Dolores Prerelease Release Notes 07-16

Flash Player 11.4 and AIR 3.4 Release Notes

July 16, 2012. Welcome to Adobe® Flash® Player 11.4 and Adobe AIR 3.4!

This pre-release includes new features as well as enhancements and bug fixes related to security, stability, performance, and device compatibility for Flash Player 11.4 and AIR 3.4 codenamed 'Dolores'. This document may be updated periodically as more information becomes available.
NOTE:

- The ActiveX Flash Player in this release is not compatible with Windows® 8.Flash Player for Windows® 8 is available as part of the Windows® 8 release preview.

Release features

- **ActionScript Workers (Flash Player)**
- **Stage3D "constrained" profile for increased GPU reach (Flash Player and AIR)**
- **Sandbox Bridge support (Flash Player)**
- **LZMA support for ByteArray (Flash Player and AIR)**
- **StageVideo attachCamera/Camera improvements (Flash Player and AIR)**
- **Compressed texture with alpha support for Stage3D (Flash Player and AIR)**
- **Deprecated Carbon APIs for AIR (AIR)**
- **Direct AIR deployment using ADT(without iTunes) (AIR)**
- **Push Notifications for iOS (AIR)**
- **Ambient AudioPlaybackMode (AIR)**
- **iOS 5.1 SDK Support (AIR)**
- **Exception Support in Native Extensions for iOS (AIR)**
- **New option in ADT to list the attached mobile devices (AIR)**
- **Licensing support: Flash Player Premium Features for Gaming (Flash Player only)**

ActionScript Workers

- With the introduction of Workers to ActionScript and the Flash Runtime, Flash Developers can now offload certain tasks like high-latency operations and long-running computations to "Background Workers". These Background Workers run concurrently in order to leverage more machine resources and avoid things like UI freezes. **Note**: To fully leverage ActionScript Workers and be able to debug workers, the next version of Flash Builder is required. This next version of Flash Builder will be made available through public beta in the August timeframe.

Stage3D "constrained" profile for increased GPU reach

- The Flash Player will gate the use of hardware acceleration based on the date of your video card's driver. In previous releases, we gated support to drivers older than January 1, 2008. In this release, we will be changing the gating to apply to drivers older than January 1, 2006. Content using wmode=direct (or renderMode=direct for AIR) should be hardware accelerated on graphics card driver date newer than 1/1/2006 when possible. This applies to Stage3D and [StageVideo|StageVideo] APIs. In addition to that, we added a new profile for Stage3D called "constrained" profile, allowing your content to run hardware accelerated on the previously blacklisted Intel GMA chipsets. See below for more details about this new profile and how to leverage it.

Sandbox Bridge support

- Sandbox bridging allows specific ActionScript objects or functions to be exposed between SWF-to-SWF cross-domain communication. This feature is already available in AIR and is being ported to Flash Player in this release.

LZMA support for ByteArray

- In addition to zlib compression of ByteArray, we are introducing an additional compression type based on LZMA to compress data inside a ByteArray through ByteArray.compress() and ByteArray.uncompress().

Compressed texture with alpha support for Stage3D (Flash Player and AIR)

- Transparent images are now supported for compressed textures (ATF file format). Note: Stay tuned for the ATF tools which will be made available in the next weeks on Adobe Labs.

StageVideo attachCamera/Camera improvements

- This feature exposes a new method on StageVideo objects that allows the Actionscript code to direct the video stream from the camera to a StageVideo object thus leveraging the GPU for rendering instead of the rasterizer. This allows the player to be much faster when processing a video stream from the camera when GPU acceleration is available.

Deprecated Carbon APIs for AIR

- As of Mac OS X 10.8, Carbon APIs will no longer be supported by Apple. Hence these APIs and other deprecated code is being removed from AIR in this release to be in parity with Mac OSX 10.8.

Direct AIR deployment using ADT (without iTunes)

- This new feature enables the developer to deploy an AIR application on iOS devices without having to use iTunes or XCode.

iOS Push Notifications

- This feature will use APNS (Apple Push Notification Service) and a provider (third party server which will communicate with the APN) to generate notifications. A new package, flash.notifications has been introduced. The delivery of push notifications is completely dependent on Apple Push Notification Service and APNS does not guarantee the delivery of push notifications. Apple also recommends subscribing to push notifications every time an application is launched. Every time the client app subscribes to push notifications, the APNS provides a token-id to the client app and this token-id should be sent to the third party server or provider that will be sending the remote notifications.

