>
        <arg value="icons/*.png"/>
    </java>
</target>
```

Сообщения об ошибках ADT

В следующих таблицах перечислены возможные ошибки, которые могут регистрироваться программой ADT, и их возможные причины:

Ошибки при проверке дескриптора приложения

Код ошибки	Описание	Примечания
100	Невозможно выполнить синтаксический анализ дескриптора приложения	Проверьте, нет ли в дескрипторе приложения синтаксических ошибок XML, например незакрытых тегов.
101	Отсутствует пространство имен	Добавьте недостающее пространство имен.
102	Недопустимое пространство имен	Проверьте написание пространства имен.
103	Непредвиденный элемент или атрибут	Удалите неподходящие элементы или атрибуты. Нестандартные значения в файле дескриптора не допустимы. Проверьте написание имен элементов или атрибутов. Убедитесь, что элементы размещены внутри соответствующих родительских элементов, а атрибуты используются с допустимыми элементами.
104	Отсутствует элемент или атрибут	Добавьте недостающий элемент или атрибут.

Код ошибки	Описание	Примечания
105	Элемент или атрибут содержит недопустимое значение	Исправьте неподходящее значение.
106	Недопустимая комбинация атрибутов окна	Некоторые параметры окна, например `transparency = true` и `systemChrome = standard`, не могут использоваться одновременно. Измените один из несовместимых параметров.
107	Минимальный размер окна больше, чем максимальный размер окна.	Измените или минимальное, или максимальное значение размера окна.

Сведения о пространствах имен, элементах, атрибутах и их допустимых значениях см. в разделе ««Задание свойств приложения AIR» на странице 71».

Ошибки значка приложения

Код ошибки	Описание	Примечания
200	Невозможно открыть файл значка	Проверьте существование файла по указанному пути. Попробуйте открыть файл другим приложением.
201	Неверный размер значка	Размер значка (в пикселях) должен соответствовать тегу XML. Для примера рассмотрим элемент дескриптора приложения: `<image32x32>icon.png</image32x32>` Размер изображения в значке `icon.png` должен быть ровно 32x32 пикселя.
202	Файл значка содержит неподдерживаемый формат изображения	Поддерживается только формат PNG. Преобразуйте изображения в другие форматы перед созданием пакета приложения.

Ошибки в файле приложения

Код ошибки	Описание	Примечания
300	Файл отсутствует, или его невозможно открыть	Невозможно найти или открыть файл, указанный в командной строке.
301	Файл дескриптора приложения отсутствует, или его невозможно открыть	Файл дескриптора приложения невозможно найти по указанному пути или его нельзя открыть.
302	В пакете отсутствует корневой файл содержимого	SWF- или HTML-файл, на который ссылается элемент `<content>` дескриптора приложения, должен быть добавлен к пакету путем включения в список файлов в командной строке ADT.

Код ошибки	Описание	Примечания
303	Файл значка отсутствует в пакете	Файлы значков, указанные в дескрипторе приложения, должны быть добавлены к пакету путем включения их в список файлов в командной строке ADT. Файлы значков не добавляются автоматически.
304	Недопустимое исходное содержимое окна	Файл, на который ссылается элемент`<content>` дескриптора приложения, не распознан как допустимый HTML- или SWF-файл.
305	Версия SWF содержимого исходного окна превышает версию пространства имен	Версия SWF-файла, на который есть ссылка в элементе `<content>` дескриптора приложений, не поддерживается версией AIR, указанной в пространстве имен дескриптора. Например, при попытке создать пакет с файлом SWF10 (Adobe Flash Player 10) в качестве исходного содержимого приложения AIR 1.1 произойдет следующая ошибка.
306	Профиль не поддерживается.	Профиль, указываемый в файле дескриптора приложения, не поддерживается. См. раздел ««Ограничение профилей целевого приложения» на странице 77».
	Пространство имен не поддерживает функции.	Используйте подходящее пространство имен для функций (такое как пространство имен 2.0).

Коды выхода для других ошибок

Код выхода	Описание	Примечания
2	Ошибка использования	Проверьте правильность аргументов командной строки
5	Неизвестная ошибка	Данная ошибка указывает на ситуацию, которая не может быть объяснена условиями общих ошибок. К числу возможных причин проблемы относится несовместимость ADT и JRE, повреждения в установках ADT и JRE, а также ошибки программирования в ADT.
6	Запись в целевую папку невозможна	Убедитесь, что указанная (или подразумеваемая) целевая папка доступна, а на содержащем ее диске достаточно свободного места.
7	Доступ к сертификату невозможен	Убедитесь, что правильно указан путь к хранилищу ключей. Убедитесь, что возможен доступ к сертификату в хранилище ключей. Служебную программу Java 1.6 Keytool можно использовать для поиска и устранения проблем с доступом к сертификатам.

Код выхода	Описание	Примечания
8	Недопустимый сертификат	Файл сертификата неправильно сформирован, изменен, просрочен или отозван.
9	Не удается подписать файл AIR	Проверьте параметры подписи, переданные ADT.
10	Не удалось создать отметку времени	ADT не может установить соединение с сервером отметок времени. В случае подключения к Интернету через прокси-сервер, возможно, потребуется выполнить настройку параметров прокси для JRE.
11	Ошибка создания сертификата	Проверьте аргументы командной строки, использованной для создания подписей.
12	Недопустимый ввод	Проверьте пути к файлам и другие аргументы, переданные в ADT из командной строки.

Способы подписи командной строки с помощью ADT

ADT использует криптографическую архитектуру Java (JCA) для доступа к закрытым ключам и сертификатам подписи приложений AIR. Параметры подписи позволяют определить хранилище ключей, а также закрытый ключ и сертификат внутри него.

В хранилище ключей должны быть закрытый ключ и связанная с ним цепочка сертификатов. Если сертификат подписи ведет к надежному сертификату на компьютере, общее имя сертификата выводится в качестве имени издателя в диалоговом окне установки AIR.

ADT требует, чтобы сертификат отвечал стандарту x509v3 (RFC3280) и включал расширение Extended Key Usage с соответствующими значениями для подписания кода. Ограничения сертификата соблюдаются, поэтому некоторые сертификаты могут не подойти для подписи приложений AIR.

Примечание. *Когда возможно, ADT использует параметры прокси среды выполнения Java для подключения к интернет-ресурсам, что необходимо для проверки отозванных сертификатов и получения временных отметок. Если при использовании ADT у вас возникают проблемы с интернет-подключением, а сетевые параметры должны включать особые настройки прокси, возможно, следует изменить конфигурацию прокси рабочей среды Java.*

Указание параметров подписи AIR

❖ Для указания параметров подписи ADT для команд `-package` и `-prepare` используйте следующий синтаксис:

```
[-alias aliasName] [-storetype type] [-keystore path] [-storepass password1] [-keypass password2] [-providerName className] [-tsa url]
```

-alias aliasName — псевдоним ключа в хранилище. Псевдоним необязателен, если в хранилище всего один сертификат. Если псевдоним не задан, ADT просто использует первый ключ.

Не все программы управления хранилищами ключей позволяют давать сертификатам псевдонимы. Например, в системном хранилище ключей Windows в качестве псевдонима нужно использовать отличительное имя сертификата. Для вывода списка доступных сертификатов можно использовать утилиту Java Keytool, чтобы легче было определить псевдоним. Например, выполнение команды:

```
keytool -list -storetype Windows-MY
```

выводит следующую информацию о сертификатах:

```
CN=TestingCert,OU=QE,O=Adobe,C=US, PrivateKeyEntry,
Certificate fingerprint (MD5): 73:D5:21:E9:8A:28:0A:AB:FD:1D:11:EA:BB:A7:55:88
```

Для ссылки на этот сертификат из командной строки ADT задайте следующий псевдоним:

```
CN=TestingCert,OU=QE,O=Adobe,C=US
```

В Mac OS X псевдонимом сертификата в цепочке ключей (Keychain) является имя, отображаемое в приложении Keychain Access.

-storetype type — тип хранилища, заданный его реализацией. Реализация хранилища по умолчанию, включаемая в среды Java, поддерживает типы `JKS` и `PKCS12` types. Java 5.0 поддерживает и тип `PKCS11` для доступа к аппаратным маркерам, и тип `Keychain` для доступа к цепочке ключей Mac OS X. Java 6.0 поддерживает тип `MSCAPI` (в Windows). Если установлены и настроены другие поставщики JCA, могут быть доступны и другие типы хранилищ ключей. Если тип хранилища ключей не задан, то по умолчанию используется тип поставщика JCA.

Тип хранилища	Формат хранилища	Минимальная версия Java
JKS	Файл Java keystore (.keystore)	1.2
PKCS12	Файл PKCS12 (.p12 или .pfx)	1.4
PKCS11	Аппаратный маркер	1.5
KeychainStore	Цепочка ключей Mac OS X	1.5
Windows-MY или Windows-ROOT	MSCAPI	1.6

-keystore path — путь к файлу-хранилищу ключей, если хранилище является файлом.

-storepass password1 — пароль для доступа к хранилищу ключей. Если не задан, ADT запросит пароль.

-keypass password2 — пароль для доступа к закрытому ключу, которым подписано приложение AIR. Если не задан, ADT запросит пароль.

-providerName className — поставщик JCA для определенного типа хранилища ключей. Если не задан, ADT использует поставщика по умолчанию для этого типа хранилища.

-tsa url — задает URL-адрес сервера меток времени, соответствующего стандарту RFC3161 для добавления метки времени к цифровой подписи. Если URL не задан, используется временная отметка Geotrust по умолчанию. Когда подписанное приложение AIR снабжено временной отметкой, приложение можно будет установить и после истечения срока действия сертификата, потому что отметка подтверждает, что на момент подписания сертификат был действителен.

Если ADT не может подключиться к серверу выдачи временных отметок, подпись не выполняется и пакет не формируется. Для отмены временных отметок укажите `-tsa none`. Однако приложение AIR, упакованное без временной отметки, нельзя будет установить после истечения срока действия сертификата.

Примечание. *Параметры подписи схожи с тем же параметрами утилиты Java Keytool. Утилиту Keytool можно использовать для изучения и управления хранилищами ключей в Windows. В Mac OS X для этой цели используется утилита безопасности Apple®.*

Примеры параметров подписи

Подпись с помощью файла .p12:

```
-storetype pkcs12 -keystore cert.p12
```

Подпись с помощью файла Java keystore по умолчанию:

```
-alias AIRcert -storetype jks
```

Подпись с помощью заданного файла Java keystore:

```
-alias AIRcert -storetype jks -keystore certStore.keystore
```

Подпись с помощью файла цепочки ключей Mac OS X:

```
-alias AIRcert -storetype KeychainStore -providerName Apple
```

Подпись с помощью файла системного хранилища ключей Windows:

```
-alias cn=AIRCert -storeype Windows-MY
```

Подпись с помощью аппаратного маркера (см. инструкции производителя маркера, чтобы настроить конфигурацию Java для использования маркера и получения верного значения `providerName`):

```
-alias AIRCert -storetype pkcs11 -providerName tokenProviderName
```

Подпись без включения временной отметки:

```
-storetype pkcs12 -keystore cert.p12 -tsa none
```

Упаковка приложения AIR в собственной программе установки

В AIR 2 можно использовать ADT для создания собственных установщиков приложения для распространения приложений AIR. Например, можно создать EXE-файл установщика для установки приложения AIR в ОС Windows. Можно создать DMG-файл установщика для установки приложения AIR в ОС Mac OS. Можно создать DEB- или RPM-файл установщика для установки приложения AIR в ОС Linux.

Приложения, устанавливаемые с помощью собственной программы установки, называются приложениями с профилем расширенного рабочего стола. Нельзя использовать ADT для упаковки собственного установщика для приложения AIR, если файл дескриптора приложения не поддерживает расширенный профиль настольного компьютера. Этот профиль можно запретить путем использования элемента `supportedProfiles` в файле дескриптора приложения. См. раздел ««Ограничение профилей целевого приложения» на странице 77».

Создать версию собственного установщика приложения AIR можно двумя основными способами:

- Собственный установщик можно создать на основе файла дескриптора приложения и других исходных файлов. (В число других исходных файлов могут входить SWF-файлы, HTML-файлы и прочие активы.)
- Собственный установщик можно создать на основе AIR- или AIRI-файла.

Необходимо использовать ADT в той же ОС, для которой создается собственный файл установщика. Таким образом, чтобы создать EXE-файл для ОС, необходимо запустить ADT в ОС Windows. Чтобы создать DMG-файл для ОС Mac OS, необходимо запустить ADT в ОС Mac OS. Чтобы создать DEB- или RPG-файл для ОС Linux, необходимо запустить ADT в ОС Linux.

При создании собственного установщика для распространения приложения AIR оно приобретает следующие свойства.

- Оно может запускать и взаимодействовать с собственными процессами, используя класс NativeProcess. Дополнительные сведения см. в следующих ресурсах.
  - Взаимодействие с собственными процессами в AIR (для разработчиков ActionScript)
  - Взаимодействие с собственными процессами в AIR (для разработчиков HTML)
- Оно может вызывать метод `File.openWithDefaultApplication()` для открытия любого файла в предназначенном для его открытия приложении независимо от его типа. (Существуют ограничения на приложения, которые *не* устанавливаются собственным установщиком. Дополнительные сведения см. в справочнике ActionScript® 3.0 для Adobe® Flash® Professional CS5 в разделе, посвященном методу `File.openWithDefaultApplication()`.)

При двойном нажатии файла собственного установщика производится установка приложения AIR. Если на компьютере не установлена требуемая версия среды Adobe AIR, перед установкой приложения производится ее загрузка и установка. Если подключение к сети отсутствует, и загрузить требуемую версию Adobe AIR (если требуется) нельзя, установка прекращается. Установка также невозможна, если операционная система не поддерживается Adobe AIR 2.

Примечание. *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.)*

Создание собственного установщика из исходных файлов приложения

Для создания собственного установщика из исходных файлов приложения используйте команду `-package` со следующим синтаксисом (в единой командной строке):

```
adt -package AIR_SIGNING_OPTIONS -target native [WINDOWS_INSTALLER_SIGNING_OPTIONS]
installer_file app_xml [file_or_dir | -C dir file_or_dir | -e file dir ...] ...
```

Этот синтаксис аналогичен синтаксису упаковки файла AIR (без собственного установщика). Однако существуют некоторые отличия:

- К команде добавляется параметр `-target native`. (Если указать параметр `-target air`, ADT создаст файл AIR вместо файла собственного установщика.)
- Целевой DMG- или EXE-файл необходимо указать в качестве `installer_file`.
- К тому же в ОС Windows можно добавить второй набор параметров подписи (см. `[WINDOWS_INSTALLER_SIGNING_OPTIONS]` в примере синтаксиса). Помимо подписи файла AIR в ОС Windows можно подписать файл установщика Windows. Используйте тот же тип сертификата и синтаксис параметров подписи, что и для подписи файла AIR (см. раздел ««Способы подписи командной строки с помощью ADT» на странице 63»). Для подписи файла AIR и файла установщика можно использовать один и тот же или разные сертификаты. Когда пользователь загружает подписанный файл установщика Windows из Интернета, ОС Windows определяет источник файла по его сертификату.

Дополнительные сведения по другим параметрам ADT (помимо `-target`) см. в разделе ««Упаковка установочного файла AIR» на странице 54».

Ниже приведены примеры создания DMG-файла (файла собственного установщика в ОС Mac OS):

```
adt -package -storetype pkcs12 -keystore myCert.pfx -target native myApp.dmg application.xml
index.html resources
```

Ниже приведены примеры создания EXE-файла (файла собственного установщика в ОС Windows):

```
adt -package -storetype pkcs12 -keystore myCert.pfx -target native myApp.exe application.xml
index.html resources
```

Ниже приведен пример создания и подписи EXE-файла:

```
adt -package -storetype pkcs12 -keystore myCert.pfx -target native -storetype pkcs12 -keystore
myCert.pfx myApp.exe application.xml index.html resources
```

Создание собственного установщика из файла AIR или AIRI

С помощью ADT можно создать собственный установщик на основе файла AIR или AIRI. Для создания собственного установщика на основе файла AIR используйте команду ADT `-package` со следующим синтаксисом (в единой командной строке):

```
adt -package -target native [WINDOWS_INSTALLER_SIGNING_OPTIONS] installer_file air_file
```

Этот синтаксис аналогичен синтаксису для создания собственного установщика на основе исходных файлов приложений AIR. Однако существуют некоторые отличия:

- В качестве источника указывается файл AIR, а не файл дескриптора приложения и прочие исходные файлы приложения AIR.
- Так как файл AIR уже подписан, параметры подписи для него указывать не надо

Для создания собственного установщика на основе файла AIRI используйте команду ADT *-package* со следующим синтаксисом (в одной командной строке):

```
adt AIR_SIGNING_OPTIONS -package -target native [WINDOWS_INSTALLER_SIGNING_OPTIONS]
installer_file airi_file
```

Этот синтаксис аналогичен синтаксису для создания собственного установщика на основе файла AIR. Однако существуют некоторые отличия:

- В качестве источника указывается файл AIRI.
- Необходимо указать параметры подписи для целевого приложения AIR.

Ниже приведен пример создания DMG-файла (файла собственного установщика в ОС Mac OS) на основе файла AIR:

```
adt -package -target native myApp.dmg myApp.air
```

Ниже приведен пример создания EXE-файла (файла собственного установщика в ОС Windows) на основе файла AIR:

```
adt -package -target native myApp.exe myApp.air
```

Ниже приведен пример создания (на основе файла AIR) и подписи EXE-файла:

```
adt -package -target native -storetype pkcs12 -keystore myCert.pfx myApp.exe myApp.air
```

Ниже приведен пример создания DMG-файла (файла собственного установщика в ОС Mac OS) на основе файла AIRI:

```
adt -storetype pkcs12 -keystore myCert.pfx -packag e-target native myApp.dmg myApp.airi
```

Ниже приведен пример создания EXE-файла (файла собственного установщика в ОС Windows) на основе файла AIRI:

```
adt -storetype pkcs12 -keystore myCert.pfx -package -target native myApp.exe myApp.airi
```

Ниже приведен пример создания (на основе файла AIRI) и подписи EXE-файла:

```
adt -package -storetype pkcs12 -keystore myCert.pfx -target native -storetype pkcs12 -keystore
myCert.pfx myApp.exe myApp.airi
```

Создание неподписанного промежуточного файла AIR с помощью ADT

С помощью команды `-prepare` можно создать неподписанный промежуточный файл AIR. Промежуточный файл AIR должен быть подписан командной ADT `-sign`, чтобы получился действительный файл установки AIR.

Команда `-prepare` принимает те же настройки и параметры, что и команда `-package` (кроме параметров подписи). Единственная разница в том, что получаемый файл не подписывается. Промежуточный файл имеет расширение `airi`.

Подписать промежуточный файл AIR можно с помощью команды ADT `-sign`. (См. раздел ««Подписание промежуточного файла AIR с помощью ADT» на странице 68».)

Пример команды ADT -prepare

```
adt –prepare unsignedMyApp.airi myApp.xml myApp.swf components.swc
```

Подписание промежуточного файла AIR с помощью ADT

Подписать промежуточный файл AIR можно с помощью команды ADT `-sign`. Команда sign может использоваться только с промежуточными файлами AIR (с расширением `airi`). Файл AIR нельзя подписать повторно.

Создать промежуточный файл AIR можно с помощью команды ADT `-prepare`. (См. раздел ««Создание неподписанного промежуточного файла AIR с помощью ADT» на странице 68».)

Подпись файла AIRI

❖ Используйте команду ADT `-sign` со следующим синтаксисом:

```
adt -sign SIGNING_OPTIONSairi_fileair_file
```

SIGNING_OPTIONS. Параметры подписи определяют закрытый ключ и сертификат подписи файла AIR. Они описаны в разделе «Способы подписи командной строки с помощью ADT» на странице 63.

airi_file — путь к неподписанному промежуточному файлу AIR, который требуется подписать.

air_file — имя создаваемого файла AIR.

Пример команды ADT -sign

```
adt –sign -storetype pkcs12 -keystore cert.p12 unsignedMyApp.airi myApp.air
```

Дополнительные сведения см. в разделе ««Цифровая подпись файлов AIR» на странице 96».

Подпись файла AIR для изменения сертификата приложения

Чтобы опубликовать обновление для существующего приложения AIR при использовании нового или возобновленного сертификата подписи, выполните команду ADT `-migrate` для применения подписи переноса сертификата. Подпись переноса — это вторая подпись, применяемая к файлу AIR, использующему исходный сертификат. Подпись переноса позволяет проверить, что обновление приложения было создано владельцем исходного сертификата.

Чтобы применить подпись переноса, исходный сертификат должен быть действителен или быть просроченным не более чем на 180 дней. По окончании срока действия сертификата и прошествии 180-дневного льготного периода применить подпись переноса невозможно. Пользователям приложения нужно будет удалить имеющуюся версию, прежде чем устанавливать обновленную. Подпись переноса по умолчанию снабжается временной отметкой, поэтому обновления AIR с подписью переноса останутся действительными даже после истечения срока действия сертификата.

Примечание. *180-дневный льготный период применяется только к приложениям, для которых в пространстве имен дескриптора приложения указана версия AIR 1.5.3 или более поздняя версия.*

Для переноса приложения и использования нового или возобновленного сертификата необходимо выполнить следующее.

1 Создайте обновление приложения

2 Упакуйте и подпишите файл обновления AIR **новым** сертификатом

3 Подпишите файл AIR снова, на этот раз **оригинальным** сертификатом с помощью команды ADT `-migrate`

Файл AIR, подписанный командой `-migrate`, может использоваться как для установки новой версии приложения, так и для обновления всех предыдущих, включая и версии, подписанные старым сертификатом.

Примечание. *При обновлении приложения, опубликованного для версии AIR, предшествующей 1.5.3, в дескрипторе приложения необходимо указать исходный идентификатор издателя. В противном случае перед установкой обновления пользователям приложения потребуется удалить более раннюю версию.*

Перенос приложения AIR для использования нового сертификата

❖ Используйте команду ADT `-migrate` со следующим синтаксисом:

```
adt -migrate SIGNING_OPTIONS air_file_in air_file_out
```

SIGNING_OPTIONS. Параметры подписи определяют закрытый ключ и сертификат подписи файла AIR. Эти параметры должны определять **оригинальный** сертификат подписи (см. раздел «Способы подписи командной строки с помощью ADT» на странице 63).

air_file_in — обновляемый файл AIR, подписанный **новым** сертификатом.

air_file_out — создаваемый файл AIR.

Пример ADT

```
adt –migrate -storetype pkcs12 -keystore cert.p12 myApp.air myApp.air
```

Дополнительные сведения см. в разделе ««Цифровая подпись файлов AIR» на странице 96».

Примечание. *Команда `-migrate` была добавлена в ADT в AIR версии 1.1.*

Создание самозаверяющего сертификата с помощью ADT

Для создания допустимых установочных файлов AIR можно использовать самозаверяющие сертификаты. Но самозаверяющие сертификаты обеспечивают пользователям ограниченную гарантию безопасности. Подлинность самозаверяющих сертификатов проверить нельзя. При установке файла AIR, подписанного самозаверяющим сертификатом, его издатель отображается как «Неизвестен». Сертификат, созданный ADT, действителен в течение 5 лет.

Если вы создаете обновление для приложения AIR, подписанного самозаверяющим сертификатом, следует использовать тот же сертификат для подписи оригинального файла и обновления AIR. Сертификаты, создаваемые ADT, всегда уникальны, даже если параметры повторяются. Поэтому, если вы планируете подписывать обновления самозаверяющим сертификатом, созданным ADT, храните оригинальный сертификат в надежном месте. Кроме того, вы не сможете создать обновленный файл AIR после истечения срока действия оригинального сертификата. (Другим сертификатом можно будет подписывать новые приложения, но не новые версии этого же приложения.)

Важная информация. *Из-за ограничений самозаверяющих сертификатов корпорация Adobe настоятельно рекомендует использовать для подписи открыто распространяемых приложений AIR коммерческие сертификаты, выданные известными сертифицирующими органами.*

Сертификат и связанный с ним закрытый ключ, созданный ADT, хранятся в файле-хранилище типа PKCS12. Заданный пароль относится только к ключу, но не к хранилищу.

Генерация сертификата цифрового ID для самозаверяющих файлов AIR

❖ Используйте команду ADT `-certificate` (в единой командной строке):

```
adt -certificate -cn name [-ou org_unit][-o org_name][-c country]
key_type pfx_file password
```

-cn name — это строка, представляющая общее имя нового сертификата.

-ou org_unit — это строка, представляющая организационную единицу, выдавшую сертификат. (необязательно.)

`-o org_name` — это строка, представляющая орган, выдавший сертификат. (необязательно.)

`-c country` — двухбуквенный код страны в формате ISO-3166. Если код недопустимый, сертификат не генерируется. (необязательно.)

`key_type` — тип ключа, используемого с сертификатом: 1024-RSA или 2048-RSA.

pfx_file — путь к генерируемому файлу сертификата.

password Пароль для доступа к новому сертификату. Пароль необходим для подписи файлов AIR этим сертификатом.

Примеры генерации сертификата

```
adt -certificate -cn SelfSign -ou QE -o "Example, Co" -c US 2048-RSA newcert.p12 39#wnetx3tl
adt -certificate -cn ADigitalID 1024-RSA SigningCert.p12 39#wnetx3tl
```

Чтобы подписывать файлы AIR этим сертификатом, используйте команды ADT `-package` или `-prepare`:

```
-storetype pkcs12 -keystore newcert.p12 -keypass 39#wnetx3tl
-storetype pkcs12 -keystore SigningCert.p12 -keypass 39#wnetx3tl
```

Примечание. *Java 1.5 и более поздние версии не позволяют использовать в паролях для защиты файлов сертификатов PKCS12 нестандартные символы ASCII. Используйте в паролях только стандартные символы ASCII.*

Глава 11. Задание свойств приложения AIR

Приложению AIR, помимо всех входящих в него файлов и иных ресурсов, требуется файл дескриптора. Файл дескриптора приложения имеет формат XML и определяет основные свойства приложения.

Во многих средах разработки, поддерживающих AIR, дескриптор приложения создается автоматически при создании проекта. В противном случае необходимо создать свой собственный файл дескриптора. Образец файла дескриптора `descriptor-sample.xml` можно найти в каталоге `samples` и в AIR, и во Flex SDK.

Важно! Не редактируйте файл дескриптора приложения, когда открыто диалоговое окно «Параметры Flash Professional CS5 AIR». Сохраните изменения в файле дескриптора приложения перед открытием диалогового окна «Параметры iPhone».

Также можно использовать файл дескриптора приложения для определения параметров основанного на ActionScript приложения iPhone. Дополнительные сведения см. в разделе Параметры дескриптора приложения для разработки приложения iPhone.

Структура файла дескриптора приложения

В файле дескриптора содержатся настройки, определяющие поведение всего приложения, такие как имя, версия, информация об авторских правах и др. Файл дескриптора приложения может иметь любое имя. При упаковке приложения файл дескриптора приложения переименовывается в `application.xml` и помещается в специальном каталоге в пакете AIR.

Далее приводится образец файла дескриптора приложения:

```
<?xml version="1.0" encoding="utf-8" ?>
<application xmlns="http://ns.adobe.com/air/application/2.0">
    <id>HelloWorld</id>
    <version>2.0</version>
    <filename>Hello World</filename>
    <name>Example Co. AIR Hello World</name>
     <description>
        <text xml:lang="en">This is a example.</text>
        <text xml:lang="fr">C'est un exemple.</text>
        <text xml:lang="es">Esto es un ejemplo.</text>
    </description>
    <copyright>Copyright (c) 2006 Example Co.</copyright>
    <initialWindow>
        <title>Hello World</title>
        <content>
            HelloWorld.swf
        </content>
        <systemChrome>none</systemChrome>
        <transparent>true</transparent>
        <visible>true</visible>
        <minimizable>true</minimizable>
        <maximizable>false</maximizable>
        <resizable>false</resizable>
```

```
        <width>640</width>
        <height>480</height>
        <minSize>320 240</minSize>
        <maxSize>1280 960</maxSize>
    </initialWindow>
    <installFolder>Example Co/Hello World</installFolder>
    <programMenuFolder>Example Co</programMenuFolder>
    <icon>
        <image16x16>icons/smallIcon.png</image16x16>
        <image32x32>icons/mediumIcon.png</image32x32>
        <image48x48>icons/bigIcon.png</image48x48>
        <image128x128>icons/biggestIcon.png</image128x128>
    </icon>
    <customUpdateUI>true</customUpdateUI>
    <allowBrowserInvocation>false</allowBrowserInvocation>
    <fileTypes>
        <fileType>
            <name>adobe.VideoFile</name>
            <extension>avf</extension>
            <description>Adobe Video File</description>
            <contentType>application/vnd.adobe.video-file</contentType>
            <icon>
                <image16x16>icons/avfIcon_16.png</image16x16>
                <image32x32>icons/avfIcon_32.png</image32x32>
                <image48x48>icons/avfIcon_48.png</image48x48>
                <image128x128>icons/avfIcon_128.png</image128x128>
            </icon>
        </fileType>
    </fileTypes>
</application>
```

Если приложение использует файл HTML в качестве содержимого корневого каталога вместо SWF-файла, меняется только элемент `<content>`.