Ambient AudioPlaybackMode

- With AIR 3.4, users will have the option of one more AudioPlaybackMode i.e. AMBIENT other than VOICE and MEDIA. This new AudioPlaybackMode will allow the users to force their application to honor the "hardware mute switch" present in iphone(s) /ipad(s). The strongest use case of this Ambient playbackmode is in the game apps where the user of the app will have the option to silence the game sound and also can listen to the music of any other app in background. In Ambient mode, audio respects the hardware mute switch only on iOS devices. On Android devices, the Ambient mode will be just like the Media mode.

iOS 5.1 SDK Support

- With this AIR SDK version, you will be able to build your AIR on iOS applications by default with iOS 5.1 SDK (without using the --platformsdk switch) and leverage the enhancements available in iOS 5.1 SDK.

Exception Support in Native Extensions for iOS

- A native extension for iOS can now use both C++ and Objective-C exceptions. It is up to the extension to catch all the exceptions thrown in its code. The runtime will not catch the exceptions thrown by extensions.

ADT option to list the attached mobile devices

- A new option, devices has been added in ADT to list the attached iOS/Android devices. Users can avail this option instead of using adb for listing the Android devices and idb for listing the iOS devices.

Licensing support: Flash Player Premium Features for Gaming (Flash Player only)

- Supports licensing for the combined use of domain memory (used by some third party tools and the Project "Alchemy" C/C++ compiler) with Stage3D hardware acceleration in Flash Player.

Builds changes

Runtime Versions

Flash Player Desktop only: **11.4.400.231**
AIR Runtime Desktop: **3.4.0.2200**
AIR Runtime Android: **3.4.0.2200**

AIR SDK : **3.4.0.2200**

Security Enhancements

Adobe Flash Player 11.4.400.231 and AIR 3.4.0.2200 include security enhancements described in Security Bulletin APSB12-14

Known Issues

- Applications using Native extensions might sometimes crash on iOS when using certain external libraries (eg. Libxml2.dylib), the error on XCode console being 'dyld: Symbol not found:' . The workaround is to use -platformsdk switch while packaging an application using these ANEs. (3226974)
- Memory usage of H264Video increased about 30% on Win 7 (3209254)
- Videos from some websites cannot be played on FF or Chrome (3203907)
- Browser crashes when video is launched in full screen mode or exit from full screen intermittently (3205129)
- AIR crash when invoke methodinfo.cpp for reflection work (3225021)
- When packaging an application using native extensions for iOS using Flash Pro, an error dialog saying "Error creating files" may come up, though the app is packaged. Please ignore this error.

Fixed Issues

- GPU mode is broken on iOS 6 (3211902)
- FLV Videos not playing on flash player (3187569)
- DNSResolver doesn't return PTRRecord value for machine IP address on Windows (3188360)
- iOS5 - TextFields with embedded fonts garbled on mobile devices (3161138)

Authoring

Authoring for Flash Player 11.4

To use the new Flash Player, you will need to target SWF version 17 by passing in an extra compiler argument to the Flex compiler: -swf-version=17. Directions are below. If you are using the Adobe Flex SDK:

- Download the new playerglobal.swc for Flash Player 11.4
- Download Flex 4.5.1 SDK (4.5.1.21328) from the Flex 4.5 SDK table.
- Install the build in your development environment
- In Flash Builder, create a new ActionScript project: File -> New -> ActionScript project.
- Open the project Properties panel (right-click and chose 'Properties'). Select ActionScript Compiler from the list on the left.
- Use the 'Configure Flex SDK's' option in the upper right hand corner to point the project to Flex build 21328. Click ok.
- Configure your project to target SWF version 17
- Open the project Properties panel (right-click and chose 'Properties'). Select ActionScript Compiler from the list on the left.
- Add to the 'Additional compiler arguments' input: -swf-version=17. This ensures the outputted SWF targets SWF version 17. If you compile on the command-line and not in Flash Builder, you need to add the same compiler argument.
- Ensure you have installed the new Flash Player 11.4 build in your browser.

Authoring for AIR 3.4 Update to the AIR 3.4 namespace

You must update your application descriptor file to the 3.4 namespace in order to access the new AIR 3.4 APIs and behavior. If your application does not require the new AIR 3.4 APIs and behavior, you are not required to update the namespace. However, we recommend all users start using the AIR 3.4 namespace even if you are not yet taking advantage of the new 3.4 capabilities. To update the namespace, change the xmlns attribute in your application descriptor to: <application xmlns="http://ns.adobe.com/air/application/3.4">

Feature Usage Guidelines

ActionScript Workers

Getting started with Workers

To get you started with the workers feature, please refer to the examples in the flashplayer11-4_p1_ex_concurrency.zip file.

Note

Here is below the memory footprint measured per worker on MacOS and Windows :

#'s of workers	Mac BookPro Mac OS X 10.7.3 2.4 GHz Intel Core i7 8GB RAM **Mac OSX 10.7 memory (MB)**	PC machine Windows 7 32bit Intel Xeon Quad core 3.2GHz 2048MB RAM **Windows 7 memory (MB)**	Lenovo T61 laptop Windows XP Intel Core Duo Core 2.2GHz 2014MB RAM **Windows XP memory (MB)**
0	10	4.78	4.39
1	15.75	9.33	8.59
2	21.47	13.8	12.77
3	27.28	18.35	16.39
4	33	22.8	20.67
5	38.81	26.39	24.58
6	44.25	31.1	28.8
7	50	35.62	32.67
8	55.89	40.16	35.75
9	61.7	44.73	39.6
10	67.47	48.83	43.9

When code is executing within the context of a non-primordial worker (i.e. the background), some `flashruntime` API behavior will be different from that of Actionscript code that executes in the main thread. The following APIs will behave differently when used from within a background worker.

Non-functional APIs

The following APIs will not be available from within a background worker. Any attempt to construct an instance of any of these will throw an IllegalOperationError with the message "This feature is not available within this context," the errorID will be the same in all instances, allowing developers to key off of this value.

- flash.desktop.Clipboard // calling constructor will throw; calling generalClipboard will return null
- flash.desktop.NativeDragManager // isSupported returns false
- flash.desktop.Updater // isSupported returns false
- flash.display.NativeMenu // isSupported returns false
- flash.display.NativeWindow // isSupported returns false
- flash.display.ToastWindow // can't access instance because stage.window will never be defined
- flash.display.Window // can't access instance because stage.window will never be defined
- flash.external.ExtensionContext // createExtensionContext() will always return null or throw an error
- flash.external.ExternalInterface // available returns false
- flash.html.* // HTMLLoader.isSupported returns false
- flash.media.CameraRoll // supportsAddBitmapData and supportsBrowseForImage returns false
- flash.media.CameraUI // isSupported returns false
- flash.media.StageWebView // isSupported returns false
- flash.net.drm.* // DRMManager.isSupported returns false
- flash.printing.* // PrintJob.isSupported returns false
- flash.security.XMLSignatureValidator // isSupported returns false
- flash.system.IME // isSupported returns false
- flash.system.SystemUpdater // calling constructor throws
- flash.text.StageText // calling constructor throws
- flash.ui.ContextMenu // isSupported returns false
- flash.ui.GameInput // isSupported returns false
- flash.ui.Mouse // all methods are no-ops; setting 'cursor' property is a no-op

Behavioral changes to APIs

The following APIs have modified behavior when running from within a background worker. Some calls on methods are no-ops, while others will throw an IllegalOperationError or equivalent as is consistent with the documented API. Return values should be consistent with the documentation to the extent possible. For example, if a method returns an Array with elements in it under normal conditions, when executing from within a background worker, it will return an empty Array.