```
<content>
    HelloWorld.html
</content>
```

Определение свойств файла дескриптора приложения

Используйте элементы и атрибуты XML в дескрипторе приложения, чтобы определить следующие типы свойств для своего приложения AIR:

- Требуемая версия среды выполнения AIR
- Идентификатор приложения
- Установочная папка и папка меню программы
- Исходное содержимое и свойства окна
- Файлы значков приложений
- Обеспечивает ли приложение заказной пользовательский интерфейс обновления
- Может ли содержимое SWF, выполняемое в обозревателе пользователя, вызывать приложение

- Сопоставления типов файлов

Указание требуемой версии AIR

Атрибуты корневого элемента файла дескриптора приложения, `application`, который указывает требуемую версию среды выполнения AIR.

```
<application xmlns="http://ns.adobe.com/air/application/2.0">
```

xmlns. Пространство имен AIR, которое необходимо задать в качестве пространства имен XML по умолчанию. Пространство имен изменяется с выпуском каждой новой версии AIR (это касается только старших версий, но не младших версий и исправлений). Последний фрагмент пространства имен, например «2.0», указывает на номер версии среды выполнения, которая требуется для работы приложения. Убедитесь, что для пространства имен установлено AIR 2 (`"http://ns.adobe.com/air/application/2.0"`), если приложение использует какие-либо новые возможности AIR 2.

Для созданных на основе SWF-файлов приложений указанная в дескрипторе приложения версия среды выполнения AIR определяет максимальную версию SWF, которая может быть загружена в качестве исходного содержимого приложения. Приложения, для которых указано AIR 1.0 или AIR 1.1, могут использовать в качестве исходного содержимого только файлы SWF9 (Adobe Flash Player 9) — даже при воспроизведении в среде выполнения AIR 2. Приложения, для которых указано AIR 1.5 (или более ранние версии), могут использовать в качестве исходного содержимого файлы SWF9 или SWF10 (Adobe Flash Player 10).

Версия SWF определяет, какая версия прикладных интерфейсов программирования AIR и Adobe Flash Player доступна. Если файл SWF9 используется в качестве исходного содержимого приложения AIR 1.5, это приложение имеет доступ к прикладным интерфейсам программирования только для AIR 1.1 и Adobe Flash Player 9. Более того, изменения, внесенные в работу существующих API-интерфейсов, в AIR 2.0 или проигрывателе Flash Player 10.1 не будут действовать. (Важные, связанные с безопасностью изменения в прикладных интерфейсах программирования являются исключением из этого правила и могут применяться ретроспективно в текущих или будущих исправлениях для среды выполнения.)

Для HTML-приложений версия среды выполнения, указанная в дескрипторе приложения, определяет, какие версии прикладных интерфейсов программирования AIR и Flash Player доступны приложению. Выполнение HTML, CSS и JavaScript всегда определяется версией пакета Webkit, используемого в установленной среде выполнения AIR, но не дескриптором приложения.

Если приложение AIR загружает SWF-содержимое, версия доступных для этого содержимого прикладных интерфейсов программирования AIR и Adobe Flash Player зависит от способа загрузки содержимого. Иногда эффективная версия определяется пространством имен дескриптора приложения, иногда ее можно определить по версии загружаемого содержимого, а иногда она определяется версией загруженного содержимого. В следующей таблице показано, как определяется версия прикладных интерфейсов программирования на основе метода загрузки.

Метод загрузки содержимого	**Метод определения версии прикладного интерфейса программирования**
Исходное содержимое, приложение на основе SWF-файла	Версия SWF **загруженного** файла
Исходное содержимое, приложение на основе HTML-файла	Пространство имен дескриптора приложения
SWF, загруженный SWF-содержимым	Версия **загружающего** содержимого

Метод загрузки содержимого	**Метод определения версии прикладного интерфейса программирования**
SWF-библиотека, загруженная HTML-содержимым с помощью тега <script>	Пространство имен дескриптора приложения
SWF, загруженный HTML-содержимым с помощью прикладных интерфейсов программирования AIR или Adobe Flash Player (например, flash.display.Loader)	Пространство имен дескриптора приложения
SWF, загруженный HTML-содержимым с помощью тегов <object> или <embed> (либо эквивалентных прикладных интерфейсов программирования JavaScript)	Версия SWF **загруженного** файла

При загрузке SWF-файла другой версии, отличающейся от загружающего содержимого, могут возникнуть две следующие проблемы:

- При загрузке SWF10-содержимого файлом SWF9 (или более ранней версии) — ссылки на API-интерфейсы AIR 1.5 и более поздней версии и Flash Player 10 и более поздней версии в загруженном содержимом не будут разрешаться
- При загрузке SWF9-содержимого (или более ранней версии) из SWF10 — прикладные интерфейсы программирования, измененные в AIR 1.5 и Adobe Flash Player 10, могут работать не предусмотренными для загруженного содержимого способами.

Определение идентификатора приложения

Идентификатор приложения, идентификатор издателя, версия, имя, имя файла, описание и информация об авторских правах задаются в следующих элементах:

```
<id>TestApp</id>
<publisherID>48B5E02D9FB21E6389F73B8D17023320485DF8CE.1</publisherID>
<version>2.0</version>
<filename>TestApp</filename>
<name>
    <text xml:lang="en">Hello AIR</text>
    <text xml:lang="fr">Bonjour AIR</text>
    <text xml:lang="es">Hola AIR</text>
</name>
<description>An MP3 player.</description>
<copyright>Copyright (c) 2008 YourCompany, Inc.</copyright>
```

id — строка идентификатора для приложения, также известна как идентификатор приложения. В значении атрибута могут использоваться следующие символы:

- 0–9
- a–z
- A–Z
- . (точка)
- - (дефис).

Значение может содержать от 1 до 212 символов. Это обязательный элемент.

publisherID (необязательно): используется при обновлении приложения AIR 1.5.2 или более ранней версии до AIR 1.5.3 или более поздней версии. В этом элементе содержится идентификатор издателя более старой версии приложения AIR. Указывайте этот элемент только при разработке приложения для среды AIR 1.5.3 или более поздней версии и при наличии версии приложения, использующей среду AIR 1.5.2 или более ранней версии. Дополнительные сведения см. в разделе ««Об идентификаторах издателя AIR» на странице 98».

При разработке приложения AIR для среды AIR 1.5.3 или более поздней версии и наличии версии, использующей AIR 1.5.2 или более ранней версии, выполните следующее.

1 Определите текущий идентификатор издателя приложения. В установленном приложении он находится в файле META-INF/AIR/publisherid.

2 Добавьте идентификатор издателя в элемент `<publisherID>` файла дескриптора нового приложения.

version. Указывает номер версии приложения. (не путать с версией среды выполнения). Строка версии зависит от приложения. AIR никоим образом не интерпретирует строку версии. Следовательно, версия «3.0» не обязательно будет новее, чем «2.0». Примеры: `"1.0"`, `".4"`, `"0.5"`, `"4.9"`, `"1.3.4a"`. Это обязательный элемент.

filename. Это строка, используемая в качестве имени файла приложения (без расширения) при установке. Файл приложения запускает приложение AIR в среде выполнения. Если значение `name` не задано, `filename` также используется в качестве имени установочной папки. Это обязательный элемент.

Свойство `filename` может содержать любые символы в кодировке Unicode (UTF-8), кроме перечисленных ниже (они недопустимы для использования в именах файлов в различных системах):

Символ	Шестнадцатеричный код
разные	0x00 – x1F
*	x2A
"	x22
:	x3A
>	x3C
<	x3E
?	x3F
\	x5C
|	x7C

Значение `filename` не может заканчиваться точкой.

name (необязательно, но рекомендуется). Заголовок, отображаемый при установке приложения AIR.

Если задан единый текстовый узел (вместо нескольких элементов `text`), установщик приложения AIR использует это имя вне зависимости от языка системы:

```
<name>Test Application</name>
```

Схема дескриптора приложения AIR 1.0 позволяет задать для имени только один простой текстовый узел (а не несколько элементов `text`).

Версия AIR 1.1 (или более поздняя) позволяет задать в элементе `name` несколько языков. Например, ниже показано как задать имя на трех языках (английском, французском и испанском):

```
<name>
    <text xml:lang="en">Hello AIR</text>
    <text xml:lang="fr">Bonjour AIR</text>
    <text xml:lang="es">Hola AIR</text>
</name>
```

Атрибут `xml:lang` для каждого языка задает код языка согласно документу RFC4646 (http://www.ietf.org/rfc/rfc4646.txt).

Установщик AIR использует имя, наиболее близкое к языку пользовательского интерфейса операционной системы. Например, рассмотрим процесс установки, где элемент `name` файла дескриптора приложения содержит значение для английской локали (en). Установщик приложения AIR использует значение en, если язык пользовательского интерфейса определяется как английский (en). Также имя en используется, если языком системы является английский США (en-US). Однако если в системе используется английский США (en-US), а в файле дескриптора приложения определяются имена как для американского (en-US), так и британского (en-GB) английского, то установщик приложения AIR использует значение en-US. Если в приложение не задано имя, соответствующее языку пользовательского интерфейса, установщик приложения AIR использует первое значение `name` из списка в файле дескриптора приложения.

Если элемент `name` не задан, установщик приложения AIR отображает в качестве имени приложения значение `filename`.

Элемент `name` определяет только заголовок приложения AIR, используемый при установке. Установщик приложений AIR поддерживает несколько языков: английский, испанский, итальянский, китайский (традиционный), китайский (упрощенный), корейский, немецкий, португальский (Бразилия), русский, французский, японский, чешский, голландский, шведский, турецкий, польский. Установщик приложения AIR выбирает отображаемый язык (для всего текста, кроме заголовка и описания приложения) на основе языка интерфейса системы. Выбор языка не зависит от параметров в файле дескриптора приложения.

Элемент `name` *не определяет* локали, доступные для установленного и работающего приложения. Дополнительные сведения о разработке многоязычных приложений см. в разделе ««Локализация приложений AIR» на странице 144».

description (необязательно). Описание приложения AIR, отображаемое при установке.

Если задан единый текстовый узел (вместо нескольких элементов text), установщик приложения AIR использует это описание вне зависимости от языка системы:

```
<description>This is a sample AIR application.</description>
```

Схема дескриптора приложения AIR 1.0 позволяет задать для имени только один простой текстовый узел (а не несколько элементов `text`).

Версия AIR 1.1 (или более поздняя) позволяет задать в элементе `description` несколько языков. Например, ниже показано, как задать описание на трех языках (английском, французском и испанском):

```
<description>
    <text xml:lang="en">This is a example.</text>
    <text xml:lang="fr">C'est un exemple.</text>
    <text xml:lang="es">Esto es un ejemplo.</text>
</description>
```

Атрибут `xml:lang` для каждого языка задает код языка согласно документу RFC4646 (http://www.ietf.org/rfc/rfc4646.txt).

Установщик AIR использует описание, наиболее близкое к языку пользовательского интерфейса операционной системы. Например, рассмотрим процесс установки, где элемент `description` файла дескриптора приложения содержит значение для английской локали (en). Установщик приложения AIR использует значение en, если язык интерфейса операционной системы определяется как английский (en). Также имя en используется, если языком системы является английский США (en-US). Однако если в системе используется английский США (en-US), а в файле дескриптора приложения определяются имена как для американского (en-US), так и британского (en-GB) английского, то установщик приложения AIR использует значение en-US. Если в приложении не задано имя, соответствующее языкам пользовательского интерфейса, установщик приложения AIR использует первое значение `description` из списка в файле дескриптора приложения.

Дополнительные сведения о разработке многоязычных приложений см. в разделе ««Локализация приложений AIR» на странице 144».

copyright (необязательно). Информация об авторских правах для приложения AIR. В Mac OS информация об авторских правах отображается в диалоговом окне «О программе» в меню установленного приложения. В Mac OS информация об авторских правах также используется в поле NSHumanReadableCopyright файла Info.plist.

Ограничение профилей целевого приложения

С помощью приложений AIR 2 и iPhone, созданных с использованием кода ActionScript 3.0, можно ограничить целевой профиль скомпилированного приложения.

```
<supportedProfiles>extendedDesktop</supportedProfiles>
```

supportedProfiles (необязательный) — идентифицирует профили, поддерживаемые для приложения.

Любое из следующих трех значений можно включить в элемент supportedProfiles:

- `desktop` — профиль настольного компьютера определяет приложения AIR, которые установлены на настольном компьютере с помощью файла AIR. Эти приложения не имеют доступа к классу NativeProcess (обеспечивающему взаимодействие со стандартными приложениями).
- `extendedDesktop` — расширенный профиль настольного компьютера определяет приложения AIR, которые установлены на настольном компьютере с помощью стандартной программы установки приложений. Эти приложения имеют доступа к классу NativeProcess (обеспечивающему взаимодействие со стандартными приложениями). Дополнительные сведения см. в разделе ««Упаковка приложения AIR в собственной программе установки» на странице 65». См. также Взаимодействие с собственными процессами в AIR (для разработчиков ActionScript) и Взаимодействие с собственными процессами в AIR (для разработчиков HTML).
- `mobileDevice` — профиль мобильного устройства определяет приложения iPhone, созданные с использованием кода ActionScript 3.0.
- `extendedMobileDevice` — профиль расширенного мобильного устройства в настоящее время не используется.

Свойство `supportedProfiles` не является обязательным. Если этот элемент не включен в файл дескриптора приложения, его можно откомпилировать и развернуть для любого профиля.

Чтобы указать несколько профилей, разделите их между собой пробелами. Например, установка следующего параметра указывает, что приложение доступно только в профилях настольного компьютера и расширенных профилях.

```
<supportedProfiles>desktop extendedDesktop</supportedProfiles>
```

Дополнительные сведения см. в разделе ««Профили приложения» на странице 83».

Определение установочной папки и папки меню программы

Установочная папка и папка меню программы определяются следующими свойствами:

```
<installFolder>Acme</installFolder>
<programMenuFolder>Acme/Applications</programMenuFolder>
```

installFolder (необязательно). Определяет подкаталог в установочной папке по умолчанию.

В Windows установочным подкаталогом по умолчанию является папка Program Files. В Mac OS это папка /Applications. В Linux это /opt/. Например, если значение свойства `installFolder` равно `"Acme"`, а приложение называется `"ExampleApp"`, то оно будет установлено в каталог C:\Program Files\Acme\ExampleApp в Windows, в /Applications/Acme/Example.app в MacOS и в /opt/Acme/ExampleApp в Linux.

Если необходимо указать вложенный каталог, используйте косую черту (/) в качестве разделителя:

```
<installFolder>Acme/Power Tools</installFolder>
```

Свойство `installFolder` может содержать любые символы в кодировке Unicode (UTF-8), кроме запрещенных к использованию в именах папок в различных файловых системах (список исключений см. выше в абзаце о свойстве `filename`).

Свойство `installFolder` не является обязательным. Если свойство `installFolder` не задано, приложение устанавливается в подкаталог установочной папки по умолчанию на основании свойства `name`.

programMenuFolder (необязательно). Определяет расположение ярлыков приложения в меню «Программы» в ОС Windows или в меню «Приложения»(Applications) в Linux. (Эта настойка в настоящее время не учитывается в других операционных системах.) Ограничения по символам, которые могут использоваться в значении этого свойства такие же, как для свойства `installFolder`. Это значение *не должно* заканчиваться косой чертой (/).

Определение свойств исходного окна приложения

Когда загрузка приложения AIR завершается, среда выполнения создает исходное окно приложения, используя значения элемента `initialWindow`. После этого среда выполнения загружает SWF- или HTML-файл, указанный в элементе `content`, в это окно.

Ниже приведен пример элемента `initialWindow`:

```
<initialWindow>
    <content>AIRTunes.swf</content>
    <title>AIR Tunes</title>
    <systemChrome>none</systemChrome>
    <transparent>true</transparent>
    <visible>true</visible>
    <minimizable>true</minimizable>
    <maximizable>true</maximizable>
    <resizable>true</resizable>
    <width>400</width>
    <height>600</height>
    <x>150</x>
    <y>150</y>
    <minSize>300 300</minSize>
    <maxSize>800 800</maxSize>
</initialWindow>
```

Если корневое содержимое является файлом HTML вместо SWF-файла, элемент `content` просто указывает на этот HTML-файл:

```
<content>AIRTunes.html</content>
```

Дочерние элементы элемента `initialWindow` задают свойства окна, в которое загружается корневое содержимое.

content. Значение элемента `content` — это URL-адрес главного файла с содержимым в приложении. Это может быть файл в формате SWF или HTML. URL-адрес задается относительно корневого расположения установочной папки. (Если приложение AIR работает с ADL, URL-адрес задается относительно папки, в которой расположен файл дескриптора приложения. Указать другой корневой каталог можно с помощью параметра `root-dir` в ADL).

Примечание. *Так как значение элемента content считается URL-адресом, символы в имени файла должны кодироваться согласно документу RFC 1738. Например, знак пробела необходимо кодировать как `%20`.*

title (не обязательно). Заголовок окна.

systemChrome (необязательно). Если этот атрибут имеет значение `standard`, то отображается стандартный системный Chrome, предоставляемый операционной системой. Если же он имеет значение `none`, системный Chrome не отображается.

Если используется компонент Flex mx:WindowedApplication, то его заказной Chrome применяется тогда, когда стандартный системный Chrome не задан. Параметр системного Chrome нельзя изменить во время выполнения.

transparent (не обязательно). Установите значение `"true"`, если необходимо, чтобы окно приложения поддерживало альфа-смешивание. Окно с прозрачными областями прорисовывается медленнее и требует больше ресурсов памяти. Параметр прозрачности нельзя изменить во время выполнения.

Важная информация. *Свойству `transparent` можно задать значение `true`, только когда `systemChrome` имеет значение `none`.*

visible (необязательно). Значение по умолчанию равно `false`. Значение `true` приведет к тому, что основное окно станет видимым сразу же после создания — в этом случае изменения также затронут расположение компонентов окна.

Компонент Flex mx:WindowedApplication автоматически отображает и активирует окно непосредственно перед отправкой события `applicationComplete`, кроме случаев, когда атрибут `visible` имеет значение `false` в определении MXML.

Иногда полезно скрывать основное окно вначале, чтобы изменения его положения, размера и компоновки его элементов не отображались. Затем окно можно отобразить, вызвав для окна метод `activate()` или задав свойству `visible` значение `true`.

x, y, width, height (не обязательно). Исходные границы главного окна приложения. Если эти значения не заданы, размер окна определяется параметрами корневого SWF-файла или, в случае HTML-файла, операционной системой. Максимальным значением для параметров `width` и `height` является 2880.

minSize, maxSize (не обязательно). Минимальный и максимальный размер окна. Если эти значения не заданы, они определяются операционной системой.

minimizable, maximizable, resizable (необязательно). Указывают, можно ли изменить размер окна, в т. ч. развернуть и свернуть его. По умолчанию их значения равны `true`.

Примечание. *В операционных системах наподобие Mac OS X, где разворачивание окна является операцией изменения размера, рекомендуется задать для свойств maximizable и resizable значения `false`, чтобы размер окна не менялся.*

Дополнительные разделы справки

Внедрение SWF-содержимого в код HTML (для разработчиков ActionScript)

Внедрение SWF-содержимого в код HTML (для разработчиков HTML)

Добавление содержимого PDF в AIR (для разработчиков ActionScript)

Добавление содержимого PDF в AIR (для разработчиков HTML)

Работа с собственными окнами AIR (для разработчиков ActionScript)

Работа с собственными окнами AIR (для разработчиков HTML)

Задание файлов значков

Свойство `icon` указывает один или несколько файлов значков, которые используются в приложении. Значки не являются обязательными. Если свойство `icon` не задано, операционная система отображает значок по умолчанию.

Путь задается относительно корневого каталога приложения. Файлы значков должны быть в формате PNG. Можно задать следующие размеры значков:

```
<icon>
    <image16x16>icons/smallIcon.png</image16x16>
    <image32x32>icons/mediumIcon.png</image32x32>
    <image48x48>icons/bigIcon.png</image48x48>
    <image128x128>icons/biggestIcon.png</image128x128>
</icon>
```

Если удается обнаружить значок для заданного размера, изображение в файле должно точно ему соответствовать. Если значки какого-либо размера отсутствуют, выбирается значок ближайшего размера и масштабируется для конкретного случая.

Примечание. *Значки не добавляются в пакет AIR автоматически. К файлам значков внутри пакета должны вести правильные относительные пути.*

Рекомендуется создать изображение для всех размеров значков. Кроме того, проверьте, хорошо ли смотрятся значки в 16- и 32-битном режиме.

Создание заказного пользовательского интерфейса для обновлений приложения

AIR устанавливает и обновляет приложения с помощью установочных диалогов по умолчанию. Однако для обновления приложения можно создать заказной интерфейс. Чтобы указать, что приложение должно выполнять процесс обновления самостоятельно, необходимо задать элементу `customUpdateUI` значение `true`:

```
<customUpdateUI>true</customUpdateUI>
```

Когда в установленной версии приложения элемент `customUpdateUI` имеет значение `true` и пользователь дважды щелкает на файле AIR для получения новой версии или устанавливает приложение с помощью функции непрерывной установки, в среде выполнения открывается установленная версия приложения. Среда выполнения не открывает заданный по умолчанию установщик приложения AIR. Ход процесса обновления может определяться логикой приложения. (Идентификатор приложения и идентификатор издателя в файле AIR должны совпадать с аналогичными значениями в установленном приложении, чтобы удалось выполнить обновление).

Примечание. *Механизм* `customUpdateUI` *включается только тогда, когда приложение уже установлено и пользователь дважды щелкает на установочном файле AIR, содержащем обновление, или выполняет обновление с помощью функции непрерывной установки. Загрузить и начать обновление в рамках логики приложения с отображением заказного пользовательского интерфейса (при необходимости) можно вне зависимости от того, имеет ли свойство* `customUpdateUI` *значение* `true`.

Дополнительные сведения см. в разделе ««Обновление приложений AIR» на странице 105».

Разрешение вызова приложения из обозревателя

Если задана приведенный ниже параметр, установленное приложение AIR можно запустить из обозревателя (пользователь должен щелкнуть по ссылке на странице в веб-обозревателе).

```
<allowBrowserInvocation>true</allowBrowserInvocation>
```

Значение по умолчанию равно `false`.

Если это значение равно `true`, учитывайте ограничения системы защиты. Они описаны в разделах Вызов приложения AIR из обозревателя (для разработчиков ActionScript) и Вызов приложения AIR из обозревателя (для разработчиков HTML).

Дополнительные сведения см. в разделе ««Установка и запуск приложений AIR с веб-страницы» на странице 87».

Объявление сопоставлений типов файлов

Элемент `fileTypes` позволяет объявлять типы файлов, с которыми могут сопоставляться приложения AIR. После установки приложения AIR, любой объявляемый тип файла регистрируется в операционной системе и, если эти типы файлов еще не сопоставлены с другим приложением, они назначаются приложению AIR. Для переопределения существующих сопоставлений типов файлов с другими приложениями используйте метод `NativeApplication.setAsDefaultApplication()` во время выполнения (желательно с разрешения пользователя).

Примечание. *Методы, используемые в среде выполнения, управляют только сопоставлениями типов файлов, указанных в дескрипторе приложения.*

```
<fileTypes>
    <fileType>
        <name>adobe.VideoFile</name>
        <extension>avf</extension>
        <description>Adobe Video File</description>
        <contentType>application/vnd.adobe.video-file</contentType>
        <icon>
            <image16x16>icons/AIRApp_16.png</image16x16>
            <image32x32>icons/AIRApp_32.png</image32x32>
            <image48x48>icons/AIRApp_48.png</image48x48>
            <image128x128>icons/AIRApp_128.png</image128x128>
        </icon>
    </fileType>
</fileTypes>
```

Элемент `fileTypes` необязателен. Может содержать любое число элементов `fileType`.

Элементы `name` и `extension` необходимы для всех объявлений `fileType`. Файлы с разными расширениями могут иметь одно и то же имя. Расширение определяет уникальный файл. (Указывайте расширение без точки перед ним.) Элемент `description` необязателен, и за его отображение отвечает интерфейс операционной системы. Элемент `contentType` требуется в AIR 1.5 (он был необязателен в AIR 1.0 и 1.1). Это свойство в некоторых ситуациях помогает операционной системе находить оптимальное приложение для открытия файла. Значение должно быть типом содержимого MIME. Обратите внимание, что это значение игнорируется в Linux, если тип файла уже зарегистрирован и имеет назначенный тип MIME.

Для расширений файлов можно задать свои значки таким же образом, как и для остальных элементов приложения. Файлы значков необходимо включить в установочный файл AIR (они не включаются в пакет автоматически).

Когда тип файла сопоставлен с приложением AIR, то при открытии пользователем файла данного типа будет запускаться приложение. Если приложение уже запущено, AIR отправит объект InvokeEvent в работающий экземпляр. В противном случае AIR сначала запустит приложение. В обоих случаях путь к файлу можно извлечь из объекта InvokeEvent, отправленного объектом NativeApplication. Этот путь можно использовать для открытия файла.

Дополнительные сведения см. в следующих источниках.

- Управление связями файлов (для разработчиков ActionScript)
- Управление связями файлов (для разработчиков HTML)
- Захват аргументов командной строки (для разработчиков ActionScript)
- Захват аргументов командной строки (для разработчиков HTML)

Глава 12. Профили приложения

В Adobe AIR 2 представлено понятие *профили*. Профиль — это набор поддерживаемых API-интерфейсов и возможностей. Каждый профиль можно установить на определенном числе компьютеров или устройств.

Например, профиль рабочего стола и профиль расширенного рабочего стола используются для приложений AIR для рабочего стола. Однако профиль расширенного рабочего стола может устанавливать связь с собственными процессами. (Для приложения профиля рабочего стола эта возможность недоступна.) Аналогично, профиль мобильного устройства задает набор возможностей, которые отличаются от возможностей, доступных в других профилях.

Существуют четыре профиля:

Профиль настольного компьютера Профиль рабочего стола задает набор возможностей приложений AIR, устанавливаемых как файлы AIR на настольном компьютере. Эти приложения устанавливаются и выполняются на поддерживаемых платформах рабочих столов (ОС Mac OS, Windows и Linux). Приложения AIR, созданные в версиях, предшествующих AIR 2, имеют именно этот профиль. Некоторые API в этом профиле не работают. Например, настольные приложения не могут взаимодействовать с собственными процессами.

Расширенный профиль настольного компьютера Профиль расширенного рабочего стола задает набор возможностей приложений AIR, упакованных и устанавливаемых с помощью собственной программы установки. Собственные программы установки представляют собой EXE-файлы в ОС Windows, DMG-файлы в ОС Mac OS и DEB- или RPM-файлы в ОС Linux. Приложения расширенного рабочего стола обладают дополнительными возможностями, которые недоступны в приложениях профиля рабочего стола. Упаковка и распространение приложений AIR с помощью собственной программы установки является новой функций AIR 2. Дополнительные сведения см. в разделе ««Упаковка приложения AIR в собственной программе установки» на странице 65».

Профиль мобильного устройства Профиль мобильного устройства задает набор возможностей для приложений, устанавливаемых в мобильных устройствах. Для создания приложений iPhone, iPod touch и iPad можно использовать ActionScript 3.0 и различные API AIR. В настоящее время приложения с профилем мобильного устройства поддерживаются только этими устройствами.

Расширенное мобильное устройство Профиль расширенного мобильного устройства задает набор возможностей для приложений, устанавливаемых в подмножестве мобильных устройств. В этом подмножестве мобильных устройств в добавление к функциональным возможностям, заданным для профиля мобильного устройства, может использоваться класс HTMLLoader. В настоящее время устройства, поддерживающие этот профиль, не существуют.

Ограничение целевых профилей в файле дескриптора приложения

В AIR 2 файле дескриптора приложения содержится элемент `supportedProfiles`, позволяющий ограничивать целевые профили. Например, установка следующего параметра указывает, что приложение можно откомпилировать и развернуть только для профиля настольного компьютера:

```
<supportedProfiles>desktop</supportedProfiles>
```

Если этот элемент настроен, приложение можно упаковать только в перечисленные профили. Значения элемента supportedProfiles.

- `desktop` — профиль рабочего стола
- `extendedDesktop` — профиль расширенного рабочего стола
- `mobileDevice` — профиль мобильного устройства

Элемент `supportedProfiles` необязателен. Если этот элемент не включен в файл дескриптора приложения, его можно упаковать откомпилировать и развернуть для любого профиля.

Чтобы указать несколько профилей в элементе `supportedProfiles`, разделите их между собой пробелами, как это показано ниже:

```
<supportedProfiles>desktop extendedDesktop</supportedProfiles>
```

Возможности различных профилей

Некоторые API ActionScript не поддерживаются в определенных профилях.

Профиль настольного компьютера
Приложения с профилем настольного компьютера имеют доступ ко всем API ActionScript за исключением следующих:

API	Проверка поддержки	Комментарии
Accelerometer	`Accelerometer.isSupported`	Поддержка акселерометра доступна только в профиле мобильного устройства.
CameraRoll	`CameraRoll.isSupported`	Этот класс предоставляет доступ к камере в iPhone.
`File.openWithDefaultApplication ()`	—	Этот метод доступен для приложений с профилем настольного компьютера. Однако приложения с профилем настольного компьютера не могут открывать все типы файлов. Описание ограничений на типы открываемых файлов см. в справочнике ActionScript® 3.0 для Adobe® Flash® Professional CS5.
`File.cacheAsBitmapMatrix`	—	Это свойство поддерживается только в приложениях ActionScript 3.0 на iPhone.
Geolocation	`Geolocation.isSupported`	Поддержка определения местоположения доступна только в профиле мобильного устройства.

API	Проверка поддержки	Комментарии
`NativeApplication.systemIdleMode`	Нет.	В приложениях с профилем настольного компьютера настройка этого свойства не имеет никакого эффекта.
NativeProcess	`NativeProcess.isSupported`	Взаимодействие с собственными процессами доступно только для приложений с расширенным профилем настольного компьютера.
`Stage.orientation`	`Stage.supportsOrientationChange`	API ориентации рабочей области доступен только в профиле мобильного устройства.

Расширенный профиль настольного компьютера

Приложения с расширенным профилем настольного компьютера имеют доступ ко всем API, что и приложения с профилем настольного компьютера. Однако приложения с расширенным профилем настольного компьютера поддерживают две дополнительные возможности.

- Приложения с расширенным профилем настольного компьютера могут запускать собственные процессы и взаимодействовать с ними. Дополнительные сведения см. в следующих ресурсах.
  - Взаимодействие с собственными процессами в AIR (для разработчиков ActionScript)
  - Взаимодействие с собственными процессами в AIR (для разработчиков HTML)
- Приложения с расширенным профилем настольного компьютера могут вызывать метод `File.openWithDefaultApplication()` для всех типов файлов. Ограничения, накладываемые на приложения с профилем настольного компьютера, не распространяются на приложения с расширенным профилем настольного компьютера. Метод `File.openWithDefaultApplication()` позволяет открыть файл в зарегистрированном в системе приложении по умолчанию.

Профиль мобильного устройства

Приложения с профилем мобильного устройства имеют доступ к большинству API ActionScript, но есть некоторые исключения и ограничения. Также существуют API, которые поддерживаются исключительно в профиле мобильного устройства. Дополнительные сведения см. в разделе Поддержка API-интерфейса ActionScript 3.0 для приложений iPhone.

Указание профилей при отладке с помощью ADL

ADL проверяет профили, указанные в элементе `supportedProfiles` файла дескриптора приложения. Если профили указаны, то при отладке ADL по умолчанию использует первый поддерживаемый профиль.

Можно указать профиль в сеансе отладки ADL с помощью аргумента командной строки `-profile`. (См. раздел ««Ограничение профилей целевого приложения» на странице 77».) Этот аргумент можно использовать независимо от того, указан ли профиль в элементе `supportedProfiles` в файле дескриптора приложения. Однако, если используется элемент `supportedProfiles`, указанный в нем профиль должен совпадать с профилем, указанным в командной строке. В противном случае ADL создает ошибку.

Глава 13. Распространение, установка и запуск приложений AIR

Приложения AIR распространяются в виде единого установочного файла AIR, содержащего код и все файлы приложения. Этот файл может распространяться любыми привычными средствами: загружаться, отправляться по электронной почте, записываться на компакт-диск. Для установки пользователю нужно дважды щелкнуть по файлу AIR. Можно использовать возможность *автоматической установки*, которая позволит пользователям устанавливать приложение AIR (и сам пакет Adobe® AIR®, если потребуется) одним щелчком ссылки на веб-странице.

Перед распространением установочный файл AIR необходимо запаковать и снабдить сертификатом подписи кода и закрытым ключом. Цифровая подпись гарантирует, что после подписания приложение не модифицировалось. Кроме того, если цифровой сертификат выписан известным сертифицирующим органом, то доверие пользователей к вам как к издателю будет выше. Файл AIR подписывается при создании пакета приложения с помощью инструмента AIR Developer Tool (ADT).

Сведения об упаковке приложения в файл AIR см. в следующих источниках.

- Adobe® Flex® Builder™, см. веб-страницу Упаковка приложений AIR в Flex Builder.
- Adobe® Flash® Builder™, см. веб-страницу Упаковка приложений AIR в Flash Builder.
- Adobe® Flash® Professional, см. веб-страницу Публикация для Adobe AIR.
- Adobe® Dreamweaver®, см. раздел Создание приложения AIR в Dreamweaver.
- Adobe® AIR® SDK, см. раздел ««Упаковка установочного файла AIR с помощью AIR Developer Tool (ADT)» на странице 54».

Установка и запуск приложения AIR с рабочего стола

Файл AIR можно просто отправить получателю. Например, файл AIR может быть отправлен как вложение в письме или в виде ссылки на веб-страницу.

После загрузки приложения AIR пользователь может установить его, следуя приведенным ниже инструкциям.

1 Дважды щелкните по файлу AIR.

К этому моменту на компьютере должна быть установлена среда Adobe AIR.

2 В окне установки оставьте все параметры по умолчанию и нажмите кнопку «Продолжить».

В ОС Windows AIR автоматически выполняет следующее:

- устанавливает приложение в каталог Program Files;
- создает ярлык приложения на рабочем столе;
- создает ярлык в меню «Пуск»;
- записывает приложение в раздел панели управления «Установка и удаление программ».

В Mac OS приложение по умолчанию добавляется в каталог Applications.

Если приложение уже установлено, программа установки предлагает открыть существующую версию или обновить приложение до версии загруженного файла AIR. Программа установки находит приложение по его ID и ID издателя в файле AIR.

3 После завершения установки нажмите кнопку «Готово».

В Mac OS для установки новой версии приложения пользователю потребуются соответствующие системные привилегии. В Windows и Linux пользователю нужны права администратора.

Также приложение может быть обновлено до новой версии с помощью ActionScript или JavaScript. Дополнительные сведения см. в разделе ««Обновление приложений AIR» на странице 105».

После установки приложения AIR пользователь просто дважды щелкает по значку приложения, и оно запускается.

- В Windows дважды щелкните по значку приложения (на рабочем столе или в папке) или выберите приложение в меню «Пуск».
- В Linux дважды щелкните значок приложения (на рабочем столе или в папке) или выберите приложение в меню программ.
- В Mac OS дважды щелкните по приложению в папке, куда оно было установлено. По умолчанию приложения устанавливаются в каталог /Applications.

Непрерывная установка позволяет установить приложение AIR одним щелчком по ссылке на веб-странице. *Вызов из обозревателя* позволяет запустить установленное приложение AIR одним щелчком по ссылке на веб-странице. Эти функции будут рассмотрены в следующем разделе.

Установка и запуск приложений AIR с веб-страницы

Функция непрерывной установки позволяет встроить SWF-файл в веб-страницу, благодаря чему пользователь сможет установить приложение AIR из обозревателя. Если среда выполнения еще не установлена, то эта функция установит ее. Функция непрерывной установки позволяет пользователю установить приложение AIR, не сохраняя установочный файл на компьютер.

Файл badge.swf включен в AIR SDK и Flex SDK, который позволяет легко использовать возможность автоматической установки. Дополнительные сведения см. в разделе «Установка приложения AIR с помощью файла badge.swf» на странице 88.

Использование функции непрерывной установки продемонстрировано в статье Распространение приложения AIR через Интернет (http://www.adobe.com/go/learn_air_qs_seamless_install_ru) с ознакомительным примером.

Настройка файла непрерывной установки badge.swf

Помимо файла badge.swf, включенного в SDK, можно создать собственный SWF-файл для использования на странице в обозревателе. Этот SWF-файл может взаимодействовать с средой выполнения несколькими способами.

- Он может устанавливать приложение AIR. См. раздел ««Установка приложений AIR из обозревателя» на странице 93».
- Он может проверять, установлено ли данное приложение AIR. См. раздел ««Проверка наличия установленного AIR с веб-страницы» на странице 92».

- Он может проверять, установлена ли среда выполнения. См. раздел ««Проверка наличия установленной среды выполнения» на странице 92».
- Он может запускать установленное приложение AIR в системе пользователя. См. раздел ««Запуск установленных приложений AIR из обозревателя» на странице 94».

Для использования этих возможностей необходимо вызвать API-интерфейсы в SWF-файле на сайте adobe.com: air.swf. Можно настроить файл badge.swf и вызвать прикладные интерфейсы программирования air.swf из собственного SWF-файла.

Кроме того, SWF-файл, запущенный в обозревателе, может взаимодействовать с запущенным приложением AIR с помощью класса LocalConnection. Дополнительные сведения см. в разделе Взаимодействие с другим экземплярами Flash Player и AIR (для разработчиков ActionScript) или Взаимодействие с другим экземплярами Flash Player и AIR (для разработчиков HTML).

Важная информация. *Функции, описанные в данном разделе (а также интерфейсы API в файле air.swf), требуют, чтобы у конечного пользователя в обозревателе был установлен Adobe® Flash® Player 9 с обновлением 3 (в операционных системах Windows или Mac OS). В Linux для поддержки функции непрерывной установки требуется Adobe Flash Player 10 (не ниже версии 10.0.12.36). Можно написать код для проверки установленной версии Flash Player и создать интерфейс для случая, если необходимая версия не установлена. Например, если установлена более ранняя версия проигрывателя Flash Player, можно добавить ссылку для загрузки версии файла AIR (вместо использования файла badge.swf или API-интерфейса air.swf для установки приложения).*

Установка приложения AIR с помощью файла badge.swf

Файл badge.swf включен в AIR SDK и Flex SDK, который позволяет легко использовать возможность автоматической установки. Файл badge.swf может установить среду выполнения и приложение AIR по ссылке с веб-страницы. Файл badge.swf и его исходный код предоставляются для распространения через веб-сайт.

Встраивание файла badge.swf на веб-страницу

1. Найдите следующие файлы, находящиеся в каталоге samples/badge пакета AIR SDK или Flex SDK, а затем добавьте их к своему веб-серверу.
   - badge.swf
   - default_badge.html
   - AC_RunActiveContent.js
2. Откройте страницу default_badge.html в текстовом редакторе.
3. На странице default_badge.html в функции JavaScript `AC_FL_RunContent()` настройте определение параметра `FlashVars` следующим образом:

Параметр	Описание
`appname`	Имя приложения, отображаемое в SWF-файле, если среда выполнения не установлена.
`appurl`	(Обязательное.) URL-адрес файла AIR для загрузки. Необходимо использовать абсолютный, а не относительный URL-адрес.
`airversion`	(Обязательное.) Для среды выполнения версии 1.0 укажите «1.0».
`imageurl`	URL-адрес изображения для значка (не обязательно).
`buttoncolor`	Цвет кнопки загрузки (задается в шестнадцатеричном формате, например `FFCC00`).
`messagecolor`	Цвет текстового сообщения, которое отображается под кнопкой, если среда выполнения не установлена (задается в шестнадцатеричном формате, например `FFCC00`).

4 Минимальный размер файла badge.swf составляет 217 пикселов в ширину и 180 в высоту. Задайте нужные значения для параметров `width` и `height` функции `AC_FL_RunContent()`.

5 Переименуйте файл default_badge.html и измените его код в соответствии со своими требованиями (или включите его в другую HTML-страницу).

Примечание. *Для тега HTML `embed`, который загружает файл badge.swf, не устанавливайте атрибут `wmode`; оставьте для него заданное по умолчанию значение (`"window"`). Другие значения атрибута `wmode` помешают установке в различных системах. Также установка других значений для `wmode` может приводить к ошибке: «Error #2044: Unhandled ErrorEvent:. text=Error #2074: The stage is too small to fit the download ui» («Ошибка №2044: необработанное событие ошибки:. text=Error #2074: рабочая область недостаточна для размещения пользовательского интерфейса загрузки».)*

Также можно редактировать и перекомпилировать файл badge.swf. Дополнительные сведения см. в разделе ««Изменение файла badge.swf» на странице 89».

Установка приложения AIR с веб-страницы по ссылке для непрерывной установки

После добавления на страницу ссылки для непрерывной установки пользователь может установить приложение AIR, щелкнув по ссылке в SWF-файле.

1 Перейдите на страницу HTML в веб-обозревателе, в котором установлен Adobe Flash Player (не ниже версии 9 с обновлением 3 в Windows или Mac OS или версии 10 в Linux).

2 На веб-странице щелкните по ссылке в файле badge.swf.

- Если среда выполнения уже установлена, пропустите этот шаг.
- Если среда выполнения не установлена, отображается диалоговое окно, запрашивающее согласие на установку. Установите среду выполнения (см. раздел ««Установка Adobe AIR» на странице 6») и переходите к следующему шагу.

3 В окне установки оставьте все параметры по умолчанию и нажмите кнопку «Продолжить».

В ОС Windows AIR автоматически выполняет следующее:

- устанавливает приложение в папку C:\Program Files\
- создает ярлык приложения на рабочем столе
- создает ярлык в меню «Пуск»
- записывает приложение в раздел панели управления «Установка и удаление программ»

В Mac OS программа установки добавляет приложение в каталог программ (например, /Applications в Mac OS).

На компьютере с ОС Linux AIR автоматически выполняет следующее:

- Приложение устанавливается в каталог /opt.
- создает ярлык приложения на рабочем столе;
- создает ярлык в меню «Пуск»;
- Добавляет для приложения запись в диспетчере пакетов системы

4 Выберите нужные параметры и нажмите кнопку «Установить».

5 После завершения установки нажмите кнопку «Готово».

Изменение файла badge.swf

Flex SDK и AIR SDK предлагают исходные файлы для badge.swf. Эти файлы содержатся в папке samples/badge пакета SDK:

Исходные файлы	Описание
badge.fla	Исходный файл Flash для компиляции файла badge.swf. Файл badge.fla компилирует файл формата SWF 9 (для загрузки во Flash Player).
AIRBadge.as	Класс ActionScript 3.0, определяющий базовый класс, который используется в файле basdge.fla.

Для изменения интерфейса отображения файла badge.fla можно использовать программы Flash CS3 или Flash CS4.

Функция-конструктор `AIRBadge()`, заданная классом AIRBadge, загружает файл air.swf, размещенный по адресу http://airdownload.adobe.com/air/browserapi/air.swf. Файл air.swf содержит код для использования функции непрерывной установки.

Метод `onInit()` (в классе AIRBadge) вызывается при успешной загрузке файла air.swf:

```
private function onInit(e:Event):void {
    _air = e.target.content;
    switch (_air.getStatus()) {
        case "installed" :
            root.statusMessage.text = "";
            break;
        case "available" :
            if (_appName && _appName.length > 0) {
                root.statusMessage.htmlText = "<p align='center'><font color='#"
                        + _messageColor + "'>In order to run " + _appName +
                        ", this installer will also set up Adobe® AIR®.</font></p>";
            } else {
                root.statusMessage.htmlText = "<p align='center'><font color='#"
                        + _messageColor + "'>In order to run this application, "
                        + "this installer will also set up Adobe® AIR®.</font></p>";
            }
            break;
        case "unavailable" :
            root.statusMessage.htmlText = "<p align='center'><font color='#"
                        + _messageColor
                        + "'>Adobe® AIR® is not available for your system.</font></p>";
            root.buttonBg_mc.enabled = false;
            break;
    }
}
```

Код задает глобальную переменную `_air` основному классу загруженного файла air.swf. Этот класс содержит ряд публичных методов, к которым обращается файл badge.swf для вызова функциональности непрерывной загрузки.

Метод	Описание
`getStatus()`	Определяет, установлена ли на компьютер среда выполнения (или может ли она быть установлена). Дополнительные сведения см. в разделе ««Проверка наличия установленной среды выполнения» на странице 92».
`installApplication()`	Устанавливает указанное приложение на компьютер пользователя. Дополнительные сведения см. в разделе ««Установка приложений AIR из обозревателя» на странице 93». • `url` — строка, задающая URL-адрес. Необходимо использовать абсолютный, а не относительный URL-адрес. • `runtimeVersion` — строка, указывающая версию среды выполнения (например, `"1.0.M6"`), которая необходима для установки приложения. • `arguments` — аргументы, передаваемые приложению, если оно запускается после установки. Приложение запускается сразу после установки, если значение элемента `allowBrowserInvocation` равно `true` в файле дескриптора приложения. (Дополнительные сведения о дескрипторе см. в разделе ««Задание свойств приложения AIR» на странице 71».) Если приложение запускается в конце непрерывной установки из обозревателя (когда пользователь выбрал автоматический запуск после установки), объект приложения NativeApplication отправляет объект BrowserInvokeEvent, только если передаются какие-либо аргументы. Не забывайте об ограничениях безопасности, наложенных на данные, которые вы отправляете приложению. Дополнительные сведения см. в разделе ««Запуск установленных приложений AIR из обозревателя» на странице 94».

Параметры `url` и `runtimeVersion` передаются в SWF-файл через параметры FlashVars в контейнере HTML-страницы.

Если приложение автоматически запускается после установки, то подключение LocalConnection можно использовать для связи установленного приложения с файлом badge.swf после его вызова. Дополнительные сведения см. в разделе Взаимодействие с другим экземплярами Flash Player и AIR (для разработчиков ActionScript) или Взаимодействие с другим экземплярами Flash Player и AIR (для разработчиков HTML).

Проверить, установлено ли приложение, можно и с помощью метода `getApplicationVersion()` в файле air.swf. Этот метод можно вызвать или до начала процесса установки приложения, или после его начала. Дополнительные сведения см. в разделе ««Проверка наличия установленного AIR с веб-страницы» на странице 92».

Загрузка файла air.swf

Можно создать собственный SWF-файл, использующий API-интерфейсы файла air.swf для взаимодействия с средой выполнения и приложениями AIR со страницы в обозревателе. Файл air.swf расположен по адресу http://airdownload.adobe.com/air/browserapi/air.swf. Чтобы сослаться на API-интерфейсы файла air.swf из своего SWF-файла, загрузите файл air.swf на тот же домен, что и свой SWF-файл. В коде ниже показано, как загружать файл air.swf на тот же домен приложения, что и SWF-файл:

```
var airSWF:Object; // This is the reference to the main class of air.swf
var airSWFLoader:Loader = new Loader(); // Used to load the SWF
var loaderContext:LoaderContext = new LoaderContext();
                                // Used to set the application domain

loaderContext.applicationDomain = ApplicationDomain.currentDomain;

airSWFLoader.contentLoaderInfo.addEventListener(Event.INIT, onInit);
airSWFLoader.load(new URLRequest("http://airdownload.adobe.com/air/browserapi/air.swf"),
                  loaderContext);

function onInit(e:Event):void
{
    airSWF = e.target.content;
}
```

Как только загружается файл air.swf (объект `contentLoaderInfo` объекта Loader отправляет событие `init`), можно вызывать любые API-интерфейсы air.swf, описанные в следующих разделах.

Примечание. *Файл badge.swf, поставляемый с AIR SDK и Flex SDK, автоматически загружает файл air.swf. См. раздел ««Установка приложения AIR с помощью файла badge.swf» на странице 88». Инструкции в этом разделе применимы к созданию собственного SWF-файла, загружающего файл air.swf.*

Проверка наличия установленной среды выполнения

SWF-файл может проверять, установлена ли среда выполнения. Для этого он вызывает метод `getStatus(` в файле air.swf, загружаемом с домена http://airdownload.adobe.com/air/browserapi/air.swf. Дополнительные сведения см. в разделе ««Загрузка файла air.swf» на странице 91».

После загрузки файла air.swf ваш SWF-файл может вызывать его метод `getStatus()`, как показано ниже:

```
var status:String = airSWF.getStatus();
```

Метод `getStatus()` возвращает одно из показанных ниже строковых значений, в зависимости от состояния среды выполнения:

Строковое значение	Описание
`"available"`	Среда выполнения может быть установлена, но в настоящий момент не установлена на компьютер.
`"unavailable"`	Среда выполнения не может быть установлена на компьютер.
`"installed"`	Среда выполнения установлена на компьютер.

Метод `getStatus()` возвращает ошибку, если требуемая версия Adobe Flash Player (не ниже версии 9 с обновлением 3 в Windows и Mac OS или версия 10 в Linux) не установлена в обозревателе.

Проверка наличия установленного AIR с веб-страницы

SWF-файл может проверять, установлено ли приложение AIR (с совпадающими ID приложения и ID издателя), с помощью метода `getApplicationVersion()` в файле air.swf, загружаемом с http://airdownload.adobe.com/air/browserapi/air.swf. Дополнительные сведения см. в разделе ««Загрузка файла air.swf» на странице 91».

После загрузки файла air.swf SWF-файл может вызывать его метод `getApplicationVersion()`, как показано ниже:

```
 var appID:String = "com.example.air.myTestApplication";
var pubID:String = "02D88EEED35F84C264A183921344EEA353A629FD.1";
airSWF.getApplicationVersion(appID, pubID, versionDetectCallback);

function versionDetectCallback(version:String):void
{
    if (version == null)
    {
        trace("Not installed.");
        // Take appropriate actions. For instance, present the user with
        // an option to install the application.
    }
    else
    {
        trace("Version", version, "installed.");
        // Take appropriate actions. For instance, enable the
        // user interface to launch the application.
    }
}
```

Для метода `getApplicationVersiom()` предусмотрены следующие параметры:

Параметры	**Описание**
`appID`	ID данного приложения. Дополнительные сведения см. в разделе ««Определение идентификатора приложения» на странице 74».
`pubID`	ID издателя данного приложения. Дополнительные сведения см. в разделе ««Об идентификаторах издателя AIR» на странице 98». Если в рассматриваемом приложении отсутствует идентификатор издателя, задайте для параметра `pubID` пустую строку ("").
`callback`	Функция обратного вызова, играющая роль функции-обработчика. Метод getApplicationVersion() работает асинхронно, он вызывается при обнаружении установленной версии (или ее отсутствии). Метод обратного вызова должен включать один параметр в виде строки, содержащей версию установленного приложения. Если приложение не установлено, то функции передается значение null, как показано в предыдущем примере.

Метод `getApplicationVersion()` возвращает ошибку, если требуемая версия Adobe Flash Player (не ниже версии 9 с обновлением 3 в Windows и Mac OS или версия 10 в Linux) не установлена в обозревателе.

Примечание. *Начиная с версии AIR 1.5.3, идентификатор издателя не используется. Идентификаторы издателей больше не назначаются приложению автоматически. В целях обеспечения обратной совместимости в приложениях по-прежнему может использоваться идентификатор издателя.*

Установка приложений AIR из обозревателя

SWF-файл может устанавливать приложение AIR, вызывая метод `installApplication()` в файле air.swf, загружаемом с `http://airdownload.adobe.com/air/browserapi/air.swf`. Дополнительные сведения см. в разделе ««Загрузка файла air.swf» на странице 91».

После загрузки файла air.swf ваш SWF-файл может вызывать его метод `installApplication()`, как показано ниже:

```
 var url:String = "http://www.example.com/myApplication.air";
var runtimeVersion:String = "1.0";
var arguments:Array = ["launchFromBrowser"]; // Optional
airSWF.installApplication(url, runtimeVersion, arguments);
```

Метод `installApplication()` устанавливает заданное приложение на компьютер пользователя. Ниже перечислены параметры этого метода:

Параметр	Описание
`url`	Строка, задающая URL-адрес устанавливаемого файла AIR. Необходимо использовать абсолютный, а не относительный URL-адрес.
`runtimeVersion`	Строка, указывающая версию среды выполнения (например, «1.0»), которая необходима для установки приложения.
`arguments`	Аргументы, передаваемые приложению, если оно запускается после установки. В аргументах распознаются только алфавитно-цифровые символы. Если требуется передать другие значения, рассмотрите возможность применения схемы кодирования. Приложение запускается сразу после установки, если значение элемента `allowBrowserInvocation` равно `true` в файле дескриптора приложения. (Дополнительные сведения о дескрипторе см. в разделе ««Задание свойств приложения AIR» на странице 71».) Если приложение запускается в конце непрерывной установки из обозревателя (когда пользователь выбрал автоматический запуск после установки), объект приложения NativeApplication отправляет объект BrowserInvokeEvent только если были переданы какие-либо аргументы. Дополнительные сведения см. в разделе ««Запуск установленных приложений AIR из обозревателя» на странице 94».

Метод `installApplication()` работает только при вызове в обработчике пользовательского события, например щелчка мыши.

Метод `installApplication()` возвращает ошибку, если требуемая версия Adobe Flash Player (не ниже версии 9 с обновлением 3 в Windows и Mac OS или версия 10 в Linux) не установлена в обозревателе.

В Mac OS для установки обновленной версии приложения пользователю требуются права, разрешающие установку в каталог приложений (и администраторские права, если требуется обновить среду выполнения). В Windows пользователю нужны права администратора.

Проверить, установлено ли приложение, можно и с помощью метода `getApplicationVersion()` файла air.swf. Этот метод можно вызвать или до начала процесса установки приложения, или после его начала. Дополнительные сведения см. в разделе ««Проверка наличия установленного AIR с веб-страницы» на странице 92». Когда приложение уже запущено, оно может взаимодействовать с SWF-содержимым в обозревателе с помощью класса LocalConnection. Дополнительные сведения см. в разделе Взаимодействие с другим экземплярами Flash Player и AIR (для разработчиков ActionScript) или Взаимодействие с другим экземплярами Flash Player и AIR (для разработчиков HTML).

Запуск установленных приложений AIR из обозревателя

Для использования функции вызова из обозревателя (разрешающей запуск из обозревателя) в файл дескриптора целевого приложения необходимо включить следующий параметр:

```
<allowBrowserInvocation>true</allowBrowserInvocation>
```

Дополнительные сведения о дескрипторе см. в разделе ««Задание свойств приложения AIR» на странице 71».

SWF-файл может запускать в обозревателе приложение AIR, вызывая метод `launchApplication()` в файле air.swf, загружаемом с http://airdownload.adobe.com/air/browserapi/air.swf. Дополнительные сведения см. в разделе ««Загрузка файла air.swf» на странице 91».

После загрузки файла air.swf ваш SWF-файл может вызывать его метод `launchApplication()`, как показано ниже:

```
var appID:String = "com.example.air.myTestApplication";
var pubID:String = "02D88EEED35F84C264A183921344EEA353A629FD.1";
var arguments:Array = ["launchFromBrowser"]; // Optional
airSWF.launchApplication(appID, pubID, arguments);
```

Метод `launchApplication()` определен на верхнем уровне файла air.swf (загружаемого с домена приложения пользовательского интерфейса SWF-файла). Вызов этого метода заставляет AIR запускать указанное приложение (если оно установлено и разрешена функция запуска из обозревателя, что осуществляется с помощью параметра `allowBrowserInvocation` в файле дескриптора приложения). Ниже перечислены параметры этого метода:

Параметр	Описание
`appID`	ID запускаемого приложения. Дополнительные сведения см. в разделе ««Определение идентификатора приложения» на странице 74».
`pubID`	ID издателя запускаемого приложения. Дополнительные сведения см. в разделе ««Об идентификаторах издателя AIR» на странице 98». Если в рассматриваемом приложении отсутствует идентификатор издателя, задайте для параметра `pubID` пустую строку ("")
`arguments`	Массив аргументов, передаваемых приложению. Объект приложения NativeApplication отправляет событие BrowserInvokeEvent, свойство arguments которого указывает на этот массив. В аргументах распознаются только алфавитно-цифровые символы. Если требуется передать другие значения, рассмотрите возможность применения схемы кодирования.

Метод `launchApplication()` работает только при вызове в обработчике пользовательского события, например щелчка мыши.

Метод `launchApplication()` возвращает ошибку, если требуемая версия Adobe Flash Player (не ниже версии 9 с обновлением 3 в Windows и Mac OS или версия 10 в Linux) не установлена в обозревателе.

Если элемент `allowBrowserInvocation` имеет значение `false` в файле дескриптора приложения, вызов метода `launchApplication()` ни к чему не приведет.

Прежде чем выводить пользовательский интерфейс для запуска приложения, может потребоваться вызвать метод `getApplicationVersion()` в файле air.swf. Дополнительные сведения см. в разделе ««Проверка наличия установленного AIR с веб-страницы» на странице 92».

Когда приложение вызывается с помощью функции вызова из обозревателя, его объект NativeApplication отправляет объект BrowserInvokeEvent. Дополнительные сведения см. в разделе Вызов приложения AIR из обозревателя (для разработчиков ActionScript) или Вызов приложения AIR из обозревателя (для разработчиков HTML).

Если используется функция вызова обозревателя, не забывайте об ограничениях безопасности. Эти ограничения описаны в разделе Вызов приложения AIR из обозревателя (для разработчиков ActionScript) и Вызов приложения AIR из обозревателя (для разработчиков HTML).

Когда приложение уже запущено, оно может взаимодействовать с SWF-содержимым в обозревателе с помощью класса LocalConnection. Дополнительные сведения см. в разделе Взаимодействие с другим экземплярами Flash Player и AIR (для разработчиков ActionScript) или Взаимодействие с другим экземплярами Flash Player и AIR (для разработчиков HTML).

Примечание. *Начиная с версии AIR 1.5.3, идентификатор издателя не используется. Идентификаторы издателей больше не назначаются приложению автоматически. В целях обеспечения обратной совместимости в приложениях по-прежнему может использоваться идентификатор издателя.*

Развертывание в корпоративной сети

ИТ-администраторы могут выполнять «тихую» установку среды выполнения и приложений AIR с помощью стандартных инструментов развертывания. ИТ-администраторы могут:

- выполнить «тихую» установку среды выполнения Adobe AIR с помощью таких инструментов, как Microsoft SMS, IBM Tivoli, или любого другого инструмента, поддерживающего «тихую» установку и использующего средства загрузки;
- выполнить «тихую» установку приложения AIR с помощью тех же инструментов, которые используются для развертывания среды выполнения.

Дополнительные сведения см. в *руководстве администратора Adobe AIR (http://www.adobe.com/go/learn_air_admin_guide_ru).*

Журналы установки

Журналы установки сохраняются при установке среды выполнения AIR или приложения AIR. Журнал установки позволяет определить причину проблемы при установке или обновлении.

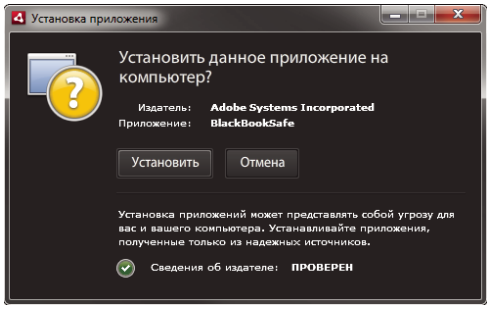
Файлы журналов создаются в следующих папках:

- ОС Mac: стандартный системный журнал (`/private/var/log/system.log`)
- Windows XP: `C:\Documents and Settings\<username>\Local Settings\Application Data\Adobe\AIR\logs\Install.log`
- Windows Vista, Windows 7: `C:\Users\<username>\AppData\Local\Adobe\AIR\logs\Install.log`
- Linux: `/home/<username>/.appdata/Adobe/AIR/Logs/Install.log`

Примечание. *Эти файлы журналов не создаются при использовании версий, предшествующих AIR 2.*

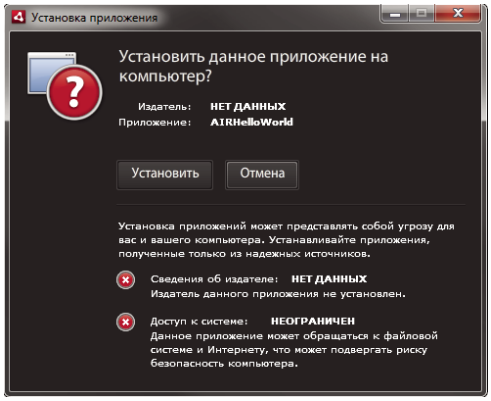
Цифровая подпись файлов AIR

Цифровая подпись установочных файлов AIR с помощью сертификатов, выданных сертифицирующими органами (СО), дает вашим пользователям гарантию того, что в устанавливаемый ими файл не был нечаянно или специально внедрен сторонний код, и обозначает вас в качестве автора подписи (издателя). Во время установки AIR отображает имя издателя, если приложение AIR заверено надежным сертификатом или сертификатом, *указывающим* на надежный сертификат на данном компьютере:



Диалоговое окно подтверждения для приложения, подписанного доверенным сертификатом

Если подписать приложение самоподписанным сертификатом (или если сертификат не связан с доверенным сертификатом), при установке приложения система безопасности пользователя будет подвергаться большей опасности. Дополнительная информация о риске отображается в следующем диалоговом окне:



Диалоговое окно подтверждения для приложения с самоподписанным сертификатом

Важная информация. *Если злоумышленник каким-либо образом получит доступ к файлу с вашим ключом подписи или закрытым ключом, он сможет использовать ваши идентификационные данные в своих файлах AIR.*

Сертификаты для подписывания кодов

Гарантии, ограничения безопасности и юридические обязательства, связанные с использованием сертификатов подписи кода, описаны в документе Certificate Practice Statements (CPS, «Положения об использовании сертификатов») и соглашениях с подписчиками, публикуемых сертифицирующими органами. Дополнительные сведения о соглашениях с сертифицирующими органами, которые выпускают сертификаты для подписания программных кодов AIR в данное время, см. по следующим адресам:

ChosenSecurity (http://www.chosensecurity.com/products/tc_publisher_id_adobe_air.htm)

ChosenSecurity CPS (http://www.chosensecurity.com/resource_center/repository.htm)

GlobalSign (http://www.globalsign.com/developer/code-signing-certificate/index.htm)

GlobalSign CPS (http://www.globalsign.com/repository/index.htm)

CPS Thawte (http://www.thawte.com/cps/index.html)

Соглашение Thawte с разработчиком по подписанию кода (http://www.thawte.com/ssl-digital-certificates/free-guides-whitepapers/pdf/develcertsign.pdf)

VeriSign CPS (http://www.verisign.com/repository/CPS/)

Соглашение с подписчиком VeriSign (https://www.verisign.com/repository/subscriber/SUBAGR.html)

О подписании кода AIR

При подписи файла AIR цифровая подпись включается в установочный файл. Подпись включает опись содержимого пакета, которая нужна, чтобы удостовериться, что файл AIR не изменялся с момента подписи, а также информацию о сертификатах подписи, которая нужна для идентификации издателя.

AIR использует инфраструктуру открытых ключей (PKI), поддерживаемую хранилищем ключей операционной системы, для проверки подлинности сертификата. Для проверки информации об издателе компьютер, на который устанавливается приложение AIR, должен либо подтвердить принятие сертификата, с помощью которого подписано приложение, либо подтвердить принятие цепочки сертификатов, ведущих к сертификату надежного CO.

Если файл AIR подписан сертификатом, не ведущим к одному из основных надежных сертификатов (как правило, это относится к самозаверяющим сертификатам), то информацию об издателе нельзя проверить. Хотя AIR может проверить, не был ли пакет AIR изменен с момента подписи, узнать, кто на самом деле создал и подписал файл, невозможно.

Примечание. *Пользователь может принять самозаверяющий сертификат, после чего приложения AIR, подписанные с его помощью, будут выводить одно для всех имя издателя. В AIR сам пользователь не может отметить сертификат как надежный. Сертификат (без учета закрытого ключа) необходимо предоставлять пользователю отдельно, а он должен импортировать сертификат в нужное место в системном хранилище, пользуясь средствами, предлагаемыми операционной системой или соответствующим инструментом.*

Об идентификаторах издателя AIR

Важная информация. *Начиная с версии AIR 1.5.3, идентификатор издателя больше не используется и не вычисляется на основе сертификата подписи кода. В новых приложениях не требуется и не должен использоваться идентификатор издателя. При обновлении существующих приложений необходимо указать исходный идентификатор издателя в файле дескриптора приложения.*

В версиях, предшествующих AIR 1.5.3, программа установки приложений AIR создавала идентификатор издателя во время установки файла AIR. Это был идентификатор, уникальный для сертификата, используемого для подписи файла AIR. Если один сертификат использовался для нескольких приложений AIR, им назначался одинаковый идентификатор издателя. При подписи обновления приложения с использованием другого сертификата, а иногда даже с использованием возобновленного экземпляра исходного сертификата, идентификатор издателя изменялся.

В AIR 1.5.3 и более поздних версиях идентификатор издателя не назначается средой AIR. В приложении, опубликованном с использованием AIR 1.5.3, строка идентификатора издателя может быть указана в дескрипторе приложения. Идентификатор издателя следует указывать только при публикации обновлений для приложений, которые изначально были опубликованы для версий AIR, предшествующих 1.5.3. Если исходный идентификатор не указывается в дескрипторе приложения, новый пакет AIR не считается обновлением существующего приложения.

Чтобы определить исходный идентификатор издателя, найдите файл `publisherid` в подкаталоге META-INF/AIR, в котором установлено исходное приложение. Строка в этом файле является идентификатором издателя. Чтобы указать идентификатор издателя вручную, в дескрипторе приложения должна быть указана среда выполнения AIR 1.5.3 (или более поздней версии) в объявлении пространства имен файла дескриптора приложения.

При наличии идентификатора издателя он используется в следующих целях:

- как часть ключа шифрования для зашифрованного локального хранилища;
- как часть пути к каталогу хранения приложения;
- как часть строки для локальных соединений;
- как часть строки учетных данных, используемой для вызова приложения в API-интерфейсе обозревателя AIR;

- как часть идентификатора OSID (используется при создании пользовательских программ установки/удаления).

При изменении идентификатора издателя также изменяется поведение функций AIR, зависящих от идентификатора. Например, данные в существующем зашифрованном локальном хранилище становятся недоступными, и в любых экземплярах Flash или AIR, создающих локальное соединение с приложением, в строке подключения должен использоваться новый идентификатор. В среде AIR 1.5.3 или более поздней версии изменение идентификатора издателя установленного приложения невозможно. При использовании другого идентификатора издателя и публикации пакета AIR в программе установки новый пакет считается другим приложением, а не обновлением.

О форматах сертификатов

Средства подписи AIR принимают любые хранилища ключей, доступные в рамках криптографической архитектуры Java (JCA). Сюда включены файлы-хранилища, например файлы формата PKCS12 (обычно имеют расширение .pfx или .p12), файлы Java .keystore, аппаратные хранилища PKCS11 и системные хранилища ключей. Форматы хранилищ ключей, к которым может иметь доступ средство ADT, зависят от версии и конфигурации среды выполнения Java, в которой запускается ADT. Доступ к некоторым типам хранилищ, например аппаратным маркерам PKCS11, возможен только при установке и конфигурации дополнительных программных драйверов и подключаемых модулей JCA.

Чтобы подписать файлы AIR, можно использовать большинство существующих сертификатов для подписания программного кода, а также можно получить новый ускоренно выпущенный сертификат для подписания приложений AIR. Например, можно использовать любой из следующих типов сертификатов, выпущенных VeriSign, Thawte, GlobalSign или ChosenSecurity:

- ChosenSecurity
  - Идентификатор TC Publisher для Adobe AIR
- GlobalSign
  - Сертификат для подписания кода ObjectSign
- Thawte:
  - сертификат разработчика AIR
  - сертификат разработчика Apple
  - сертификат разработчика JavaSoft
  - сертификат Microsoft Authenticode
- VeriSign:
  - Цифровое удостоверение Adobe AIR
  - цифровое удостоверение Microsoft Authenticode
  - цифровое удостоверение Sun Java

Примечание. *Этот сертификат должен создаваться для подписания кода. Нельзя использовать SSL или сертификаты другого типа для подписания файлов AIR.*

Временные отметки

При подписи файла AIR средство упаковки отправляет серверу запрос временной отметки, чтобы получить независимо проверяемую дату и время подписи. Полученная временная отметка включается в файл AIR. Если сертификат подписи действителен на время подписания, приложение AIR может быть установлено, даже если на момент установки срок действия сертификата уже истек. С другой стороны, если временная отметка не получена, файл AIR не может быть установлен после истечения срока действия или отозвания сертификата.

Средства упаковки AIR получают временные отметки по умолчанию. Однако, чтобы приложение можно было упаковать, когда временную отметку получить невозможно, можно отключить эту функцию. Adobe рекомендует включать временные отметки во все файлы AIR, подлежащие открытому распространению.

По умолчанию средства упаковки AIR используют временные отметки Geotrust.

Получение сертификата

Для получения сертификата обычно нужно посетить сайт сертифицирующего органа и пройти процесс заверения. Средства, используемые для создания файла-хранилища ключей для средств AIR, зависят от типа приобретенного сертификата, способа хранения сертификата на принимающем компьютере, а в некоторых случаях и от обозревателя, через который получается сертификат. Например, чтобы получить и экспортировать сертификат разработчика Adobe от Thawte, необходимо использовать Mozilla Firefox. Затем этот сертификат можно экспортировать как файл P12 или PFX непосредственно из интерфейса пользователя Firefox.

Примечание. *Java 1.5 и более поздние версии не позволяют использовать в паролях для защиты файлов сертификатов PKCS12 нестандартные символы ASCII. Язык Java используется средствами разработки AIR для создания подписанного пакета AIR. При экспорте сертификата в виде файла с расширением P12 или PFX, в пароле используются только стандартные символы ASCII.*

С помощью средства ADT, которое используется для упаковки установочных файлов AIR, можно создать собственный самозаверяющий сертификат. Также можно использовать и инструменты сторонних разработчиков.

Инструкции по созданию самозаверяющих сертификатов и по подписи файлов AIR см. в разделе ««Упаковка установочного файла AIR с помощью AIR Developer Tool (ADT)» на странице 54». Файлы AIR также можно экспортировать и подписывать с помощью Flash Builder, Dreamweaver и обновления AIR для Flash.

В примере ниже показано, как получить от Thawte сертификат разработчика AIR и подготовить его для использования с ADT.

Пример: получение сертификата разработчика AIR от Thawte

Примечание. *Рассматривается лишь один из возможных способов получения и подготовки сертификата подписания кода. У каждого выпускающего сертификаты органа есть собственные политики и процедуры.*

Для приобретения сертификата разработчика AIR Thawte требуется использовать обозреватель Mozilla Firefox. Закрытый ключ сертификата сохраняется в хранилище ключей обозревателя. Убедитесь, что хранилище ключей Firefox защищено основным паролем, а компьютер защищен физически. (После завершения процесса заверения можно экспортировать и удалить сертификат из хранилища ключей обозревателя.)

В ходе процесса регистрации сертификата генерируется связка закрытого и открытого ключей. Закрытый ключ автоматически заносится в хранилище ключей Firefox. Для запроса и получения сертификата с сайта Thawte необходимо использовать один и тот же компьютер и один и тот же обозреватель.

1. Зайдите на сайт Thawte и перейдите к странице Product page for Code Signing Certificates («Сертификаты подписания кода»).
2. Выберите из списка сертификатов подписания кода сертификат разработчика AIR (Adobe AIR Developer Certificate).
3. Выполните все три шага процесса регистрации сертификата. Необходимо предоставить информацию об организации и контактные данные. Затем Thawte проверяет подлинность личности заявителя и запрашивает дополнительную информацию. После завершения проверки Thawte отправляет электронное письмо с инструкциями по получению сертификата.

   Примечание. *Дополнительные сведения о типе требуемой документации см. здесь: https://www.thawte.com/ssl-digital-certificates/free-guides-whitepapers/pdf/enroll_codesign_eng.pdf.*
4. Получите выписанный сертификат на сайте Thawte. Сертификат автоматически заносится в хранилище ключей Firefox.
5. Экспортируйте файл-хранилище с закрытым ключом и сертификат из хранилища ключей Firefox, как описано ниже.

   Примечание. *При экспорте закрытого ключа и сертификата из хранилища Firefox файл получает формат .p12 (pfx), который распознают ADT, Flex, Flash и Dreamweaver.*

   a. Откройте диалоговое окно *Менеджер сертификатов* в Firefox:

   b. В ОС Windows откройте Сервис -> Параметры -> Дополнительно -> Шифрование -> Просмотр сертификатов

   c. В Mac OS откройте Firefox -> Установки -> Дополнительно -> Шифрование -> Просмотр сертификатов

   d. В Linux откройте «Правка -> Установки -> Дополнительно -> Шифрование -> Просмотр сертификатов» (Edit -> Preferences -> Advanced -> Encryption -> View Certificates)

   e. Выберите сертификат подписания кода Adobe AIR из списка и нажмите кнопку **Резервная копия**.

   f. Введите имя файла и место его экспорта и нажмите кнопку **Сохранить**.

   g. Если используется основной пароль Firefox, вам потребуется ввести пароль устройства защиты ПО для экспорта файла. (Этот пароль используется только в Firefox.)

   h. В диалоговом окне *Выберите пароль резервного копирования* создайте пароль для файла-хранилища.

      Важная информация. *Пароль нужен для защиты файла хранилища и требуется, когда файл используется для подписи приложений AIR. Необходимо выбирать надежный пароль.*

   i. Нажмите кнопку «ОК». Вы должны увидеть сообщение об успешном принятии пароля резервного копирования. Файл-хранилище с закрытым ключом и сертификатом сохраняется с расширением .p12 (в формате PKCS12)
6. Экспортированный файл можно использовать в ADT, Flash Builder, Flash Professional или Dreamweaver. Пароль, созданный для файла, потребуется при подписи приложения AIR.

Важная информация. *Закрытый ключ и сертификат по-прежнему сохраняются в хранилище ключей Firefox. Это не только позволяет экспортировать дополнительную копию файла с сертификатом, но и открывает еще один канал доступа, который также необходимо обезопасить, чтобы не ставить под угрозу общую безопасность сертификата и закрытого ключа.*

Замена сертификатов

В некоторых случаях необходимо изменить сертификат, используемый для подписи обновлений приложения AIR. Вот некоторые из них:

- Возобновление исходного сертификата подписи.
- обновление самозаверяющего сертификата до сертификата, выданного сертифицирующим органом;
- замена истекающего самозаверяющего сертификата на новый;
- замена одного коммерческого сертификата другим, например при изменении корпоративных данных

Чтобы файл AIR распознавался в среде AIR как обновление, необходимо подписать исходный и обновленный файлы AIR с использованием одного сертификата или применить подпись переноса сертификата для обновления. Подпись переноса — это вторая подпись, применяемая для обновления пакета AIR, использующего исходный сертификат. Подпись переноса использует исходный сертификат для определения того, что подписавшее лицо является издателем приложения.

После установки файла AIR с подписью переноса новый сертификат становится основным. Для последующих обновлений подпись переноса не требуется. Однако подписи переноса следует применять как можно дольше, чтобы обеспечить работу пользователей, пропустивших обновления.

Важная информация. *Сертификат должен быть изменен до того, как истечет срок действия исходного сертификата. Если не создать обновление с подписью переноса до того, как истечет срок действия сертификата, пользователям придется удалять существующую версию приложения перед установкой новой версии. Начиная с версии AIR 1.5.3, просроченный сертификат можно использовать для применения подписи переноса в течение 180-дневного льготного периода со дня окончания срока действия сертификата. (Нельзя использовать просроченный сертификат для применения основной подписи приложения.)*

Для замены сертификата выполните следующие действия:

1. Создайте обновление приложения
2. Упакуйте и подпишите файл обновления AIR **новым** сертификатом
3. Подпишите файл AIR снова, на этот раз **оригинальным** сертификатом (с помощью команды ADT `-migrate`)

В остальных отношениях файл AIR с подписью переноса является обычным файлом AIR. Если в системе, куда устанавливается приложение, нет оригинальной версии, AIR установит новую версию обычным образом.

Примечание. *В версиях, предшествующих AIR 1.5.3, для подписи приложения AIR с использованием возобновленного сертификата не всегда требовалась подпись переноса. Начиная с версии AIR 1.5.3, подпись переноса всегда требуется для возобновленных сертификатов.*

Процедура применения подписи переноса описана в разделе ««Подпись файла AIR для изменения сертификата приложения» на странице 68».

Изменение учетных данных приложения

В версиях, предшествующих AIR 1.5.3, учетные данные приложения AIR изменялись при установке обновления с подписью переноса. Изменение учетных данных приложения имеет несколько последствий, в том числе:

- Новая версия приложения не может получать доступ к данным в существующем зашифрованном локальном хранилище.
- Изменяется адрес каталога хранилища приложения. Данные из старого каталога не копируются в новый каталог. (Но новое приложение может найти старый каталог с помощью ID издателя.)

- Приложение больше не может устанавливать локальные соединения с помощью ID издателя.
- Изменяется строка учетных данных, используемая для доступа к приложению с веб-страницы.
- Изменяется идентификатор OSID приложения. (Идентификатор OSID используется при написании пользовательских программ установки/удаления.)

При публикации обновления с использованием AIR 1.5.3 изменение учетных данных приложения невозможно. Идентификаторы исходного приложения и издателя должны быть указаны в дескрипторе приложения файла AIR обновления. В противном случае новый пакет не распознается как обновление.

Примечание. *При публикации нового приложения AIR с использованием AIR 1.5.3 или более поздней версии не следует указывать идентификатор издателя.*

Просроченные сертификаты

Начиная с версии AIR 1.5.3, сертификат, просроченный не более чем на 180 дней, можно по-прежнему использовать для применения подписи переноса к обновлению приложения. Чтобы воспользоваться этим льготным периодом, в атрибуте пространства имен в дескрипторе приложения обновления должна быть указана версия 1.5.3. По окончании этого льготного периода дальнейшее использование сертификата невозможно. Пользователям, обновляющим приложение до новой версии, потребуется удалить существующую версию. Обратите внимание, что льготный период не предусмотрен в версиях AIR, предшествующих 1.5.3.

Терминология

В этом разделе приводятся определения основных терминов, которые необходимо понимать, чтобы принимать верные решения о том, как подписывать свое приложение для открытого распространения.

Термин	**Описание**
Сертифицирующий орган (CO)	Сеть инфраструктуры открытых ключей, выступающая в роли независимой стороны, которая выдает сертификаты, тем самым подтверждая личность владельца ключа. Обычно CO выдают цифровые сертификаты, подписанные собственным закрытым ключом, — это означает, что личность держателя сертификата подтверждена.
Certificate Practice Statement (CPS, «Положение об использовании сертификатов»)	Содержит рекомендации и правила по выдаче и проверке сертификатов для сертифицирующих органов. CPS является частью договора между CO и его подписчиками и связанными лицами. Также в нем описывается политика проверки подлинности и степени гарантий различных сертификатов.
Список отозванных сертификатов (CRL)	Список выданных сертификатов, которые были впоследствии отозваны и более недействительны. AIR проверяет список CRL во время подписи приложения AIR, а если временной отметки нет, то еще раз при установке.
Цепочка сертификатов	Цепочка сертификатов — это последовательность, в которой каждый сертификат подписывается следующим.
Цифровой сертификат	Цифровой документ, содержащий информацию об идентификационных данных владельца, открытом ключе владельца и идентификационных данных самого сертификата. Сертификат, выданный CO, сам подписан сертификатом, принадлежащим этому CO.
Цифровая подпись	Зашифрованное сообщение или его свертка, которые могут быть расшифрованы только открытым ключом, образующим пару с закрытым. В PKI цифровая подпись содержит один или несколько цифровых сертификатов, которые выводят на выдавших их орган. Цифровая подпись может использоваться для проверки сообщения (или компьютерного файла) на предмет изменений с момента подписи (в пределах, разрешенных криптографическим алгоритмом) и, если исходить из того, что сертифицирующему органу доверяют, проверки подписавшего лица.

Термин	Описание
Хранилище ключей	База данных с цифровыми сертификатами и (в некоторых случаях) связанными с ними закрытыми ключами.
Криптографическая архитектура Java (JCA)	Открытая архитектура для управления и доступа к хранилищам ключей. Дополнительные сведения см. в руководстве по криптографической архитектуре Java.
PKCS #11	Стандарт интерфейса криптографических маркеров, разработанный Лабораторией RSA. Аппаратное хранилище ключей на основе маркеров.
PKCS #12	Стандарт синтаксиса обмена личной информацией, разработанный Лабораторией RSA. Файл-хранилище ключей, содержащий закрытый ключ и связанный с ним цифровой сертификат.
Закрытый ключ	Закрытая часть ассиметричной криптографической системы, состоящей из закрытого и открытого ключей. Закрытый ключ необходимо держать в секрете, и его нельзя передавать по сети. Сообщения, подписанные цифровой подписью, шифруются автором с помощью закрытого ключа.
Открытый ключ	Открытая часть ассиметричной криптографической системы, состоящей из закрытого и открытого ключей. Открытый ключ находится в общем доступе и используется для расшифровки сообщений, зашифрованных закрытым ключом.
Инфраструктура открытых ключей (PKI)	Система доверительных отношений, в которой CO подтверждают личности владельцев открытых ключей. Клиенты сети доверяют цифровым сертификатам, выданным CO, при проверке личности подписавшего цифровое сообщение или файл.
Временная отметка	Цифровая метка, содержащая дату и время того или иного события. ADT может ставить временные отметки согласно серверу времени пакета AIR, соответствующего стандарту RFC 3161. Если функция временных отметок включена, AIR использует ее для проверки подлинности сертификата во время подписания. Это позволяет устанавливать приложение AIR после истечения срока действия сертификата.
Орган выдачи временных отметок	Орган, выдающий временные отметки. Чтобы AIR распознавала временные отметки, они должны соответствовать стандарту RFC 3161, а подпись временной отметки должна вести к надежному основному сертификату на принимающем компьютере.

Глава 14. Обновление приложений AIR

Пользователи могут установить или обновить приложение AIR, дважды нажав файл AIR на компьютере или в обозревателе (пользуясь функцией автоматической установки). Программа установки Adobe® AIR® будет управлять установкой, уведомив пользователя о том, что выполняется обновление уже существующего приложения. (См. раздел ««Распространение, установка и запуск приложений AIR» на странице 86».)

Однако обновить приложение можно и при помощи класса Updater. (Установленное приложение может проверять наличие новой версии для загрузки и установки.) Класс Updater имеет метод `update()`, который позволяет указать файл AIR на компьютере пользователя и обновить приложение до этой версии.

Идентификатор приложения и идентификатор издателя файла обновления AIR должны соответствовать обновляемому приложению. Идентификатор издателя можно получить из сертификата подписи. Обновление, и обновляемое приложение должны быть подписаны одним сертификатом.

В AIR 1.5.3 или более поздней версии файл дескриптора приложения содержит элемент `<publisherID>`. Этот элемент необходимо использовать, если версии приложения создавались в AIR 1.5.2 или более ранней версии. Дополнительные сведения см. в разделе ««Определение идентификатора приложения» на странице 74».

Что касается AIR 1.1 и более поздних версий, можно выполнить перенос приложения для использования нового сертификата подписи кода. Перенос приложения в целях использования новой подписи включает в себя подписывание файла обновления AIR новым и исходным сертификатом. Перенос сертификата является односторонним процессом. После переноса только файлы AIR, подписанные новым сертификатом (или обоими сертификатами), будут распознаны как обновления существующей установки.

Управление обновлением приложений может быть достаточно сложным. AIR 1.5 содержит новую *инфраструктуру обновления для приложений AdobeAIR*. Эта инфраструктура обеспечивает прикладные интерфейсы программирования, помогающие разработчикам создавать удобные возможности обновления в приложениях AIR.

Перенос сертификата можно выполнить для замены самозаверяющего сертификата коммерческим сертификатом подписи кода или смены одного самозаверяющего или коммерческого сертификата на другой. Если не выполнить перенос сертификата, пользователи должны будут удалить текущую версию приложения перед установкой новой. Дополнительные сведения см. в разделе ««Замена сертификатов» на странице 102».

Рекомендуется включить новый механизм обновления в приложение. При появлении новой версии приложения, механизм обновления выведет пользователю приглашение на ее установку.

Программа установки приложения AIR создает файлы журнала при установке, обновлении или удалении приложения AIR. Журналы установок позволяют определить причины проблем при установке или обновлении. См. раздел ««Журналы установки» на странице 96».

Примечание. *Новые версии среды Adobe AIR могут включать в себя обновленные версии WebKit. Обновленная версия WebKit может привести к неожиданным изменениям в содержимом HTML развернутого приложения AIR. Поэтому может потребоваться обновить приложение. Механизм обновления может уведомлять пользователей о выходе новой версии приложения. Дополнительные сведения см. в разделе Сведения о среде HTML (для разработчиков ActionScript) или Сведения о среде HTML (для разработчиков HTML).*

Об обновлении приложений

Класс Updater (в пакете flash.desktop) содержит один метод, `update()`, который можно использовать для обновления выполняемого приложения. Например, если на рабочем столе пользователя размещена версия файла AIR («Sample_App_v2.air»), следующий код обновляет приложение.

Пример ActionScript:

```
var updater:Updater = new Updater();
var airFile:File = File.desktopDirectory.resolvePath("Sample_App_v2.air");
var version:String = "2.01";
updater.update(airFile, version);
```

Пример JavaScript:

```
var updater = new air.Updater();
var airFile = air.File.desktopDirectory.resolvePath("Sample_App_v2.air");
var version = "2.01";
updater.update(airFile, version);
```

Перед использованием класса Updater приложением пользователь или приложение должны загрузить обновленную версию файла AIR на компьютер. Дополнительные сведения см. в разделе ««Загрузка файла AIR на компьютер пользователя» на странице 108».

Результаты вызова метода Updater.update()

Когда приложение вызывает в среде выполнения метод `update()`, среда выполнения закрывает приложение, а затем пытается установить новую версию из файла AIR. Среда выполнения проверяет идентификатор приложения и идентификатор издателя, указанные в файле AIR, на соответствие идентификатору приложения и идентификатору издателя приложения, вызывающего метод `update()`. (Сведения об идентификаторе приложения и идентификаторе издателя см. в разделе ««Задание свойств приложения AIR» на странице 71».) Она также проверяет соответствие строки version строке `version`, передаваемой методу `update()`. Если установка прошла успешно, среда выполнения открывает новую версию приложения. В противном случае (если установка не может быть завершена) снова открывается существующая (до установки) версия приложения.

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.

Если обновленная версия приложения требует обновленной версии среды выполнения, устанавливается новая версия среды выполнения. Для обновления среды выполнения на компьютере необходимы права администратора.

При тестировании приложения в ADL вызов метода `update()` выдает исключение среды выполнения.

О строке version

Строка, представленная параметром `version` метода `update()`, должна соответствовать строке атрибута `version` основного элемента `application` в файле дескриптора приложения устанавливаемого файла AIR. Задание параметра `version` является обязательным в целях защиты. Заставив приложение проверять номер версии в файле AIR, можно предотвратить случайную установку старой версии приложения. (В более старой версии приложения возможны уязвимые места в системе защиты, которые были исправлены в установленном приложении.) Приложение должно сверить строку version в файле AIR со строкой version установленного приложения во избежание атак с установкой устаревшей версии.

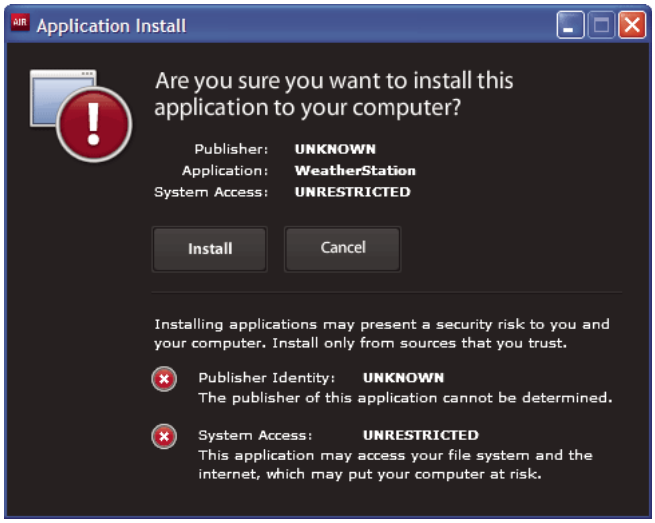
Строка version может иметь любой формат. Например, это может выглядеть как «2.01» или как «version 2». Формат этой строки выбирает разработчик приложения. Среда выполнения не проверяет строку version; это должен сделать код приложения перед его обновлением.

Если приложение Adobe AIR загружает файл AIR из Интернета, рекомендуется иметь механизм, при помощи которого веб-служба может сообщить приложению Adobe AIR номер загружаемой версии. Приложение может затем использовать эту строку в качестве параметра `version` метода `update()`. При получении файла AIR из другого источника, который не предоставляет информацию о версии файла AIR, приложение AIR может проверить файл AIR для получения информации о версии. (Файл AIR является сжатым ZIP-архивом, а файл дескриптора приложения занимает вторую позицию в архиве.)

Дополнительные сведения о файле дескриптора приложения см. в разделе ««Задание свойств приложения AIR» на странице 71».

Использование настраиваемого пользовательского интерфейса обновления приложения

AIR включает в себя интерфейс обновления по умолчанию:



Этот интерфейс используется при первой установке пользователем на компьютер версии приложения. Однако можно задать собственный интерфейс для последующих версий. Если в приложении определяется настраиваемый пользовательский интерфейс обновления, укажите элемент `customUpdateUI` в файле дескриптора приложения для текущего установленного приложения.

```
<customUpdateUI>true</customUpdateUI>
```

После установки приложения и открытия пользователем файла AIR с идентификатором приложения и идентификатором издателя, соответствующими установленному приложению, среда выполнения открывает приложение, а не программу установки приложения AIR по умолчанию. Дополнительные сведения см. в разделе ««Создание заказного пользовательского интерфейса для обновлений приложения» на странице 80».

При выполнении приложения оно может принять решение (при отправке объектом `NativeApplication.nativeApplication` события `load`) об установке обновления (при помощи класса Updater). Если принято решение в пользу обновления, приложение может предоставить пользователю собственный интерфейс установки (который отличается от стандартного интерфейса выполнения).

Загрузка файла AIR на компьютер пользователя

Для использования класса Updater пользователь или приложение должны сначала сохранить файл AIR на компьютер пользователя.

Примечание. *AIR 1.5 содержит инфраструктуру обновления, помогающую разработчикам создавать удобные возможности обновления в приложениях AIR. Использовать эту инфраструктуру гораздо проще, чем напрямую применять метод `update()` класса Update. Дополнительные сведения см. в разделе ««Использование инфраструктуры обновления» на странице 112».*

Следующий код выполняет чтение файла AIR с URL-адреса (http://example.com/air/updates/Sample_App_v2.air) и сохраняет этот файл AIR в каталог хранилища приложения.

Пример ActionScript:

```
var urlString:String = "http://example.com/air/updates/Sample_App_v2.air";
var urlReq:URLRequest = new URLRequest(urlString);
var urlStream:URLStream = new URLStream();
var fileData:ByteArray = new ByteArray();
urlStream.addEventListener(Event.COMPLETE, loaded);
urlStream.load(urlReq);

function loaded(event:Event):void {
    urlStream.readBytes(fileData, 0, urlStream.bytesAvailable);
    writeAirFile();
}

function writeAirFile():void {
    var file:File = File.applicationStorageDirectory.resolvePath("My App v2.air");
    var fileStream:FileStream = new FileStream();
    fileStream.open(file, FileMode.WRITE);
    fileStream.writeBytes(fileData, 0, fileData.length);
    fileStream.close();
    trace("The AIR file is written.");
}
```

Пример JavaScript:

```
var urlString = "http://example.com/air/updates/Sample_App_v2.air";
var urlReq = new air.URLRequest(urlString);
var urlStream = new air.URLStream();
var fileData = new air.ByteArray();
urlStream.addEventListener(air.Event.COMPLETE, loaded);
urlStream.load(urlReq);

function loaded(event) {
    urlStream.readBytes(fileData, 0, urlStream.bytesAvailable);
    writeAirFile();
}

function writeAirFile() {
    var file = air.File.desktopDirectory.resolvePath("My App v2.air");
    var fileStream = new air.FileStream();
    fileStream.open(file, air.FileMode.WRITE);
    fileStream.writeBytes(fileData, 0, fileData.length);
    fileStream.close();
    trace("The AIR file is written.");
}
```

Дополнительные сведения см. в разделе:

- Технологический процесс чтения и записи файлов (для разработчиков ActionScript)
- Технологический процесс чтения и записи файлов (для разработчиков HTML)

Проверка факта первичного запуска приложения

После обновления приложения можно отобразить экран приветствия. При запуске приложение проверяет факт первичного запуска и принимает решение о том, отображать приветствие или нет.

Примечание. *AIR 1.5 содержит инфраструктуру обновления, помогающую разработчикам создавать удобные возможности обновления в приложениях AIR. Эта инфраструктура предоставляет удобные методы проверки, первый ли раз запущена версия приложения. Дополнительные сведения см. в разделе ««Использование инфраструктуры обновления» на странице 112».*

Для этого можно сохранить файл в каталог хранилища приложения при инициализации приложения. При каждом запуске приложения оно должно проверять наличие этого файла. Если файл не существует, то приложение выполняется первый раз для данного пользователя. Существование файла означает, что приложение уже выполнялось по крайней мере один раз. Если файл существует и содержит номер более старой версии, чем номер текущей версии, значит пользователь выполняет новую версию приложения в первый раз.

Следующий пример Flex иллюстрирует данную концепцию:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    title="Sample Version Checker Application"
    applicationComplete="system extension()">
    <mx:Script>
        <![CDATA[
            import flash.filesystem.*;
            public var file:File;
            public var currentVersion:String = "1.2";
            public function system extension():void {
                file = File.applicationStorageDirectory;
                file = file.resolvePath("Preferences/version.txt");
                trace(file.nativePath);
                if(file.exists) {
                    checkVersion();
                } else {
                    firstRun();
                }
            }
            private function checkVersion():void {
                var stream:FileStream = new FileStream();
                stream.open(file, FileMode.READ);
                var reversion:String = stream.readUTFBytes(stream.bytesAvailable);
                stream.close();
                if (reversion != currentVersion) {
                    log.text = "You have updated to version " + currentVersion + ".\n";
                } else {
                    saveFile();
                }
                log.text += "Welcome to the application.";
            }
            private function firstRun():void {
                log.text = "Thank you for installing the application. \n"
                    + "This is the first time you have run it.";
                saveFile();
            }
            private function saveFile():void {
                var stream:FileStream = new FileStream();
                stream.open(file, FileMode.WRITE);
                stream.writeUTFBytes(currentVersion);
                stream.close();
            }
        ]]>
    </mx:Script>
    <mx:TextArea ID="log" width="100%" height="100%" />
</mx:WindowedApplication>
```

В следующем примере показана реализация этой концепции в JavaScript:

```
<html>
    <head>
        <script src="AIRAliases.js" />
        <script>
            var file;
            var currentVersion = "1.2";
            function system extension() {
                file = air.File.appStorageDirectory.resolvePath("Preferences/version.txt");
                air.trace(file.nativePath);
                if(file.exists) {
                    checkVersion();
                } else {
                    firstRun();
                }
            }
            function checkVersion() {
                var stream = new air.FileStream();
                stream.open(file, air.FileMode.READ);
                var reversion = stream.readUTFBytes(stream.bytesAvailable);
                stream.close();
                if (reversion != currentVersion) {
                    window.document.getElementById("log").innerHTML
                            = "You have updated to version " + currentVersion + ".\n";
                } else {
                    saveFile();
                }
                window.document.getElementById("log").innerHTML
                                += "Welcome to the application.";
            }
            function firstRun() {
                window.document.getElementById("log").innerHTML
                            = "Thank you for installing the application. \n"
                            + "This is the first time you have run it.";
                saveFile();
            }
            function saveFile() {
                var stream = new air.FileStream();
                stream.open(file, air.FileMode.WRITE);
                stream.writeUTFBytes(currentVersion);
                stream.close();
            }
        </script>
    </head>
    <body onLoad="system extension()">
        <textarea ID="log" rows="100%" cols="100%" />
    </body>
</html>
```

Если приложение сохраняет данные локально (например, в каталоге хранилища приложения), можно проверить ранее сохраненные данные (предыдущими версиями) при первом запуске.

Использование инфраструктуры обновления

Управление обновлением приложений может быть достаточно сложным. *Инфраструктура обновлений для приложений AdobeAIR* предлагает прикладные интерфейсы программирования, помогающие разработчикам создавать удобные возможности обновления в приложениях AIR. Эти функции инфраструктуры обновления AIR помогают разработчикам решать следующие задачи:

- Периодическая проверка наличия обновлений через заданный интервал времени или по запросу пользователя
- Загрузка файлов AIR (с обновлениями) из интернет-источников
- Предупреждение пользователя о первом запуске вновь установленной версии
- Подтверждение того, что пользователю требуется проверять наличие обновлений
- Отображение информации о новой версии обновления пользователю
- Отображение хода выполнения загрузки и информации об ошибках пользователю

Инфраструктура обновления AIR предлагает пользовательский интерфейс, который может использоваться приложением. Она предоставляет пользователю основную информацию по обновлениям приложений и возможности, связанные с их выполнением. Приложение также может определить собственный настраиваемый пользовательский интерфейс для использования с инфраструктурой обновления.

Инфраструктура обновления AIR позволяет сохранять информацию по обновлению версии приложения AIR в простых файлах конфигурации XML. В большинстве приложений настройка этих файлов конфигурации и включение некоторых простейших программных кодов обеспечивает удобные функции обновления для конечных пользователей.

Даже без использования инфраструктуры обновлений Adobe AIR содержит класс Updater, который можно применять для перехода на новые версии. Этот класс позволяет приложению выполнить обновление до версии, содержащейся в файле AIR на компьютере пользователя. Но управление обновлениями может включать задачи, более сложные, чем обновление из локально сохраненного файла AIR.

Файлы инфраструктуры обновления AIR

Инфраструктура обновления AIR содержится в каталоге frameworks/libs/air пакета AIR 2 SDK. Она включает следующие файлы:

- applicationupdater.swc: определяет основные функциональные возможности библиотеки обновления, используемые в ActionScript. В этой версии отсутствует интерфейс пользователя.
- applicationupdater.swf: определяет основные функциональные возможности библиотеки обновления, используемые в JavaScript. В этой версии отсутствует интерфейс пользователя.
- applicationupdater_ui.swc: определяет основные функциональные возможности библиотеки обновления версии Flex 4, включая интерфейс пользователя, с помощью которого приложение может отображать параметры обновления.
- applicationupdater_ui.swf: определяет основные функциональные возможности библиотеки обновления версии JavaScript, включая интерфейс пользователя, с помощью которого приложение может отображать параметры обновления.
- flex3/applicationupdater_ui.swc: версия файла applicationupdater_ui, используемая в среде Flex 3.

  Важно! При использовании пакета AIR 2 SDK с приложением Flex 3 создайте ссылку на версию файла applicationupdater_ui.swc в подкаталоге flex3. (Или можно заменить версию в каталоге frameworks/libs/air версией в подкаталоге flex3.)

Дополнительные сведения см. в следующих разделах:

- «Настройка среды разработки Flex» на странице 113
- «Включение файлов инфраструктуры в HTML-приложение AIR» на странице 113
- «Простой пример: использование версии ApplicationUpdaterUI (с интерфейсом пользователя)» на странице 114

Настройка среды разработки Flex

В файлах SWC в каталоге frameworks/libs/air пакета AIR 2 SDK определены классы, которые можно использовать при разработке в средах Flex и Flash.

Чтобы использовать инфраструктуру обновления при компиляции с помощью пакета Flex SDK, необходимо включить файл ApplicationUpdater.swc или ApplicationUpdater_UI.swc в вызов компилятора amxmlc. В следующем примере компилятор загружает файл ApplicationUpdater.swc в подкаталог lib пакета Flex SDK.

```
amxmlc -library-path+=lib/ApplicationUpdater.swc  -- myApp.mxml
```

В следующем примере компилятор загружает файл ApplicationUpdater_UI.swc в подкаталог lib пакета Flex SDK.

```
amxmlc -library-path+=lib/ApplicationUpdater_UI.swc  -- myApp.mxml
```

При разработке с помощью Flash Builder добавьте SWC-файл на вкладке «Путь к библиотеке» в настройках путей к библиотекам Flex Builder в диалоговом окне «Свойства».

Обязательно скопируйте SWC-файлы в каталог, который будет указан при вызове компилятора amxmlc (во Flex SDK) или Flash Builder.

Включение файлов инфраструктуры в HTML-приложение AIR

В каталоге frameworks/html инфраструктуры обновления размещаются следующие SWF-файлы:

- ApplicationUpdater.swf — Определяет основные функциональные возможности библиотеки обновления, без использования какого-либо интерфейса пользователя
- ApplicationUpdater_UI.swf — Определяет основные функциональные возможности библиотеки обновления, включая интерфейс пользователя, с помощью которого приложение может отображать параметры обновления

Код JavaScript в приложениях AIR может использовать классы, определенные в SWF-файлах.

Чтобы использовать инфраструктуру обновления, добавьте файл ApplicationUpdater.swf или ApplicationUpdater_UI.swf в каталог приложения (или его подкаталог). Затем включите в HTML-файл, который будет использовать инфраструктуру (в коде JavaScript), тег `script`, загружающий этот файл.

```
<script src="applicationUpdater.swf" type="application/x-shockwave-flash"/>
```

Также можно использовать тег `script` для загрузки файла ApplicationUpdater_UI.swf.

```
<script src="ApplicationUpdater_UI.swf" type="application/x-shockwave-flash"/>
```

Прикладной интерфейс программирования, определенный в этих двух файлах, описывается в оставшейся части данного документа.

Простой пример: использование версии ApplicationUpdaterUI (с интерфейсом пользователя)

Входящая в инфраструктуру обновления версия файла ApplicationUpdater с интерфейсом пользователя предлагает простой интерфейс, который можно использовать в приложении. Далее приведен простейший пример его использования.

Вначале создайте приложение AIR, которое вызывает инфраструктуру обновления.

1 Если это HTML-приложение AIR, загрузите файл ApplicationUpdaterUI.js.

```
<script src="ApplicationUpdater_UI.swf" type="application/x-shockwave-flash"/>
```

2 Инициируйте объект ApplicationUpdaterUI в программной логике приложения AIR.

В ActionScript используйте следующий программный код.

```
var appUpdater:ApplicationUpdaterUI = new ApplicationUpdaterUI();
```

В JavaScript используйте следующий программный код.

```
var appUpdater = new runtime.air.update.ApplicationUpdaterUI();
```

Возможно, потребуется добавить этот код в функцию инициализации, которая выполняется при загрузке приложения.

3 Создайте текстовый файл с именем updateConfig.xml и добавьте к нему следующий код:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://ns.adobe.com/air/framework/update/configuration/1.0">
    <url>http://example.com/updates/update.xml</url>
    <delay>1</delay>
</configuration>
```

Отредактируйте элемент `URL` файла updateConfig.xml в соответствии с фактическим местоположением файла дескриптора на веб-сервере (см. следующую процедуру).

Свойство `delay` указывает число дней, которое приложение ждет, прежде чем выполнить следующую проверку наличия обновлений.

4 Добавьте файл updateConfig.xml к каталогу проекта своего приложения AIR.

5 Проверьте ссылку на файл updateConfig.xml в объекте Updater и вызовите для него метод `initialize()`.

В ActionScript используйте следующий программный код.

```
appUpdater.configurationFile = new File("app:/updateConfig.xml");
appUpdater.initialize();
```

В JavaScript используйте следующий программный код.

```
appUpdater.configurationFile = new air.File("app:/updateConfig.xml");
appUpdater.initialize();
```

6 Создайте вторую версию приложения AIR, отличающуюся номером версии от первого приложения. (Версия указывается в файле дескриптора приложения, в элементе `version`.)

Затем добавьте обновленную версию приложения AIR на веб-сервер.

1 Поместите на веб-сервер обновленную версию файла AIR.

2 Создайте текстовый файл с именем updateDescriptor.xml и добавьте к нему следующий код:

```
<?xml version="1.0" encoding="utf-8"?>
     <update xmlns="http://ns.adobe.com/air/framework/update/description/1.0">
       <version>1.1</version>
       <url>http://example.com/updates/sample_1.1.air</url>
       <description>This is the latest version of the Sample application.</description>
    </update>
```

Отредактируйте значения `version`, `URL` и `description` в файле updateDescriptor.xml для соответствия с обновленным файлом AIR.

3 Добавьте файл updateDescriptor.xml к тому же каталогу на веб-сервере, где находится обновленный файл AIR.

Это простейший пример, но в нем показано, как работает функция обновления, очень важная для многих приложений. В оставшейся части этого документа описывается, как оптимально использовать инфраструктуру обновления.

Другой пример использования инфраструктуры обновления см. в следующих демонстрационных приложениях в центре разработки Adobe AIR:

- Инфраструктура обновления в приложении Flex(http://www.adobe.com/go/learn_air_qs_update_framework_flex_ru)
- Инфраструктура обновления в приложении Flash(http://www.adobe.com/go/learn_air_qs_update_framework_flash_ru)
- Инфраструктура обновления в HTML-приложении(http://www.adobe.com/go/learn_air_qs_update_framework_html_ru)

Определение файла дескриптора обновления и добавление файла AIR на веб-сервер

При использовании инфраструктуры обновления основные сведения о доступном обновлении определяются в файле дескриптора обновления, хранящемся на веб-сервере. Файл дескриптора обновления является простым XML-файлом. Инфраструктура обновления, входящая в приложение, проверяет этот файл, чтобы определить, не выложена ли новая версия.

Файл дескриптора обновления содержит следующие данные:

- `version` — Новая версия приложения AIR. Должна быть та же строка, что используется для указания версии в новом файле дескриптора обновления. Если версия в файле дескриптора обновления не соответствует версии обновленного файла AIR, инфраструктура обновления вызывает исключение.
- `url` — Местоположение обновленного файла AIR. Это файл, который содержит обновленную версию приложения AIR.
- `description` — Подробные сведения о новой версии. Эта информация может отображаться пользователю во время выполнения обновления.

Элементы `version` и `url` обязательны. Элемент `description` не обязателен.

Здесь приведен пример файла дескриптора приложения.

```
<?xml version="1.0" encoding="utf-8"?>
     <update xmlns="http://ns.adobe.com/air/framework/update/description/1.0">
       <version>1.1a1</version>
       <url>http://example.com/updates/sample_1.1a1.air</url>
       <description>This is the latest version of the Sample application.</description>
    </update>
```

Если требуется определить тег `description` для нескольких языков, используйте несколько элементов `text`, определяющих атрибут `lang`.

```
<?xml version="1.0" encoding="utf-8"?>
     <update xmlns="http://ns.adobe.com/air/framework/update/description/1.0">
       <version>1.1a1</version>
       <url>http://example.com/updates/sample_1.1a1.air</url>
       <description>
           <text xml:lang="en">English description</text>
           <text xml:lang="fr">French description</text>
           <text xml:lang="ro">Romanian description</text>
       </description>
    </update>
```

Поместите файл дескриптора обновления вместе с обновленным файлом AIR на веб-сервер.

В создаваемом с дескриптором обновления каталоге templates также располагаются демонстрационные файлы дескриптора обновления. Среди них есть дескрипторы для одного языка и для нескольких языков.

Создание экземпляра объекта Updater

После загрузки инфраструктуры обновления AIR в программном коде (см. разделы ««Настройка среды разработки Flex» на странице 113» и ««Включение файлов инфраструктуры в HTML-приложение AIR» на странице 113») необходимо создать экземпляр объекта, как показано в следующем примере.

Пример ActionScript:

```
var appUpdater:ApplicationUpdater = new ApplicationUpdater();
```

Пример JavaScript:

```
var appUpdater = new runtime.air.update.ApplicationUpdater();
```

В предыдущем примере используется класс ApplicationUpdater (который не предоставляет интерфейса пользователя). Если требуется использовать класс ApplicationUpdaterUI (предоставляющий интерфейс пользователя), код должен быть следующим.

Пример ActionScript:

```
var appUpdater:ApplicationUpdaterUI = new ApplicationUpdaterUI();
```

Пример JavaScript:

```
var appUpdater = new runtime.air.update.ApplicationUpdaterUI();
```

В оставшихся примерах кода в этом документе предполагается, что уже создан экземпляр объекта Updater с именем `appUpdater`.

Настройка параметров обновления

Как класс ApplicationUpdater, так и класс ApplicationUpdaterUI можно настроить с помощью файла конфигурации, поставляемого с приложением, используя ActionScript или JavaScript.

Определение параметров обновления в файле конфигурации XML

Конфигурационный файл обновления — это XML-файл. В нем содержатся следующие элементы:

- `updateURL` — Значение типа String. Задает местоположения дескриптора обновления на удаленном сервере. Разрешено любое допустимое местоположение URLRequest. Необходимо определить свойство `updateURL` либо с помощью файла конфигурации, либо с помощью сценария (см. раздел ««Определение файла дескриптора обновления и добавление файла AIR на веб-сервер» на странице 115»). Необходимо определить это свойство до того, как будет использоваться объект updater (перед вызовом метода `initialize()` для объекта updater, описанного в разделе «Инициализация инфраструктуры обновления» на странице 120).
- `delay` — Значение типа Number. Задает интервал времени в днях (разрешены такие значения, как `0.25`) для проверки наличия обновлений. Значение 0 (задаваемое по умолчанию) означает, что объект updater не выполняет периодической автоматической проверки.

Файл конфигурации для класса ApplicationUpdaterUI может содержать следующий элемент в дополнение к элементам `updateURL` и `delay`.

- `defaultUI`: Список элементов `dialog`. Каждый элемент `dialog` имеет атрибут `name`, соответствующий диалоговому окну в интерфейсе пользователя. Каждый элемент `dialog` имеет атрибут `visible`, определяющий, видимо ли это диалоговое окно. Значение по умолчанию — `true`. Для атрибута `name` возможны следующие значения:
  - `"checkForUpdate"` — Соответствует диалоговым окнам «Проверка обновления», «Нет обновлений» и «Ошибка обновления»
  - `"downloadUpdate"` — Соответствует диалоговому окну «Загрузка обновления»
  - `"downloadProgress"` — Соответствует диалоговым окнам «Выполнение загрузки» и «Ошибка загрузки»
  - `"installUpdate"` — Соответствует диалоговому окну «Установка обновления»
  - `"fileUpdate"` — Соответствует диалоговым окнам «Обновление файла», «Без обновления файла» и «Ошибка файла»
- `"unexpectedError"` — Соответствует диалоговому окну «Непредвиденная ошибка»

  Если установлено значение `false`, соответствующее диалоговое окно не отображается в ходе процедуры обновления.

Здесь приведен пример файла конфигурации для инфраструктуры ApplicationUpdater.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://ns.adobe.com/air/framework/update/configuration/1.0">
      <url>http://example.com/updates/update.xml</url>
      <delay>1</delay>
</configuration>
```

Здесь приведен пример файла конфигурации для инфраструктуры ApplicationUpdaterUI, которая содержит определение элемента `defaultUI`.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://ns.adobe.com/air/framework/update/configuration/1.0">
      <url>http://example.com/updates/update.xml</url>
      <delay>1</delay>
      <defaultUI>
         <dialog name="checkForUpdate" visible="false" />
         <dialog name="downloadUpdate" visible="false" />
         <dialog name="downloadProgress" visible="false" />
      </defaultUI>
</configuration>
```

Укажите местоположение этого файла в свойстве `configurationFile`:

Пример ActionScript:

```
appUpdater.configurationFile = new File("app:/cfg/updateConfig.xml");
```

Пример JavaScript:

```
appUpdater.configurationFile = new air.File("app:/cfg/updateConfig.xml");
```

Каталог templates инфраструктуры обновления содержит демонстрационный файл конфигурации config-template.xml.

Код ActionScript или JavaScript для определения параметров обновления

Эти параметры конфигурации также могут быть установлены с помощью программного кода приложения, как в следующем примере:

```
appUpdater.updateURL = " http://example.com/updates/update.xml";
appUpdater.delay = 1;
```

Свойствами объекта updater являются `updateURL` и `delay`. Эти свойства определяют те же параметры, что и элементы `updateURL` и `delay` в файле конфигурации: адрес URL файла дескриптора обновления и интервал времени между проверками обновлений. Если в программном коде указывается файл конфигурации *и* параметры, то любое свойство, установленное в коде, переопределяет соответствующие параметры файла конфигурации.

Необходимо определить свойство `updateURL` либо с помощью файла конфигурации, либо с помощью сценария (см. раздел ««Определение файла дескриптора обновления и добавление файла AIR на веб-сервер» на странице 115») прежде, чем использовать объект updater (до вызова метода `initialize()` объекта updater, как описано в разделе ««Инициализация инфраструктуры обновления» на странице 120»).

Инфраструктура ApplicationUpdaterUI определяет эти дополнительные свойства объекта updater следующим образом:

- `isCheckForUpdateVisible` — Соответствует диалоговым окнам «Проверка обновления», «Нет обновлений» и «Ошибка обновления»
- `isDownloadUpdateVisible` — Соответствует диалоговому окну «Загрузка обновления»
- `isDownloadProgressVisible` — Соответствует диалоговым окнам «Выполнение загрузки» и «Ошибка загрузки»
- `isInstallUpdateVisible` — Соответствует диалоговому окну «Установка обновления»
- `isFileUpdateVisible` — Соответствует диалоговым окнам «Обновление файла», «Без обновления файла» и «Ошибка файла»
- `isUnexpectedErrorVisible` — Соответствует диалоговому окну «Непредвиденная ошибка»

Каждое свойство соответствует одному или нескольким диалоговым окнам в интерфейсе пользователя ApplicationUpdaterUI. Каждое свойство является логическим значением, для которого по умолчанию задано значение `true`. Если установлено значение `false`, соответствующие диалоговые окна не отображаются в ходе процедуры обновления.

Эти свойства диалоговых окон переопределяют параметры, установленные в файле конфигурации обновления.

Процесс обновления

Инфраструктура обновления AIR выполняет процесс обновления, используя следующие шаги:

1. При инициализации объекта updater выполняется проверка, определяющая, проверялось ли наличие обновлений в течение заданного интервала задержки (см. раздел ««Настройка параметров обновления» на странице 116»). Если проверка наличия обновлений только предстоит, процесс обновления продолжается.
2. Объект updater загружает и интерпретирует файл дескриптора обновления.
3. Объект updater загружает обновленный файл AIR.
4. Объект updater устанавливает обновленную версию приложения.

Объект updater отправляет соответствующие события при завершении каждого из описанных шагов. В версии ApplicationUpdater (без интерфейса пользователя) можно отменить события, сообщающие об успешном выполнении шагов этого процесса. Если какое-то из этих событий отменяется, то отменяется и следующий шаг в процессе. В версии ApplicationUpdaterUI (с интерфейсом пользователя) объект updater открывает диалоговые окна, позволяющие пользователю отменить или выполнить каждый шаг в процессе обновления.

Если отменяется событие, можно вызвать соответствующие методы объекта updater для возобновления процесса.

По мере того как версия ApplicationUpdater объекта updater выполняет различные этапы процесса обновления, она фиксирует его текущее состояние в свойстве `currentState`. Для этого свойства устанавливается переменная типа string со следующими возможными значениями:

- `"UNINITIALIZED"` — Объект updater не был инициализирован.
- `"INITIALIZING"` — Объект updater инициализируется.
- `"READY"` — Объект updater инициализирован
- `"BEFORE_CHECKING"` — Объект updater еще не выполнил поиск файла дескриптора обновления.
- `"CHECKING"` — Объект updater выполняет поиск файла дескриптора обновления.
- `"AVAILABLE"` — Файл дескриптора обновления доступен.
- `"DOWNLOADING"` — Объект updater выполняет загрузку файла AIR.
- `"DOWNLOADED"` — Объект updater выполнил загрузку файла AIR.
- `"INSTALLING"` — Объект updater выполняет установку файла AIR.
- `"PENDING_INSTALLING"` — Объект updater инициализирован и есть неоконченные обновления.

Некоторые методы объекта updater выполняются только в определенном состоянии этого объекта.

Инициализация инфраструктуры обновления

После установки параметров конфигурации (см. раздел ««Простой пример: использование версии ApplicationUpdaterUI (с интерфейсом пользователя)» на странице 114»), вызовите метод `initialize()` для инициализации обновления.

```
appUpdater.initialize();
```

Этот метод выполняет следующие действия:

- Инициализируется инфраструктура обновления, автоматически в синхронном режиме устанавливаются все незавершенные обновления. Необходимо вызывать этот метод во время запуска приложения, поскольку при его вызове приложение может быть перезапущено.
- Выполняется проверка, нет ли отложенных обновлений, а затем завершается их установка.
- Если в ходе процесса обновления происходит ошибка, файл обновления и информация о версии удаляются из области хранения приложения.
- Если истекает время задержки, начинается процесс обновления. Иначе происходит перезапуск таймера.

Вызов этого метода может привести к отправке следующих событий объектом updater:

- `UpdateEvent.INITIALIZED` — Событие отправляется, когда выполнена инициализация.
- `ErrorEvent.ERROR` — Событие отправляется, если в ходе инициализации произошла ошибка.

После отправки события `UpdateEvent.INITIALIZED` процесс обновления завершается.

При вызове метода `initialize()` объект updater запускает процесс обновления и выполняет все шаги, учитывая параметр задержки таймера. Однако также можно запустить процесс обновления в любой момент времени, вызвав метод `checkNow()` объекта updater:

```
appUpdater.checkNow();
```

Этот метод не делает ничего, если процесс обновления уже запущен. Иначе происходит запуск процесса обновления.

Объект updater может отправлять следующие события в результате вызова метода `checkNow()`:

- Событие `UpdateEvent.CHECK_FOR_UPDATE` отправляется точно перед тем, как будет предпринята попытка загрузить файл дескриптора обновления.

Если отменяется событие `checkForUpdate`, можно вызвать метод `checkForUpdate()` объекта updater. (См. следующий раздел.) Если событие не отменяется, процесс обновления выполняет проверку файла дескриптора обновления.

Управление процессом обновления в версии ApplicationUpdaterUI (с интерфейсом пользователя)

В версии ApplicationUpdaterUI (с интерфейсом пользователя) пользователь может отменить процесс с помощью кнопки «Отмена» в диалоговом окне интерфейса пользователя. Также процесс обновления можно отменить программно, вызвав метод `cancelUpdate()` объекта ApplicationUpdaterUI.

Можно задать свойства объекта ApplicationUpdaterUI или определить элементы в файле конфигурации обновления, указав какие подтверждения в диалоговых окнах будут отображаться объектом updater. Дополнительные сведения см. в разделе ««Настройка параметров обновления» на странице 116».

Управление процессом обновления в версии ApplicationUpdater (без интерфейса пользователя)

Можно вызвать метод `preventDefault()` объектов событий, созданных объектом ApplicationUpdater, чтобы отменить шаги процесса обновления (см. раздел ««Процесс обновления» на странице 119»). Отмена заданного по умолчанию поведения позволяет приложению отобразить сообщение пользователю, запросив у него подтверждение выполнения.

В следующих разделах описано, как продолжить процесс выполнения, если какой-либо шаг этого процесса был отменен.

Загрузка и интерпретирование файла дескриптора обновления

Объект ApplicationUpdater отправляет событие `checkForUpdate` перед началом процесса обновления, до того момента, как объект updater попытается загрузить файл дескриптора обновления. Если отменяется заданное по умолчанию поведение события `checkForUpdate`, то объект updater не загружает файл дескриптора обновления. Можно вызвать метод `checkForUpdate()`, чтобы возобновить процесс обновления:

```
appUpdater.checkForUpdate();
```

Вызов метода `checkForUpdate()` заставляет объект updater загружать и интерпретировать файл дескриптора обновления в асинхронном режиме. В результате вызова метода `checkForUpdate()` объект updater может отправлять следующие события:

- `StatusUpdateEvent.UPDATE_STATUS` — Объект updater успешно загрузил и интерпретировал файл дескриптора обновления. Это событие имеет следующие свойства:
  - `available` — Логическое значение. Установите значение `true`, если доступна другая версия, кроме текущей версии приложения; иначе устанавливается значение `false` (версии совпадают).
  - `version` — Значение типа String. Номер версии в файле дескрипторе приложения из файла обновления
  - `details` — Массив. Если нет локализованной версии описания, этот массив возвращает пустую (`""`) в качестве первого элемента и описание в качестве второго элемента.

    При наличии нескольких версий описания (в файле дескриптора обновления), этот массив содержит несколько подмассивов. Каждый массив содержит два элемента: первый с кодом языка (таким, как `"en"`), а второй с соответствующим описанием (значением типа String) для этого языка. См. раздел ««Определение файла дескриптора обновления и добавление файла AIR на веб-сервер» на странице 115».
- `StatusUpdateErrorEvent.UPDATE_ERROR` — произошла ошибка, и объект updater не смог загрузить и интерпретировать файл дескриптора обновления.

Загрузка AIR-файла обновления

Объект ApplicationUpdater отправляет событие `updateStatus` после того, как объект updater успешно загрузит и интерпретирует файл дескриптора обновления. По умолчанию, если доступен файл обновления, должна запускаться его загрузка. Если поведение по умолчанию отменено, можно вызвать метод `downloadUpdate()`, чтобы возобновить процесс обновления.

```
appUpdater.downloadUpdate();
```

Вызов этого метода заставляет объект updater асинхронно загружать обновленную версию AIR-файла.

Метод `downloadUpdate()` может отправлять следующие события:

- `UpdateEvent.DOWNLOAD_START` — Установлено соединение с сервером. При использовании библиотеки ApplicationUpdaterUI это событие открывает диалоговое окно с индикатором выполнения для отслеживания текущего состояния загрузки.
- `ProgressEvent.PROGRESS` — Отправляется периодически, по мере выполнения загрузки файла.
- `DownloadErrorEvent.DOWNLOAD_ERROR` — Отправляется, если произошла ошибка при подключении или загрузке файла обновления. Также отправляется в случае недопустимых HTTP-состояний (например, «404 — Файл не найден»). У этого события есть свойство `errorID`, целое значение, определяющее дополнительную информацию об ошибке. Дополнительное свойство `subErrorID` может содержать дополнительные сведения об ошибке.
- `UpdateEvent.DOWNLOAD_COMPLETE` — Объект updater успешно загрузил и интерпретировал файл дескриптора обновления. Если это событие не отменено, в версии ApplicationUpdater (без интерфейса пользователя) продолжается установка обновления. В версии ApplicationUpdaterUI (с интерфейсом пользователя) пользователю предлагается диалоговое окно, позволяющее им подтвердить продолжение обновления.

Обновление приложения

Объект ApplicationUpdater отправляет событие `downloadComplete` после завершения загрузки файла обновления. Если поведение по умолчанию отменено, можно вызвать метод `installUpdate()`, чтобы возобновить процесс обновления:

```
appUpdater.installUpdate(file);
```

Вызов этого метода заставляет объект updater устанавливать обновленную версию AIR-файла. Этот метод включает один параметр, `file`, который является объектом File и содержит ссылку на AIR-файл, используемый для обновления.

Объект ApplicationUpdater может отправлять событие `beforeInstall` в результате вызова метода `installUpdate()`.

- `UpdateEvent.BEFORE_INSTALL` — Отправляется непосредственно перед установкой обновления. Иногда бывает полезно приостановить немедленную установку обновления, чтобы пользователь мог завершить текущую работу перед началом процесса обновления. Вызов метода `preventDefault()` объекта Event откладывает установку до следующего перезапуска, при этом никакой дополнительный процесс обновления не может быть запущен. (Сюда входят обновления, сведения о которых появляются в результате вызова метода `checkNow()` или в результате периодических проверок.)

Установка из произвольно выбранного AIR-файла

Можно вызвать метод `installFromAIRFile()` для установки обновленной версии из AIR-файла на компьютере пользователя.

```
appUpdater.installFromAIRFile();
```

Этот метод заставляет объект updater устанавливать обновленную версию приложения из указанного AIR-файла.

Метод `installFromAIRFile()` может отправлять следующие события:

- `StatusFileUpdateEvent.FILE_UPDATE_STATUS` — Отправляется после того, как ApplicationUpdater подтвердит успешную отправку файла с помощью метода `installFromAIRFile()`. Это событие обладает следующими свойствами.
  - `available` — Установите значение `true`, если доступна другая версия, кроме текущей версии приложения; иначе устанавливается значение `false` (версии совпадают).
  - `version` — В этой переменной типа string указывается новая доступная версия.
  - `path` — Передает исходный путь к файлу обновления.

  Это событие можно отменить, если для свойства Available объекта StatusFileUpdateEvent установлено значение `true`. Отмена этого события отменяет процесс обновления. Вызовите метод `installUpdate()`, чтобы продолжить отмененное обновление.
- `StatusFileUpdateErrorEvent.FILE_UPDATE_ERROR` — произошла ошибка и объект updater не смог установить приложение AIR.

Отмена процесса обновления

Можно вызвать метод `cancelUpdate()`, чтобы отменить процесс обновления.

```
appUpdater.cancelUpdate();
```

Этот метод отменяет любые незавершенные загрузки, удаляет любые незавершенные файлы загрузки и перезапускает таймер периодических проверок.

Этот метод не выполняет никаких действий, если инициализирован объект updater.

Локализация интерфейса ApplicationUpdaterUI

Класс ApplicationUpdaterUI обеспечивает интерфейс для выполнения процесса обновления, предоставляемый пользователю по умолчанию. Этот интерфейс пользователя содержит диалоговые окна, позволяющие запускать и отменять процесс, а также выполнять связанные действия.

Элемент `description` файла дескриптора обновления позволяет определить описание приложения на нескольких языках. Можно использовать несколько элементов `text`, определяющих атрибуты `lang`, как показано в следующем примере:

```
<?xml version="1.0" encoding="utf-8"?>
     <update xmlns="http://ns.adobe.com/air/framework/update/description/1.0">
       <version>1.1a1</version>
       <url>http://example.com/updates/sample_1.1a1.air</url>
       <description>
           <text xml:lang="en">English description</text>
           <text xml:lang="fr">French description</text>
           <text xml:lang="ro">Romanian description</text>
       </description>
    </update>
```

Инфраструктура обновления использует описание, которое лучше соответствует цепочке языковых настроек пользователя. Дополнительные сведения см. в разделе «Определение файла дескриптора обновления и добавление файла AIR на веб-сервер».

Разработчики Flex могут добавить поддержку нового языка непосредственно к пакету `"ApplicationUpdaterDialogs"`.

Разработчики JavaScript могут вызвать метод `addResources()` объекта updater. Этот метод динамически добавляет новый пакет ресурсов для языка. Этот пакет ресурсов определяет локализованные строки для языка. Эти строки используются в различных текстовых полях диалоговых окон.

Разработчики JavaScript могут использовать свойство `localeChain` класса ApplicationUpdaterUI для определения цепочки локалей, используемой интерфейсом пользователя. Обычно разработчики JavaScript (HTML) используют это свойство. Разработчики Flex могут применять ResourceManager для управления цепочкой локалей.

Например, в следующем коде JavaScript определяются пакеты ресурсов для румынского и венгерского языков:

```
appUpdater.addResources("ro_RO",
                 {titleCheck: "Titlu", msgCheck: "Mesaj", btnCheck: "Buton"});
appUpdater.addResources("hu", {titleCheck: "Cím", msgCheck: "Üzenet"});
var languages = ["ro", "hu"];
languages = languages.concat(air.Capabilities.languages);
var sortedLanguages = air.Localizer.sortLanguagesByPreference(languages,
                    air.Capabilities.language,
                    "en-US");
sortedLanguages.push("en-US");
appUpdater.localeChain = sortedLanguages;
```

Дополнительные сведения см. в описании метода `addResources()` класса ApplicationUpdaterUI в справочнике ActionScript® 3.0 для Adobe® Flash® Professional CS5.

Глава 15. Считывание параметров приложения

В среде выполнения можно получить свойства файла дескриптора приложения, а также идентификатор издателя приложения. Эти сведения заданы в свойствах `applicationDescriptor` и `publisherID` объекта NativeApplication.

Чтение файла дескриптора приложения

Считывание файла дескриптора выполняющегося в данный момент приложения (как объекта XML) можно произвести путем получения свойства `applicationDescriptor` объекта NativeApplication.

Пример ActionScript 3.0:

```
var appXml:XML = NativeApplication.nativeApplication.applicationDescriptor;
```

Пример JavaScript:

```
var appXml:XML = air.ativeApplication.nativeApplication.applicationDescriptor;
```

В ActionScript 3.0 затем можно получить доступ к данным дескриптора приложения как к объекту XML (E 4X), например.

```
var appXml:XML = NativeApplication.nativeApplication.applicationDescriptor;
var ns:Namespace = appXml.namespace();
var appId = appXml.ns::id[0];
var appVersion = appXml.ns::version[0];
var appName = appXml.ns::filename[0];
air.trace("appId:", appId);
air.trace("version:", appVersion);
air.trace("filename:", appName);
var xmlString = air.NativeApplication.nativeApplication.applicationDescriptor;
```

В JavaScript при помощи объекта DOMParser можно произвести анализ этих данных, как в следующем примере:

```
var xmlString = air.NativeApplication.nativeApplication.applicationDescriptor;
var appXml = new DOMParser();
var xmlobject = appXml.parseFromString(xmlString, "text/xml");
var root = xmlobject.getElementsByTagName('application')[0];
var appId = root.getElementsByTagName("id")[0].firstChild.data;
var appVersion = root.getElementsByTagName("version")[0].firstChild.data;
var appName = root.getElementsByTagName("filename")[0].firstChild.data;
air.trace("appId:", appId);
air.trace("version:", appVersion);
air.trace("filename:", appName);
```

Дополнительные сведения см. в разделе ««Структура файла дескриптора приложения» на странице 71».

Определение идентификаторов приложения и издателя

Идентификаторы приложения и издателя — это уникальные идентификаторы приложения AIR. Идентификатор приложения указывается в элементе `<id>` дескриптора приложения. Идентификатор издателя извлекается из сертификата, при помощи которого подписан пакет установки AIR.

Идентификатор приложения может быть считан из свойства `id` объекта NativeApplication, как показано в следующем коде.

Пример ActionScript 3.0:

```
trace(NativeApplication.nativeApplication.applicationID);
```

Пример JavaScript:

```
air.trace(air.NativeApplication.nativeApplication.applicationID);
```

Идентификатор издателя может быт считан из свойства `publisherID` объекта NativeApplication.

Пример ActionScript 3.0:

```
trace(NativeApplication.nativeApplication.publisherID);
```

Пример ActionScript 3.0:

```
air.trace(air.NativeApplication.nativeApplication.publisherID);
```

Примечание. *При выполнении приложения AIR в ADL оно не имеет идентификатора издателя, если он не был временно присвоен при помощи флага `-pubID` в командной строке ADL.*

Идентификатор издателя установленного приложения можно также найти в файле `META-INF/AIR/publisherid` в каталоге установки приложения.

Дополнительные сведения см. в разделе ««Об идентификаторах издателя AIR» на странице 98».

Глава 16. Просмотр исходного кода

Подобно тому как пользователь может просматривать исходный код для HTML-страницы в веб-обозревателе, пользователи могут просматривать исходный код HTML-приложений AIR. Adobe® AIR® SDK включает файл AIRSourceViewer.js на языке JavaScript, который можно использовать в приложении для удобной демонстрации исходного кода конечным пользователям.

Загрузка, настройка и открытие объекта просмотра исходного кода

Код объекта просмотра исходного кода содержится в файле JavaScript (AIRSourceViewer.js), который находится в каталоге frameworks пакета AIR SDK. Чтобы использовать объект просмотра исходного кода в своем приложении, скопируйте файл AIRSourceViewer.js в каталог проекта приложения и загрузите файл с помощью тега скрипта в основной HTML-файл приложения.

```
<script type="text/javascript" src="AIRSourceViewer.js"></script>
```

Файл AIRSourceViewer.js определяет класс SourceViewer, к которому можно обращаться из кода JavaScript путем вызова `air.SourceViewer`.

Класс SourceViewer определяет три метода: `getDefault()`, `setup()` и `viewSource()`.

Метод	Описание
`getDefault()`	Статический метод. Запускает экземпляр SourceViewer, который можно использовать для вызова других методов.
`setup()`	Применяет настройки конфигурации к объекту просмотра исходного кода. Дополнительные сведения см. в разделе ««Настройка объекта просмотра исходного кода» на странице 127»
`viewSource()`	Открывает новое окно, в котором пользователь может выполнить поиск и открыть файлы источника приложения.

Примечание. *Код, использующий класс SourceViewer, должен находиться в изолированной программной среде приложения (в файле, находящемся в каталоге приложения).*

Например, следующий код JavaScript создает экземпляр объекта SourceViewer и открывает окно просмотра кода источника, в котором перечислены все файлы источников:

```
var viewer = air.SourceViewer.getDefault();
viewer.viewSource();
```

Настройка объекта просмотра исходного кода

Применение метода `config()` передает настройки объекту просмотра исходного кода. Этот метод использует один параметр: `configObject`. Объект `configObject` содержит свойства, определяющие настройки конфигурации для объекта просмотра исходного кода. К этим свойствам относятся `default`, `exclude`, `initialPosition`, `modal`, `typesToRemove` и `typesToAdd`.

default
Строка, указывающая относительный путь к исходному файлу, который будет отображаться в объекте просмотра исходного кода.

Например, следующий код JavaScript открывает окно Source Viewer с файлом index.html в качестве отображаемого исходного файла:

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.default = "index.html";
viewer.viewSource(configObj);
```

exclude
Массив строк, в котором указаны файлы или каталоги, которые будут исключены из списка отображаемых в объекте просмотра исходного кода. Эти пути указываются относительного каталога приложения. Подстановочные знаки не поддерживаются.

Например, следующий код JavaScript открывает окно объекта просмотра исходного кода и выводит список всех файлов источников за исключением файла AIRSourceViewer.js и файлов в подкаталогах Images и Sounds:

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.exclude = ["AIRSourceViewer.js", "Images" "Sounds"];
viewer.viewSource(configObj);
```

initialPosition
Массив, в каждой строке которого содержатся два числа, указывающие исходные координаты X и Y для окна объекта просмотра исходного кода.

Например, следующий код JavaScript открывает окно просмотра исходного кода с координатами на экране [40, 60] (X = 40, Y = 60):

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.initialPosition = [40, 60];
viewer.viewSource(configObj);
```

modal
Логическое значение, указывающее, должно ли окно объекта просмотра исходного кода быть модальным (TRUE) или немодальным (FALSE). По умолчанию окно объект просмотра исходного кода модально.

Например, в следующем коде JavaScript окно объекта просмотра исходного кода открывается таким образом, что пользователь может взаимодействовать как с этим окном объекта просмотра исходного кода, так и с любыми другими окнами приложений:

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.modal = false;
viewer.viewSource(configObj);
```

typesToAdd
Массив строк, указывающий типы файлов, которые включены в список объекта просмотра исходного кода в дополнение к заданным по умолчанию типам.

По умолчанию в списке объекта просмотра исходного кода указаны следующие типы файлов:

- Текстовые файлы: TXT, XML, MXML, HTM, HTML, JS, AS, CSS, INI, BAT, PROPERTIES, CONFIG
- Файлы изображений: JPG, JPEG, PNG, GIF

  Если значений не указано, в список включены все заданные по умолчанию типы (за исключением тех, что указаны в свойстве `typesToExclude`).

  Например, следующий код JavaScript открывает окно объекта просмотра исходного кода и добавляет к списку файлы VCF и VCARD:

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.typesToAdd = ["text.vcf", "text.vcard"];
viewer.viewSource(configObj);
```

  Для каждого типа файлов, добавляемого в список, необходимо указать «text» (для типов текстовых файлов) или «image» (для типов фалов изображений).

typesToExclude
Массив строк, указывающий типы файлов, которые будут исключены из списка объекта просмотра исходного кода.

По умолчанию в списке объекта просмотра исходного кода указаны следующие типы файлов:

- Текстовые файлы: TXT, XML, MXML, HTM, HTML, JS, AS, CSS, INI, BAT, PROPERTIES, CONFIG
- Файлы изображений: JPG, JPEG, PNG, GIF

  Например, следующий код JavaScript открывает окно объекта просмотра исходного кода, исключая из списка файлы GIF или XML:

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.typesToExclude = ["image.gif", "text.xml"];
viewer.viewSource(configObj);
```

  Для каждого типа файлов, добавляемого в список, необходимо указать `text` (для типов текстовых файлов) или `image` (для типов фалов изображений).

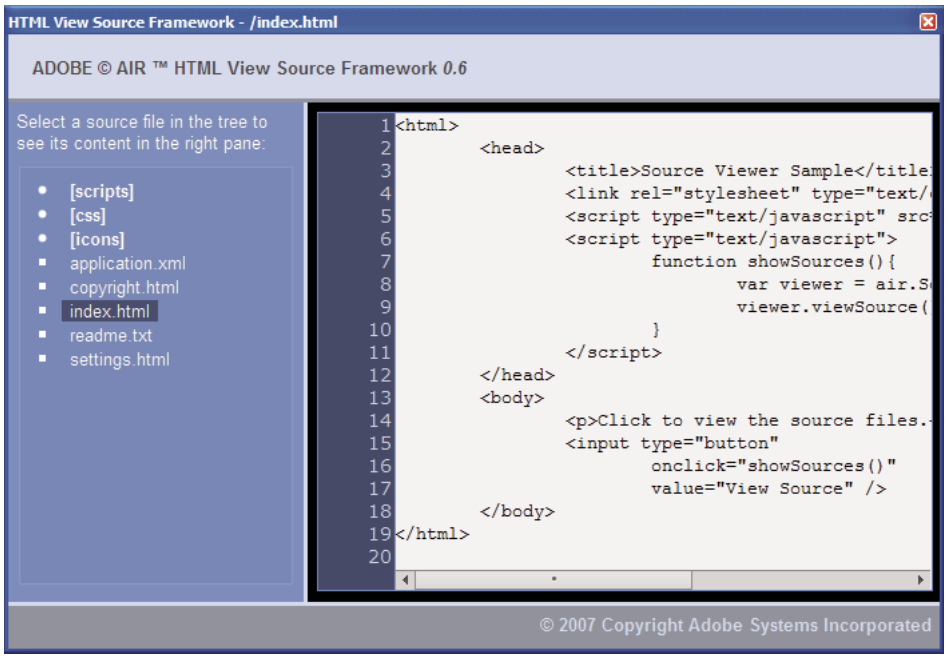
Открытие объекта просмотра исходного кода

Необходимо также добавить элемент пользовательского интерфейса, например ссылку, кнопку или команду меню, которые вызывают код объекта просмотра исходного кода при нажатии их пользователем. Например, в следующем простейшем приложении объект просмотра исходного кода открывается после того, как пользователь щелкает ссылку:

```
<html>
    <head>
        <title>Source Viewer Sample</title>
        <script type="text/javascript" src="AIRSourceViewer.js"></script>
        <script type="text/javascript">
            function showSources(){
                var viewer = air.SourceViewer.getDefault();
                viewer.viewSource()
            }
        </script>
    </head>
    <body>
        <p>Click to view the source files.</p>
        <input type="button"
            onclick="showSources()"
            value="View Source" />
    </body>
</html>
```

Интерфейс пользователя для объекта просмотра исходного кода

Когда приложение вызывает метод `viewSource()` объекта SourceViewer, приложение AIR открывает окно просмотра исходного кода. В окне отображаются список файлов источников и каталогов (слева), а также область просмотра исходного кода для выбранного файла (справа).



Каталоги в списке выделяются скобками. Пользователь может щелкнуть фигурную скобку, чтобы развернуть или свернуть список данного каталога.

Объект просмотра исходного кода может отображать источники для текстовых файлов с известными расширениями (такими как HTML, HTML, JS, TXT, XML и т.п.) или для файлов изображений с известными расширениями (JPG, JPEG, PNG и GIF). Если пользователь выбирает файл, расширение которого не известно, отображается сообщение об ошибке («Не удается извлечь текстовое содержимое из файла данного типа»).

Любые файлы источников, которые исключены при помощи метода `setup()`, не указываются в списке (см. раздел ««Загрузка, настройка и открытие объекта просмотра исходного кода» на странице 127»).

Глава 17. Отладка с помощью AIR HTML Introspector

Пакет Adobe® AIR® SDK включает файл JavaScript, AIRIntrospector.js, который можно включить в приложение для помощи при откладке HTML-приложений.

О программе AIR Introspector

Программа Introspector для HTML/JavaScript-приложений Adobe AIR (называемая также AIR HTML Introspector) обеспечивает следующие полезные функциональные возможности, помогающие при разработке и отладке HTML-приложений.

- Включает инструмент самоанализа, который позволяет указать элемент пользовательского интерфейса в приложении и просмотреть его свойства разметки и DOM.
- Включает консоль для отправки ссылки на объекты для анализа, где можно настроить правильные значения и выполнить код JavaScript. Также можно сериализовать объекты при выводе их на консоль, что ограничит возможность редактирования данных. Кроме того, можно скопировать и сохранить текст с консоли.
- Консоль содержит три представления для свойств и функций DOM.
- Они позволяют редактировать атрибуты и текстовые узлы для элементов DOM.
- В консоли отображаются списки ссылок, стилей CSS, изображений и файлов JavaScript, загруженных в приложение.
- Можно просматривать HTML-источник и текущий источник разметки для интерфейса пользователя.
- Позволяет обращаться к файлам в каталоге приложения. (Эта возможность доступна только в консоли AIR HTML Introspector, открытой в изолированной программной среде приложения. Она недоступна для консолей, открытых для содержимого не в изолированной программной среде приложения.)
- Консоль включает средство просмотра объектов XMLHttpRequest и их свойств, включая свойства `responseText` и `responseXML` (если они доступны).
- Можно выполнить поиск соответствующего критериям текста в коде источника и в файлах.

Загрузка кода AIR Introspector

Код AIR Introspector содержится в файле JavaScript (AIRIntrospector.js), который находится в каталоге frameworks пакета AIR SDK. Чтобы использовать AIR Introspector в своем приложении, скопируйте файл AIRIntrospector.js в каталог проекта приложения и загрузите файл с помощью тега скрипта в основной HTML-файл приложения:

```
<script type="text/javascript" src="AIRIntrospector.js"></script>
```

Также включите этот файл в любой HTML-файл, соответствующий различным исходным окнам приложения.

Важная информация. *Включайте файл AIRIntrospector.js только при развертывании и отладке приложения. Удалите его из упакованного приложения AIR перед его распространением.*

Файл AIRIntrospector.js определяет класс Console, к которому можно обращаться из кода JavaScript путем вызова `air.Introspector.Console`.

Примечание. *Код, использующий AIR Introspector, должен находиться в изолированной программной среде приложения (в файле, находящемся в каталоге приложения).*

Проверка объекта на вкладке Console (Консоль)

Класс Console определяет пять методов: `log()`, `warn()`, `info()`, `error()` и `dump()`.

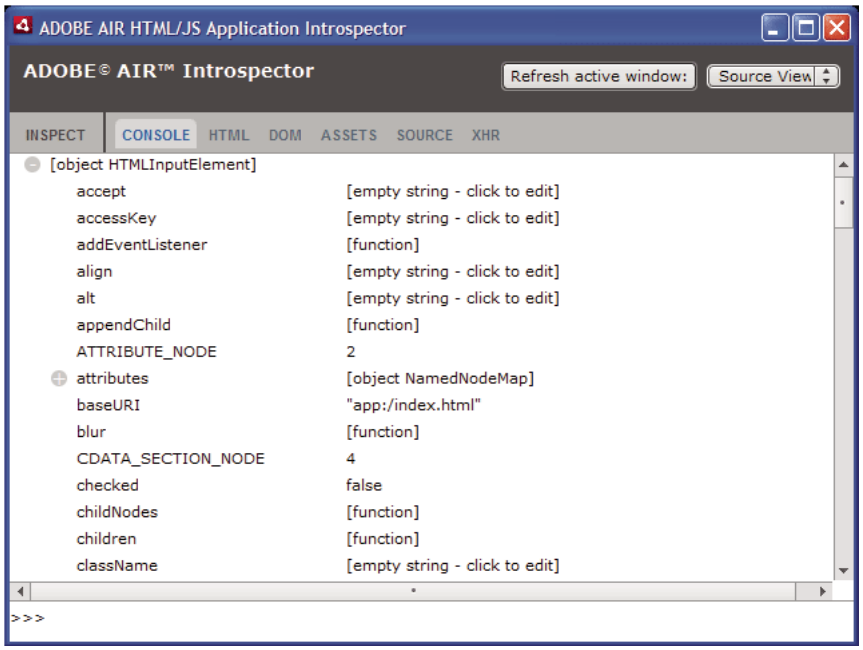
Методы `log()`, `warn()`, `info()` и `error()` все позволяют отправлять объект на вкладку Console (Консоль). Наиболее простым из этих методов является `log()`. В следующем примере простой объект, представленный переменной `test`, передается на вкладку Console (Консоль):

```
var test = "hello";
air.Introspector.Console.log(test);
```

Но гораздо полезнее отправлять на вкладку Console (Консоль) сложные объекты. Например, следующая страница HTML содержит кнопку (`btn1`), которая вызывает функцию, отправляющую сам объект кнопки на вкладку Console (Консоль):

```
<html>
    <head>
        <title>Source Viewer Sample</title>
        <script type="text/javascript" src="scripts/AIRIntrospector.js"></script>
        <script type="text/javascript">
            function logBtn()
            {
                var button1 = document.getElementById("btn1");
                air.Introspector.Console.log(button1);
            }
        </script>
    </head>
    <body>
        <p>Click to view the button object in the Console.</p>
        <input type="button" id="btn1"
            onclick="logBtn()"
            value="Log" />
    </body>
</html>
```

При нажатии этой кнопки на вкладке Console (Консоль) отображается объект btn1, можно развернуть древовидное представление этого объекта для проверки его свойств:



Можно изменить свойство объекта, щелкнув запись справа от имени свойства и изменив эту текстовую запись.

Методы `info()`, `error()` и `warn()` сходны с методом `log()`. Но при вызове этих методов на вкладке Console (Консоль) отображается значок в начале строки:

Метод	**Значок**
`info()`	
`error()`	
`warn()`	

Методы `log()`, `warn()`, `info()` и `error()` отправляют ссылку только на фактический объект, поэтому доступны только те свойства, которые актуальны на момент просмотра. Если требуется сериализовать этот фактический объект, используйте метод `dump()`. Метод содержит два параметра:

Параметр	**Описание**
`dumpObject`	Объект для сериализации.
`levels`	Максимальное число уровней, которые будут проверяться в дереве объекта (помимо корневого уровня). Значение по умолчанию 1 (означает, что показывается только один уровень за корневым уровнем дерева). Этот параметр не обязателен.

Вызов метода `dump()` сериализует объект перед его отправкой на вкладку Console (Консоль), поэтому свойства этого объекта нельзя редактировать. Например, рассмотрим следующий код:

```
var testObject = new Object();
testObject.foo = "foo";
testObject.bar = 234;
air.Introspector.Console.dump(testObject);
```

При выполнении этого кода на консоли отображается объект `testObject` и его свойства, но значения свойств редактировать нельзя.

Настройка AIR Introspector

Можно настроить консоль, установив свойства глобальной переменной `AIRIntrospectorConfig`. Например, в следующем коде JavaScript выполняется настройка AIR Introspector для перехода на новую строку в столбцах через 100 символов:

```
var AIRIntrospectorConfig = new Object();
AIRIntrospectorConfig.wrapColumns = 100;
```

Убедитесь, что свойства переменной `AIRIntrospectorConfig` установлены до загрузки файла AIRIntrospector.js (с помощью тега `script`).

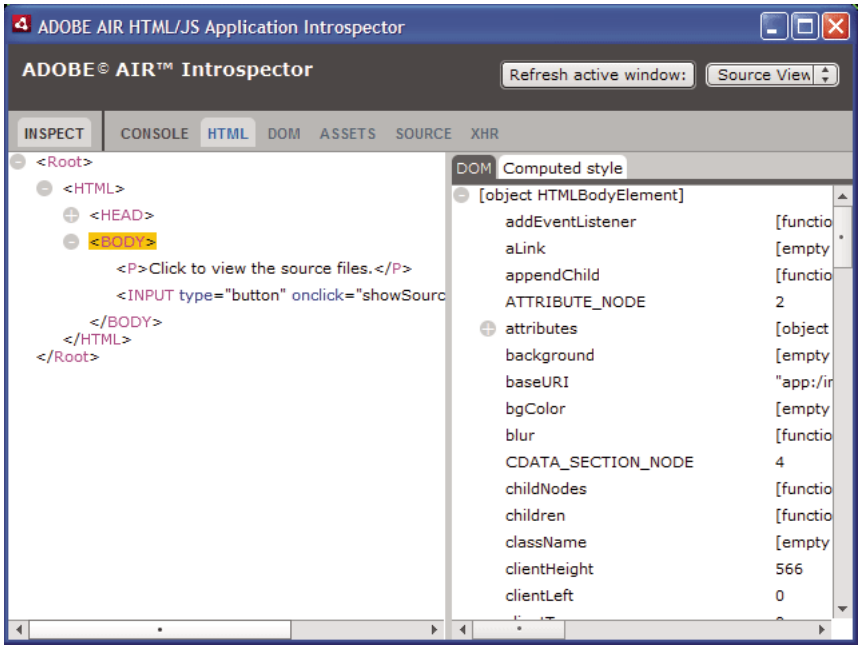
Для переменной `AIRIntrospectorConfig` можно определить восемь следующих свойств:

Свойство	**Значение по умолчанию**	**Описание**
`closeIntrospectorOnExit`	`true`	Задает, что окно Inspector будет закрываться, когда будут закрыты все другие окна приложения.
`debuggerKey`	123 (клавиша F12)	Код ключа для клавиши быстро вызова, позволяющей отображать и скрывать окно AIR Introspector.
`debugRuntimeObjects`	`true`	Задает, чтобы Introspector развертывал объекты выполнения в дополнение к объектам, определенным в JavaScript.
`flashTabLabels`	`true`	Задает, чтобы вкладки Console (Консоль) и XMLHttpRequest мигали, показывая, когда на них произошли изменения (например, когда изменяется текст на этих вкладках).
`introspectorKey`	122 (клавиша F11)	Код ключа для клавиши быстрого вызова, для открытия панели Inspect.
`showTimestamp`	`true`	Задает, чтобы в начале каждой строки на вкладке Console (Консоль) отображалась метка времени.
`showSender`	`true`	Задает, чтобы в начале каждой строки на вкладке Console (Консоль) отображалась информация об объекте, отправившем это сообщение.
`wrapColumns`	2000	Число столбцов, на которые разбиты исходные файлы.

Интерфейс AIR Introspector

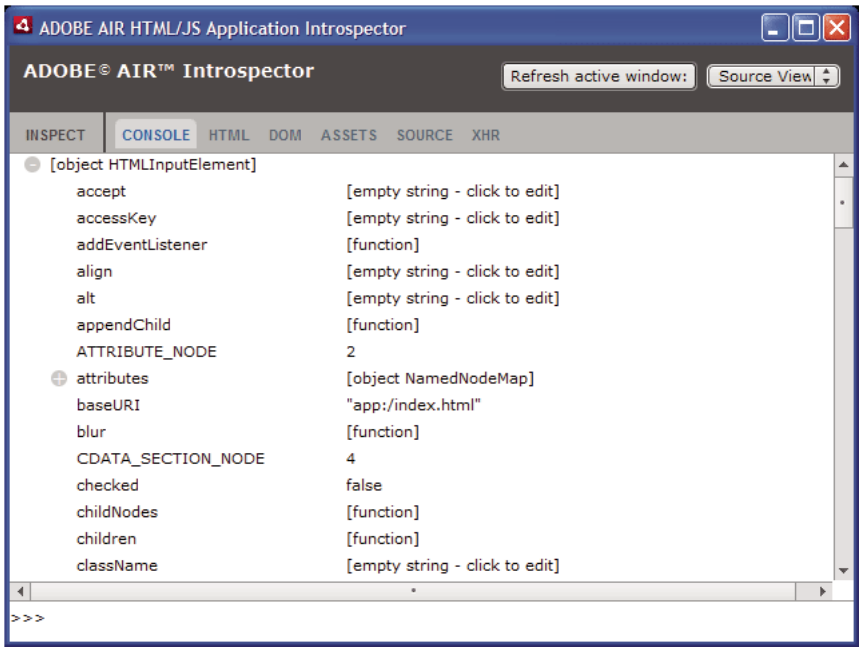
Чтобы открыть окно AIR Introspector при отладке приложения, нажмите клавишу F12 или вызовите один из методов класса Console (см. раздел ««Проверка объекта на вкладке Console (Консоль)» на странице 133»). Можно настроить другую клавишу быстрого вызова вместо клавиши F12; см. раздел ««Настройка AIR Introspector» на странице 135».

В окне AIR Introspector содержится шесть вкладок: Console (Консоль), HTML, DOM, Assets, Source и XHR, как показано на следующем рисунке:



Вкладка Console (Консоль)

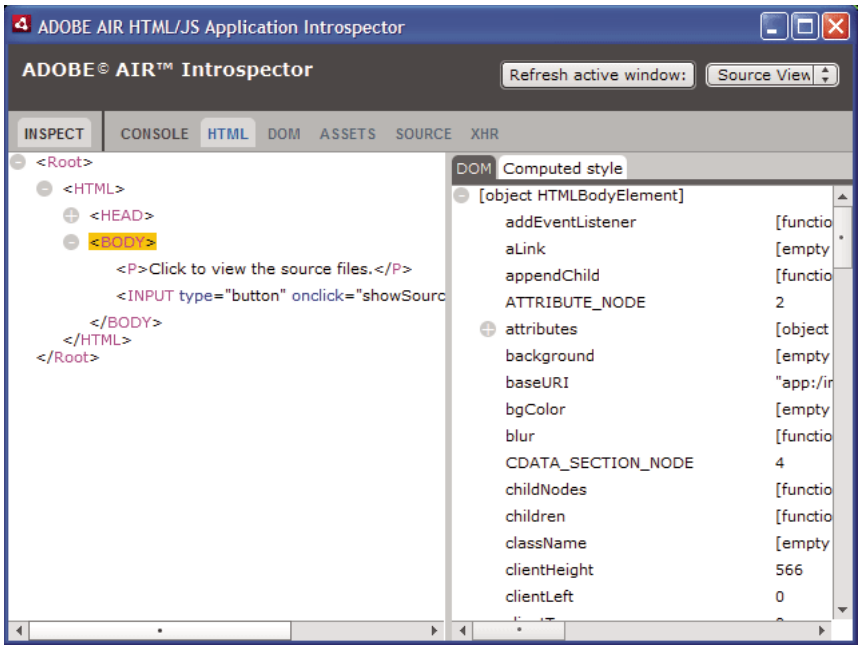
На вкладке Console (Консоль) отображаются значения свойств, переданные как параметры одному из методов класса air.Introspector.Console. Дополнительные сведения см. в разделе ««Проверка объекта на вкладке Console (Консоль)» на странице 133».



- Чтобы очистить консоль, щелкните текст правой кнопкой мыши и выберите Clear Console (Очистить консоль).
- Чтобы сохранить текст из вкладки Console (Консоль) в файл, щелкните вкладку Console (Консоль) правой кнопкой мыши и выберите Save Console To File (Сохранить консоль в файл).
- Чтобы сохранить текст из вкладки Console (Консоль) в буфер обмена, щелкните вкладку Console (Консоль) правой кнопкой мыши и выберите Save Console To Clipboard (Сохранить консоль в буфер обмена). Чтобы скопировать в буфер обмена только выделенный текст, щелкните текст правой кнопкой мыши и выберите команду «Копировать».
- Чтобы сохранить текст из класса Console в файл, щелкните вкладку Console (Консоль) правой кнопкой мыши и выберите Save Console To File (Сохранить консоль в файл).
- Чтобы выполнить поиск соответствующего критерию текста, отображаемого на вкладке, нажмите CTRL+F в Windows или Command+F в Mac OS. (Узлы дерева, которые не отображаются, не доступны для поиска.)

Вкладка HTML

Вкладка HTML позволяет просматривать весь HTML DOM в древовидной структуре. Щелкните элемент, чтобы просмотреть его свойства в правой части вкладки. Щелкните значки «+» и «-», чтобы развернуть или свернуть узел в дереве.



Можно отредактировать любой атрибут или текстовый элемент на вкладке HTML и измененное значение отобразится в приложении.

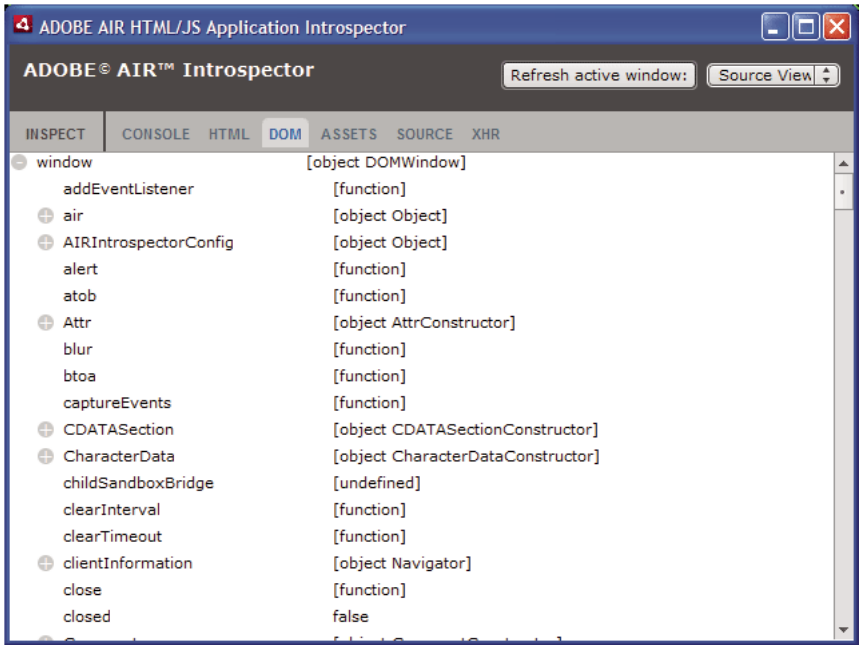
Нажмите кнопку Inspect (Проверить) (слева от списка вкладок в окне AIR Introspector). Можно щелкнуть любой элемент на странице HTML главного окна и связанный объект DOM отобразится на вкладке HTML. Если для главного окна включен фокус, можно также нажать клавишу быстрого вызова, чтобы включить или отключить кнопку Inspect. По умолчанию для этого используется клавиша быстрого вызова F11. Можно настроить другую клавишу быстрого вызова вместо клавиши F11. См. раздел ««Настройка AIR Introspector» на странице 135».

Нажмите кнопку Refresh Active Window (Обновить активное окно) в верхней части окна AIR Introspector, чтобы обновить данные, отображаемые на вкладке HTML.

Нажмите CTRL+F в Windows или Command+F в Mac OS, чтобы выполнить поиск соответствующего критерию текста, отображаемого на вкладке. (Узлы дерева, которые не отображаются, не доступны для поиска.)

Вкладка DOM

На вкладке DOM показывается объект окна в древовидной структуре. Можно изменить любые строчные или численные свойства, отредактированные значения отображаются в приложении.

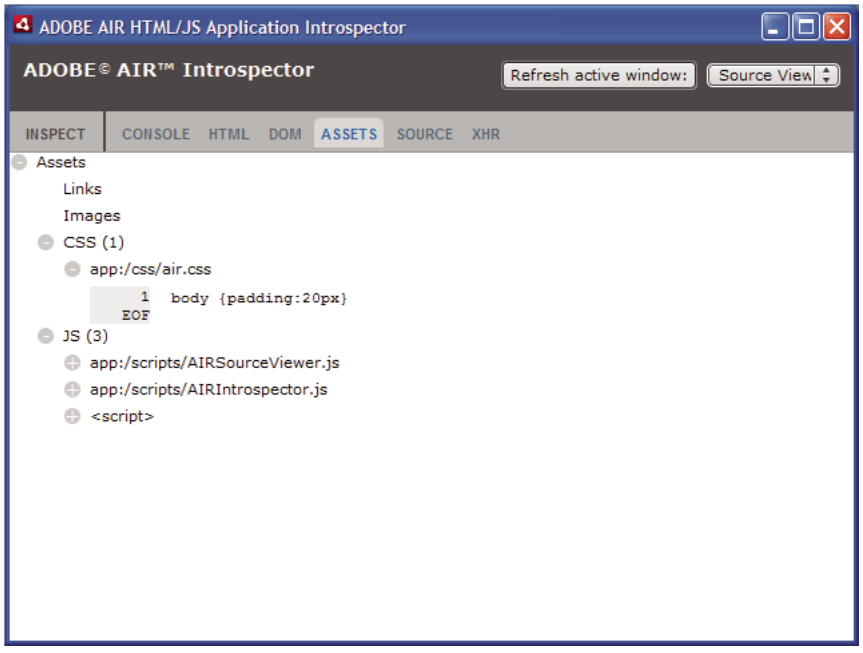


Нажмите кнопку Refresh Active Window (Обновить активное окно) в верхней части окна AIR Introspector, чтобы обновить данные, отображаемые на вкладке DOM.

Нажмите CTRL+F в Windows или Command+F в Mac OS, чтобы выполнить поиск соответствующего критерию текста, отображаемого на вкладке. (Узлы дерева, которые не отображаются, не доступны для поиска.)

Вкладка Assets (Активы)

На вкладке Assets (Активы) можно проверить ссылки, изображения, CSS и файлы JavaScript, загруженные в исходном окне. При развертывании одного из этих узлов показывается содержимое файла или отображается фактически используемое изображение.



Нажмите кнопку Refresh Active Window (Обновить активное окно) в верхней части окна AIR Introspector, чтобы обновить данные, отображаемые на вкладке Assets (Активы).

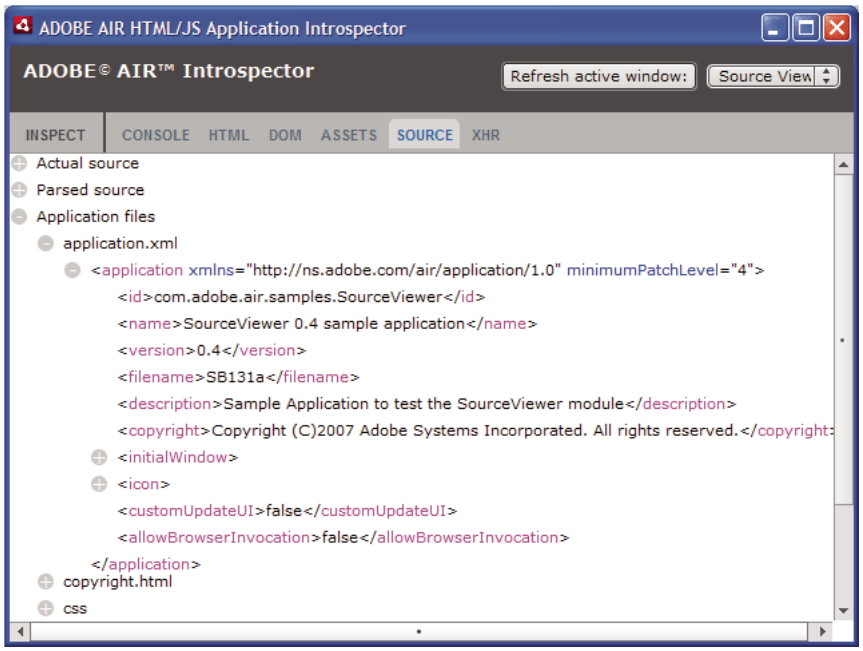
Нажмите CTRL+F в Windows или Command+F в Mac OS, чтобы выполнить поиск соответствующего критерию текста, отображаемого на вкладке. (Узлы дерева, которые не отображаются, не доступны для поиска.)

Вкладка Source (Источник)

Вкладка Source (Источник) содержит три раздела:

- Фактический источник: показывается исходный HTML-код страницы, загруженной в качестве основного содержимого при запуске приложения.
- Разобранный источник: показывается текущая разметка, образующая интерфейс пользователя приложения, которая может отличаться от фактического источника, поскольку в ходе работы приложение создает код разметки, используя технологии Ajax.

- Файлы приложения: отображается список файлов в каталоге приложения. Этот список доступен только для AIR Introspector, если он запущен из содержимого в изолированной программной среде приложения. В этом разделе можно просматривать содержимое текстовых файлов или изображения.

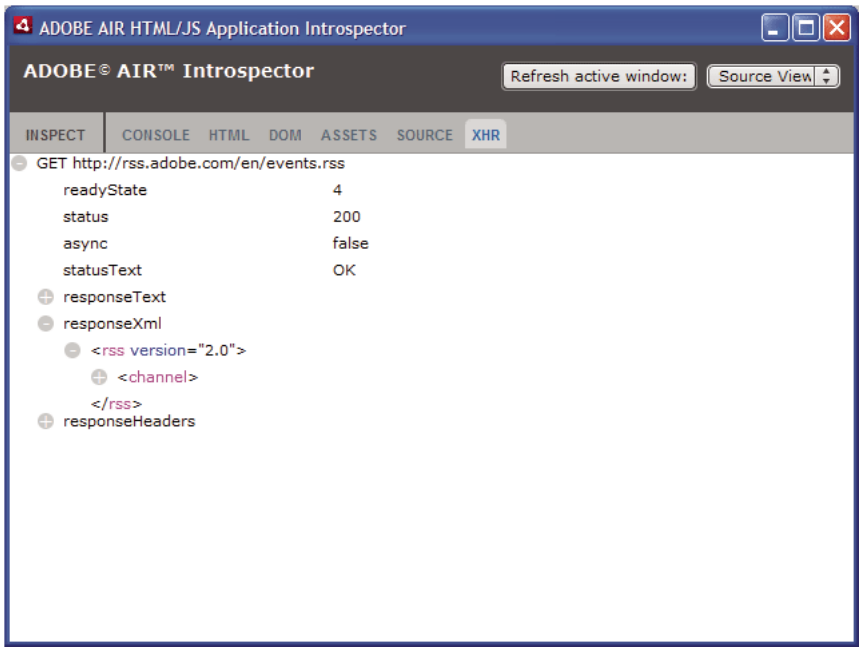


Нажмите кнопку Refresh Active Window (Обновить активное окно) в верхней части окна AIR Introspector, чтобы обновить данные, отображаемые на вкладке Source (Источник).

Нажмите CTRL+F в Windows или Command+F в Mac OS, чтобы выполнить поиск соответствующего критерию текста, отображаемого на вкладке. (Узлы дерева, которые не отображаются, не доступны для поиска.)

Вкладка XHR

Вкладка XHR перехватывает все обращения к XMLHttpRequest в приложении и протоколирует эту информацию. Это позволяет просматривать в древовидном представлении свойства XMLHttpRequest, включая `responseText` и `responseXML` (если они доступны).



Нажмите CTRL+F в Windows или Command+F в Mac OS, чтобы выполнить поиск соответствующего критерию текста, отображаемого на вкладке. (Узлы дерева, которые не отображаются, не доступны для поиска.)

Использование AIR Introspector с содержимым, находящимся вне изолированной программной среды приложения

Можно загрузить содержимое из каталога приложения в объект iframe или frame, который отображается на изолированную программную среду, не связанную с приложением. (См. раздел Система защиты HTML в Adobe AIR для разработчиков ActionScript или Система защиты HTML в Adobe AIR для разработчиков ActionScript). Можно использовать AIR Introspector с таким содержимым, но соблюдая следующие правила:

- Файл AIRIntrospector.js должен быть включен в содержимое, находящееся как в изолированной программной среде приложения, так и вне ее(iframe).
- Не переопределяйте свойство `parentSandboxBridge`; оно используется кодом AIR Introspector. Добавьте необходимые свойства. Вместо того, чтобы писать следующий код:

  ```
  parentSandboxBridge = mytrace: function(str) {runtime.trace(str)}} ;
  ```

  Используйте следующий синтаксис:

  ```
  parentSandboxBridge.mytrace = function(str) {runtime.trace(str)};
  ```

- Из содержимого, находящегося вне изолированной программной среды приложения, нельзя открыть AIR Introspector нажатием клавиши F12 или вызовом одного из методов класса air.Introspector.Console. Можно открыть окно Introspector только нажатием кнопки Open Introspector (Открыть Introspector). Эта кнопка по умолчанию добавляется в правый верхний угол объекта iframe или frame. (Из-за ограничений безопасности, связанных с использованием содержимого вне изолированной программной среды приложения, новое окно может быть открыто только в результате действия пользователя, например нажатия кнопки.)
- Можно открыть отдельные окна AIR Introspector для содержимого, находящегося в изолированной программной среде приложения и вне его. Различать эти окна AIR Introspector можно по их заголовкам.
- На вкладке Source (Источник) файлы приложения не отображаются, если AIR Introspector запущен из содержимого, находящегося вне изолированной программной среды приложения
- AIR Introspector может просматривать только тот код в изолированной программной среде приложения, из которого он был запущен.

Глава 18. Локализация приложений AIR

Adobe AIR 1.1 и более новых версий

Adobe® AIR® предлагает поддержку нескольких языков.

Обзор локализованного содержимого ActionScript 3.0 и инфраструктуры Flex см. в разделе «Локализация приложений» в руководстве разработчика ActionScript 3.0.

Языки, поддерживаемые AIR

Поддержка локализации приложений AIR для следующих языков впервые появилась в версии AIR 1.1:

- Китайский (упрощенный)
- Китайский (традиционный)
- Французский
- Немецкий
- Итальянский
- Японский
- Корейский
- Португальский (Бразилия)
- Русский
- Испанский

В версии AIR 1.5 были добавлены следующие языки:

- Чешский
- Голландский
- Польский
- Шведский
- Турецкий

Дополнительные разделы справки

Создание многоязычных приложений Flex в Adobe AIR

Создание многоязычных HTML-приложений

Локализация названия и описания приложения в программе установки приложения AIR

Adobe AIR 1.1 и более новых версий

Для элементов `name` и `description` в файле дескриптора приложения можно задать несколько языков. Например, ниже показано как задать название на трех языках (английском, французском и немецком):

```
<name>
    <text xml:lang="en">Sample 1.0</text>
    <text xml:lang="fr">Échantillon 1.0</text>
    <text xml:lang="de">Stichprobe 1.0</text>
</name>
```

Атрибут `xml:lang` для каждого текстового элемента·задает код языка согласно документу RFC4646 (http://www.ietf.org/rfc/rfc4646.txt).

Элемент name задает название приложения, которое отображается в программе установки AIR. Программа установки приложения AIR использует локализованное значение, лучше всего подходящее к языку пользовательского интерфейса, определенному параметрами операционной системы.

Таким же образом можно задать несколько языков для элемента `description` в файле дескриптора приложения. Этот элемент задает текст описания, который выводится в программе установки приложения.

Эти параметры относятся только к языкам, доступным в программе установки AIR. Они не определяют локали, доступные для установленного и работающего приложения. Приложения AIR могут иметь интерфейсы на многих языках, помимо тех, которые доступны в программе установки AIR.

Дополнительные сведения см. в разделе ««Определение свойств файла дескриптора приложения» на странице 72».

Дополнительные разделы справки

Создание многоязычных приложений Flex в Adobe AIR

Создание многоязычных HTML-приложений

Локализация HTML-содержимого с помощью инфраструктуры локализации AIR HTML

Adobe AIR 1.1 и более новых версий

В комплект AIR 1.1 SDK включена инфраструктура локализации HTML. Она определяется файлом AIRLocalizer.js JavaScript. Файл AIRLocalizer.js находится в каталоге frameworks комплекта AIR SDK. В нем задан класс air.Localizer, обеспечивающий поддержку многоязычных приложений.

Загрузка кода HTML-инфраструктуры локализации AIR

Чтобы воспользоваться многоязычной функциональностью, скопируйте файл AIRLocalizer.js в своей проект. Затем включите его в основной HTML-файл приложения, заключив в тег script:

```
<script src="AIRLocalizer.js" type="text/javascript" charset="utf-8"></script>
```

Код JavaScript может вызвать объект `air.Localizer.localizer`:

```
<script>
    var localizer = air.Localizer.localizer;
</script>
```

`air.Localizer.localizer` — это единый объект, определяющий методы и свойства использования локализованных ресурсов. Класс Localizer содержит ряд методов DRM:

Метод	Описание
`getFile()`	Получает текст определенного пакета ресурсов для заданной локали. См. раздел ««Получение ресурсов для определенной локали» на странице 152».
`getLocaleChain()`	Возвращает языки в цепочке локалей. См. раздел ««Определение цепочки локалей» на странице 151».
`getResourceBundle()`	Возвращает ключи пакета и соответствующие значения в виде объекта. См. раздел ««Получение ресурсов для определенной локали» на странице 152».
`getString()`	Получает строку, определенную для данного ресурса. См. раздел ««Получение ресурсов для определенной локали» на странице 152».
`setBundlesDirectory()`	Задает расположение каталога пакетов. См. раздел ««Изменение параметров Localizer в AIR HTML» на странице 150».
`setLocalAttributePrefix()`	Задает префикс, используемый атрибутами localizer в HTML-элементах DOM. См. раздел ««Изменение параметров Localizer в AIR HTML» на странице 150»
`setLocaleChain()`	Задает порядок языков в цепочке локалей. См. раздел ««Определение цепочки локалей» на странице 151».
`sortLanguagesByPreference()`	Располагает локали в цепочке согласно порядку локалей в параметрах операционной системы. См. раздел ««Определение цепочки локалей» на странице 151».
`update()`	Обновляет модель DOM HTML (или элемент DOM), используя локализованные строки из текущей цепочки локалей. Описание цепочек локалей см. в разделе ««Управление цепочками локалей» на странице 147». Дополнительные сведения о методе `update()` см. в разделе ««Обновление элементов DOM для использования текущей локали» на странице 149».

Класс Localizer содержит ряд статичных свойств.

Свойство	Описание
`localizer`	Возвращает ссылку на единственный в приложении объект Localizer.
`ultimateFallbackLocale`	Локаль, используемая, когда приложение не поддерживает выбор пользователя. См. раздел ««Определение цепочки локалей» на странице 151».

Определение пакетов ресурсов

Инфраструктура локализации HTML считывает локализованные строки из файлов *локализации*. Файл локализации — это набор значений, привязанных к ключам и последовательно собранных в текстовый файл. Файл локализации иногда называют *пакетом*.

Создайте в каталоге приложения подкаталог с названием locale. (Можно использовать другое имя, см. раздел ««Изменение параметров Localizer в AIR HTML» на странице 150».) Здесь будут храниться файлы локализации. Этот каталог известен как *каталог пакетов*.

Для каждой локали, которая поддерживается приложением, создайте подкаталог каталога пакетов. Назовите каждый подкаталог в соответствии с кодом локали. Например, для французского каталога используйте имя «fr», а для английского — «en». С помощью знака подчеркивания (_) можно определить локаль, содержащую язык и код страны. Например, каталог для английского (США) может называться «en_us». (Также можно использовать дефис, в этом случае каталог будет называться «en-us». Инфраструктура локализации распознает оба варианта.)

В подкаталог locale можно добавить сколь угодно много файлов ресурсов. Как правило, файл локализации создается для каждого языка (и помещается в каталог соответствующего языка). Инфраструктура локализации HTML содержит метод `getFile()`, позволяющий считывать содержимое файла (см. раздел ««Получение ресурсов для определенной локали» на странице 152»).

Файлы с расширением .properties называются файлами свойств локализации. Они используются для определения пар «ключ-значение» той или иной локали. Файл свойств задает строковое значение на каждой строке. Например, ниже показано, как задать строковое значение `"Hello in English."` для ключа с именем `greeting`:

```
greeting=Hello in English.
```

В файле свойств содержится следующий текст и задаются шесть пар «ключ-значение»:

```
title=Sample Application
greeting=Hello in English.
exitMessage=Thank you for using the application.
color1=Red
color2=Green
color3=Blue
```

Этом примере показана английская версия файла свойств из каталога en.

Французская версия этого файла будет помещена в каталог fr:

```
title=Application Example
greeting=Bonjour en français.
exitMessage=Merci d'avoir utilisé cette application.
color1=Rouge
color2=Vert
color3=Bleu
```

Для разных типов информации можно задать разные файлы ресурсов. Скажем, в файле legal.properties может храниться какой-нибудь шаблонный юридический документ (к примеру, информация об авторских правах). Эти ресурсы можно повторно использовать в различных приложения Точно так же можно задать файлы для определения локализованного содержимого разных частей пользовательского интерфейса.

В этих файлах следует использовать кодировку UTF-8, чтобы языки отображались корректно.

Управление цепочками локалей

Когда приложение загружает файл AIRLocalizer.js, оно исследует доступные локали. Эти локали соответствуют подкаталогам каталога пакетов (см. раздел ««Определение пакетов ресурсов» на странице 146»). Список поддерживаемых локалей называется *цепочкой локалей*. Файл AIRLocalizer.js автоматически определяет порядок локалей в цепочке на основании соответствующих параметров операционной системы. (Свойство `Capabilities.languages` содержит список языков пользовательского интерфейса операционной системы, в порядке предпочтения.)

Поэтому, если в приложении определены ресурсы для локалей «en», «en_US» и «en_UK», инфраструктура Localizer AIR HTML упорядочивает цепочку локалей соответственно. Если приложение запускается в системе, где основной локалью является «en», цепочка локалей будет иметь порядок `["en", "en_US", "en_UK"]`. В этом случае приложение сначала будет искать ресурсы в пакете «en», потом в «en_US».

Однако, если основной локалью в системе является «en-US», то порядок будет таков: `["en_US", "en", en_UK"]`. В этом случае приложение сперва обратится к пакету «en_US», а потом к «en».

По умолчанию, первая локаль в цепочке считается стандартом по умолчанию. Можно попросить пользователя выбрать локаль при первом запуске приложения. Затем можно сохранить этот выбор в файле настроек и при последующих запусках сразу включать эту локаль.

Могут использоваться строки ресурсов любой локали из цепочки. Если в какой-то из локалей строка ресурсов не задана, приложение обратится к следующей подходящей строке из другой локали.

Для изменения настроек цепочки локалей вызовите метод `setLocaleChain()` объекта Localizer. См. раздел ««Определение цепочки локалей» на странице 151».

Обновление элементов DOM с использованием локализованного содержимого

Элемент приложения может обращаться к ключу в файле свойств локализации. Например, элемент `title` в примере ниже задает атрибут `local_innerHTML`. Инфраструктура локализации использует этот атрибут для нахождения локализованного значения. По умолчанию, инфраструктура ищет атрибуты, имена которых начинаются с `"local_"`. Она обновляет атрибуты с совпадающей частью имени после `"local_"`. В этом случае инфраструктура задает атрибут `innerHTML` элемента `title`. Атрибут `innerHTML` использует значение, заданное для ключа `mainWindowTitle` в файле свойств по умолчанию (default.properties):

```
<title local_innerHTML="default.mainWindowTitle"/>
```

Если в текущей локали нет подходящего значения, тогда инфраструктура локализации просматривает остальные локали цепочки. Используется ближайшая локаль в цепочке, в которой значение определено.

В примере ниже в тексте (атрибут `innerHTML` attribute) элемента `p` используется значение ключа `greeting`, заданное в файле свойств по умолчанию:

```
<p local_innerHTML="default.greeting" />
```

В примере ниже в атрибуте value (и отображаемом тексте) элемента `input` используется значение ключа `btnBlue` из файла свойств по умолчанию:

```
<input type="button" local_value="default.btnBlue" />
```

Чтобы обновить модель DOM HTML и использовать строки, заданные в текущей цепочке ключей, вызовите метод `update()` объекта Localizer. Вызов метода `update()` позволяет объекту Localizer разобрать DOM и выполнить необходимые действия над найденными атрибутами локализации (`"local_..."`):

```
air.Localizer.localizer.update();
```

Можно задать значения для атрибута (например, «innerHTML») и соответствующего ему атрибута локализации (например, «local_innerHTML»). В этом случае инфраструктура локализации перепишет значение атрибута, только если найдет совпадающее значение в цепочке. Например, следующий элемент задает атрибуты `value` и `local_value`:

```
<input type="text" value="Blue" local_value="default.btnBlue"/>
```

Можно обновить и отдельный элемент DOM. См. следующий раздел ««Обновление элементов DOM для использования текущей локали» на странице 149».

По умолчанию Localizer AIR HTML использует `"local_"` в качестве префикса для атрибутов, определяющий параметры локализации элемента. Например, по умолчанию атрибут `local_innerHTML` определяет имя пакета и ресурса для значения `innerHTML` элемента. Кроме того, по умолчанию атрибут `local_value` определяет имя пакета и ресурса для значения `value` элемента. Localizer можно изменить, чтобы использовать другой префикс атрибута (не `"local_"`). См. раздел ««Изменение параметров Localizer в AIR HTML» на странице 150».

Обновление элементов DOM для использования текущей локали

Когда объект Localizer обновляет модель DOM HTML, помеченные элементы вынуждены использовать значения атрибутов, основанные на строках, которые заданы в текущей цепочке локалей. Чтобы Localizer HTML обновил DOM HTML, вызовите метод `update()` объекта `Localizer`:

```
air.Localizer.localizer.update();
```

Чтобы обновить только отдельный элемент DOM, передайте его в качестве параметра методу `update()`. Метод `update()` имеет только один параметр, `parentNode`, и тот не обязателен. Если параметр `parentNode` задан, он определяет локализуемый элемент DOM. При вызове метода `update()` и указании параметра `parentNode` задаются локализованные значения для всех дочерних элементов, которые указывают атрибуты локализации.

Рассмотрим элемент `div`:

```
<div id="colorsDiv">
    <h1 local_innerHTML="default.lblColors" ></h1>
    <p><input type="button" local_value="default.btnBlue" /></p>
    <p><input type="button" local_value="default.btnRed" /></p>
    <p><input type="button" local_value="default.btnGreen" /></p>
</div>
```

Чтобы обновить этот элемент и использовать локализованные строки, заданные в цепочке локалей, воспользуйтесь следующим кодом JavaScript:

```
var divElement = window.document.getElementById("colorsDiv");
air.Localizer.localizer.update(divElement);
```

Если значение ключа не найдено в цепочке локалей, инфраструктура использует значение атрибута `"local_"`. Положим, в предыдущем примере инфраструктуре не удалось найти значение для ключа `lblColors` (ни в одном из файлов default.properties в цепочке локалей). Тогда в качестве значения `innerHTML` будет использоваться `"default.lblColors"`. Использование этого значения сообщает разработчику, что каких-то ресурсов недостает.

Метод `update()` отправляет событие `resourceNotFound`, когда не может найти ресурс в цепочке локалей. Константа `air.Localizer.RESOURCE_NOT_FOUND` задает строку `"resourceNotFound"`. У этого события есть три свойства: `bundleName`, `resourceName` и `locale`. Свойство `bundleName` — это имя пакета, в котором не удалось обнаружить ресурс. Свойство `resourceName` — это имя ресурса, который не удалось обнаружить. Свойство `locale` — это имя локали, в которой не удалось обнаружить ресурс.

Метод `update()` отправляет событие `bundleNotFound`, когда не может найти указанный пакет. Константа `air.Localizer.BUNDLE_NOT_FOUND` задает строку `"bundleNotFound"`. У события есть два свойства: `bundleName` и `locale`. Свойство `bundleName` — это имя пакета, в котором не удалось обнаружить ресурс. Свойство `locale` — это имя локали, в которой не удалось обнаружить ресурс.

Метод `update()` работает асинхронно (и асинхронно отправляет события `resourceNotFound` и `bundleNotFound`). Следующий код задает прослушиватели для событий `resourceNotFound` и `bundleNotFound`:

```
air.Localizer.localizer.addEventListener(air.Localizer.RESOURCE_NOT_FOUND, rnfHandler);
air.Localizer.localizer.addEventListener(air.Localizer.BUNDLE_NOT_FOUND, rnfHandler);
air.Localizer.localizer.update();
function rnfHandler(event)
{
    alert(event.bundleName + ": " + event.resourceName + ":." + event.locale);
}
function bnfHandler(event)
{
    alert(event.bundleName + ":." + event.locale);
}
```

Изменение параметров Localizer в AIR HTML

Метод `setBundlesDirectory()` объекта Localizer позволят изменить путь к каталогу пакетов. Метод `setLocalAttributePrefix()` объекта Localizer позволяет изменить путь к каталогу пакетов и значения атрибутов, используемых объектом Localizer.

Каталог пакетов по умолчанию определен как подкаталог locale в каталоге приложения. Можно задать другой каталог, вызвав метод `setBundlesDirectory()` объекта Localizer. Этот метод принимает один параметр, `path`, задающий в виде строки путь к нужному каталогу пакетов. Значением параметра `path` может быть одно из следующих:

- объект String, определяющий относительный путь к каталогу, например `"locales"`
- объект String, задающий действительный URL-адрес, использующий схемы URL-адресов `app`, `app-storage` или `file`, например `"app://languages"` (*не* используйте схему `http`)
- объект File

Сведения об URL-адресах и путях к каталогам см. в следующих источниках:

- Пути объектов File (для разработчиков ActionScript)
- Пути объектов File (для разработчиков HTML)

Например, в коде ниже в качестве каталога пакетов задается подкаталог languages каталога хранилища приложения (не каталога приложения):

```
air.Localizer.localizer.setBundlesDirectory("languages");
```

В качестве параметра `path` должен передаваться действительный путь. В противном случае метод генерирует исключение BundlePathNotFoundError. `"BundlePathNotFoundError"` является значением свойства `name` этой ошибки, а свойство `message` указывает недействительный путь.

По умолчанию Localizer AIR HTML использует `"local_"` в качестве префикса для атрибутов, определяющий параметры локализации элемента. Например, атрибут `local_innerHTML` определяет имена пакета и ресурса в значении `innerHTML` следующего элемента `input`:

```
<p local_innerHTML="default.greeting" />
```

Метод `setLocalAttributePrefix()` объекта Localizer позволяет использовать префикс атрибута, отличный от `"local_"`. Статический метод принимает один параметр — строку, которая используется в качестве префикса атрибута. Например, в коде ниже инфраструктура локализации использует в качестве префикса атрибута «loc_»:

```
air.Localizer.localizer.setLocalAttributePrefix("loc_");
```

Этот префикс можно изменять. Например, это может потребоваться, если префикс по умолчанию (`"local_"`) приводит к конфликту с именем другого атрибута в коде. При вызове этого метода используйте допустимые символы для HTML-атрибутов. (Например, знак пробела использовать нельзя.)

Дополнительные сведения об использовании атрибутов локализации в HTML-элементах см. в разделе ««Обновление элементов DOM с использованием локализованного содержимого» на странице 148».

Параметры каталога пакетов и префикса атрибута не сохраняются между разными сеансами приложения. Если вы используете собственный каталог пакетов или префикс атрибута, его потребуется задавать каждый раз при запуске приложения.

Определение цепочки локалей

По умолчанию, когда загружается файл AIRLocalizer.js, он задает цепочку локалей. Эту цепочку определяют локали, доступные в каталоге пакетов и параметры языка пользовательского интерфейса системы. (Дополнительные сведения см. в разделе ««Управление цепочками локалей» на странице 147».)

Для изменения настроек цепочки локалей вызовите статический метод `setLocaleChain()` объекта Localizer. Например, этот метод можно вызвать, если пользователь указал, какой язык предпочитает. Метод `setLocaleChain()` принимает один параметр, `chain`, представляющий собой массив локалей, например `["fr_FR","fr","fr_CA"]`. Инфраструктура ищет ресурсы (в последовательности операций) в том порядке, в каком идут локали в массиве. Если ресурс не удается найти в первой локали, то поиск продолжается в ресурсах других локалей. Если аргумент `chain` отсутствует, не является массивом или является пустым массивом, то функция дает сбой и генерирует исключение IllegalArgumentsError.

Статический метод `getLocaleChain()` объекта Localizer возвращает объект Array со списком локалей в текущей цепочке.

Приведенный ниже код читает текущую цепочку локалей и добавляет в начало две французских локали:

```
var currentChain = air.Localizer.localizer.getLocaleChain();
newLocales = ["fr_FR", "fr"];
air.Localizer.localizer.setLocaleChain(newLocales.concat(currentChain));
```

При обновлении цепочки локалей метод `setLocaleChain()` отправляет событие `"change"`. Константа `air.Localizer.LOCALE_CHANGE` определяет строку `"change"`. Событие имеет единственное свойство, `localeChain`, представляющее собой массив кодов локалей в новой цепочке. В коде ниже задается прослушиватель для этого события:

```
var currentChain = air.Localizer.localizer.getLocaleChain();
newLocales = ["fr_FR", "fr"];
localizer.addEventListener(air.Localizer.LOCALE_CHANGE, changeHandler);
air.Localizer.localizer.setLocaleChain(newLocales.concat(currentChain));
function changeHandler(event)
{
    alert(event.localeChain);
}
```

Статическое свойство `air.Localizer.ultimateFallbackLocale` представляет локаль, которая используется, если приложение не поддерживает пользовательские установки. Значение по умолчанию равно `"en"`. Можно задать и другую локаль, как показано ниже:

```
air.Localizer.ultimateFallbackLocale = "fr";
```

Получение ресурсов для определенной локали

Метод `getString()` объекта Localizer возвращает строку, определенную для ресурса в той или иной локали. При вызове этого метода значение `locale` задавать необязательно. В этом случае метод просматривает всю цепочку локалей и возвращает строку в первой же локали, которая содержит искомое имя ресурса. Ниже перечислены параметры этого метода:

Параметр	**Описание**
`bundleName`	Пакет, содержащий ресурс. Это имя файла свойств без расширения .properties. (Например, если этот параметр равен `"alerts"`, код Localizer просматривает файлы локализации alerts.properties.)
`resourceName`	Имя ресурса.
`templateArgs`	Необязательно. Массив строк для замены пронумерованных тегов в строке замещения. Например, рассмотрим вызов функции, где параметр `templateArgs` равен `["Raúl", "4"]`, а совпадающая строка ресурса — `"Hello, {0}. You have {1} new messages."` ("Здравствуйте, {0}. У вас {1} новых сообщения."). В этом случае функция возвратит запись `"Hello, Raúl. You have 4 new messages."` ("Здравствуйте, Рауль. У вас 4 новых сообщения"). Чтобы игнорировать это значение, передайте `null`.
`locale`	Необязательно. Код локали (например, `"en"`, `"en_us"` или `"fr"`). Если локаль задана, а подходящих значений не обнаружено, метод не будет искать значения в остальных локалях цепочки. Если код локали не задан, функция возвращает строку в первой локали из цепочки, в которой обнаруживается нужное значение имени ресурса.

Инфраструктура локализации может обновлять отмеченные атрибуты DOM HTML. Однако локализованные строки могут использоваться иными способами. Например, можно использовать строку в каком-нибудь динамически создаваемом HTML-файле или в качестве значения параметра при вызове функции. В приведенном ниже коде в вызове функции `alert()` участвует строка, определенная в ресурсе `error114` в файле свойств по умолчанию для локали fr_FR:

```
alert(air.Localizer.localizer.getString("default", "error114", null, "fr_FR"));
```

Когда метод `getString()` не может найти ресурс в определенном пакете, он отправляет событие `resourceNotFound`. Константа `air.Localizer.RESOURCE_NOT_FOUND` задает строку `"resourceNotFound"`. У этого события есть три свойства: `bundleName`, `resourceName` и `locale`. Свойство `bundleName` — это имя пакета, в котором не удалось обнаружить ресурс. Свойство `resourceName` — это имя ресурса, который не удалось обнаружить. Свойство `locale` — это имя локали, в которой не удалось обнаружить ресурс.

Метод `getString()` отправляет событие `bundleNotFound`, когда не может найти указанный пакет. Константа `air.Localizer.BUNDLE_NOT_FOUND` задает строку `"bundleNotFound"`. У события есть два свойства: `bundleName` и `locale`. Свойство `bundleName` — это имя пакета, в котором не удалось обнаружить ресурс. Свойство `locale` — это имя локали, в которой не удалось обнаружить ресурс.

Метод `getString()` работает асинхронно (и асинхронно отправляет события `resourceNotFound` и `resourceNotFound`). Следующий код задает прослушиватели для событий `resourceNotFound` и `bundleNotFound`:

```
air.Localizerlocalizer.addEventListener(air.Localizer.RESOURCE_NOT_FOUND, rnfHandler);
air.Localizerlocalizer.addEventListener(air.Localizer.BUNDLE_NOT_FOUND, bnfHandler);
var str = air.Localizer.localizer.getString("default", "error114", null, "fr_FR");
function rnfHandler(event)
{
    alert(event.bundleName + ": " + event.resourceName + ":." + event.locale);
}
function bnfHandler(event)
{
    alert(event.bundleName + ":." + event.locale);
}
```

Метод `getResourceBundle()` объекта Localizer возвращает указанный пакет для заданного языка (параметр locale). Возвращаемое методом значение является объектом со свойствами, соответствующими ключам в пакете. (Если приложение не может найти указанный пакет, метод возвращает значение `null`.)

Метод использует два параметра: `locale` и `bundleName`.

Параметр	**Описание**
`locale`	Локаль (например, `"fr"`).
`bundleName`	Имя пакета.

Например, в следующем коде вызывается метод `document.write()` для загрузки пакета по умолчанию для локали (fr). Затем вызывается метод `document.write()` для записи значений ключей `str1` и `str2` в этом пакете:

```
var aboutWin = window.open();
var bundle = localizer.getResourceBundle("fr", "default");
aboutWin.document.write(bundle.str1);
aboutWin.document.write("<br/>");
aboutWin.document.write(bundle.str2);
aboutWin.document.write("<br/>");
```

Метод `getResourceBundle()` отправляет событие `bundleNotFound`, когда не может найти указанный пакет. Константа `air.Localizer.BUNDLE_NOT_FOUND` задает строку `"bundleNotFound"`. У события есть два свойства: `bundleName` и `locale`. Свойство `bundleName` — это имя пакета, в котором не удалось обнаружить ресурс. Свойство `locale` — это имя локали, в которой не удалось обнаружить ресурс.

Метод `getFile()` объекта Localizer возвращает содержимое пакета для данной локали в виде строки. Файл пакета считывается как файл UTF-8. У этого метода есть ряд параметров:

Параметр	Описание
`resourceFileName`	Имя файла ресурса (например, `"about.html"`).

`templateArgs`	Необязательно. Массив строк для замены пронумерованных тегов в строке замещения. Например, рассмотрим вызов функции, где параметр `templateArgs` равен `["Raúl", "4"]`, а совпадающая строка ресурса содержит две строки:
	`<html>`
	`<body>Hello, {0}. You have {1} new messages.</body>`
	`</html>`
	В этом случае функция возвратит строку с двумя строками:
	`<html>`
	`<body>Hello, Raúl. You have 4 new messages. </body>`
	`</html>`

`locale`	Используемый код локали, например `"en_GB"`. Если локаль задана, а подходящих файлов не обнаружено, метод не будет продолжать поиск в остальных локалях цепочки. Если код локали *не* задан, функция возвращает текст в первой же локали цепочки, в котором удается найти файл, соответствующий `resourceFileName`.

Например, в коде ниже метод `document.write()` вызывается с помощью содержимого файла about.html локали fr:

```
var aboutWin = window.open();
var aboutHtml = localizer.getFile("about.html", null, "fr");
aboutWin.document.close();
aboutWin.document.write(aboutHtml);
```

Метод `getFile()` отправляет событие `fileNotFound`, когда не может найти ресурс в цепочке локалей. Константа `air.Localizer.FILE_NOT_FOUND` определяет строку `"resourceNotFound"`. Метод `getFile()` работает асинхронно (и асинхронно отправляет событие `fileNotFound`). У события есть два свойства: `fileName` и `locale`. Свойство `fileName` — это имя файла, который не удалось найти. Свойство `locale` — это имя локали, в которой не удалось обнаружить ресурс. В коде ниже задается прослушиватель для этого события:

```
air.Localizer.localizer.addEventListener(air.Localizer.FILE_NOT_FOUND, fnfHandler);
air.Localizer.localizer.getFile("missing.html", null, "fr");
function fnfHandler(event)
{
    alert(event.fileName + ": " + event.locale);
}
```

Дополнительные разделы справки

Создание многоязычных HTML-приложений