- flash.accessibility.Accessibilty
    - active // always returns false
    - updateProperties(); // no-op
- flash.desktop.NativeApplication** properties

        - supportsDefaultApplication, supportsDockIcon, supportsMenu, supportsStartAtLogin, supportsSystemTrayIcon // all return false
        - activeWindow // returns null
        - autoExit // setter is a no-op
        - idleThreshold // setter is a no-op
        - openedWindows // returns an empty Array []
        - systemIdleMode // setter is no-op
    - methods
        - activate(), clear(), copy(), cut(), paste(), selectAll() // no-op
        - exit() // forces this background worker to stop and shutdown
    - events
        - only exiting event is supported – this event will get dispatched when the worker is shutting down (happens either with life cycle method calls or “main” worker is shutting down, i.e. runtime is exiting; not cancelable from background worker)
- flash.display.Stage
    - allowsFullScreen // always returns false
    - stage3Ds // always returns empty Vector
    - stageVideos // always returns empty Vector
    - supportsOrientationChange // always returns false
    - wmodeGPU // always returns false
- flash.filesystem.File
    - browseForDirectory(), browseForOpen(), browseForOpenMultiple(), browseForSave() // no-op
- flash.net.FileReference** browse(), download(), save() // no-op; always returns false
- flash.net.FileReferenceList** browse() // no-op; always returns false
- flash.system.System
    - ime // always returns null
    - exit() // forces this background worker to stop and shutdown
    - setClipboard() // no-op

Increased GPU Reach - Constrained Stage3D profile

A new parameter to Context3D has been introduced called "profile". This parameter is supplied to the Stage3D.requestContext3D() function and you can give it one of two value Context3DProfile.BASELINE ("baseline") which will return the typical Context3D that has existed in previous releases, or Context3DProfile.BASELINE_CONSTRAINED ("baselineConstrained") which will return a new kind of Context3D capable of running with hardware acceleration on previously unsupported GPU's. Constrained profile enables content authors to create Stage3D content targeting older systems using graphics cards such as Intel GMA 900/950 on Windows. The trade-off for targeting lower end hardware with Constrained profile is that the Stage3D capabilities will be limited in order to guarantee that content runs well with that hardware. Cards such as the GMA 900/950 only support pixel/vertex shader version 2.0 which forces the following limits on Stage3D when in Constrained profile:

- Limited to 64 ALU and 32 texture instructions per shader.
- Only 4 texture read indirections per shader.
- Limited to a smaller amount of constants/varying/temps per shader.
- No support for predicate register. This affects sln/sge/seq/sne and needs to be replaced with compound mov/cmp instructions which are available with ps_2_0 (this will be done under the hood at the cost of using more instruction slots compared to ps_2_x).
- The Context3D back buffer must always be within the bounds of the stage.
- Only one instance of a Context3D running in Constrained profile is allowed within a Flash Player instance.
- As with mobile the classic Flash Display List elements will not be updated while constrained Stage3D content is visible unless Context3D.present() is called.

'Profile' is the second optional parameter of the requestContext3D method. Developers can request profile "baselineConstained" to get a Context3D in constrained mode, as the api below shows: *function requestContext3D (context3DRenderMode : String = "auto", profile : String = "baseline") : void;*
Example:

```
stage.stage3Ds[0].addEventListener(Event.CONTEXT3D_CREATE, createdHandler);
stage.stage3Ds[0].requestContext3D(Context3DRenderMode.AUTO, Context3DProfile.BASELINE_CONSTRAINED);
```

Direct AIR deployment using ADT(without iTunes)

Installation:
To install an IPA on a device, use the following command:
adt -installApp -platform ios -device <deviceID> -package <ipa-file>
Uninstall:
To uninstall an IPA from a device, use the following command:
adt -uninstallApp -platform ios -device <deviceID> -appid <app-id>
<deviceID> of your connected device can be found using the command :
@SDKLocation/lib/aot/bin/iOSBin/idb -devices
<app-id> is specified in the application descriptor file (app.xml) within the <id> tag . If you create your project using Flash Builder , by default, it would be the same as your app name.
NOTE : Location of idb binary has changed from @SDKLocation/lib/aot /idb/ to @SDKLocation/lib/aot/bin/iOSBin/
If only one device is connected to the desktop, omit the '-device <deviceID> ' argument and the application will be installed/uninstalled on the

connected device.
adt -installApp -platform ios -package <ipa-file>

iOS Push Notifications

1) The app identifier (id tag) is the app-xml should be same as the application identifier of the mobile provisioning certificate enabled with Push notifications.
2) Include the aps-environment entitlement in your app descriptor:

```
<iPhone>
        <Entitlements>
                <![CDATA[
                        <key>get-task-allow</key>
                        <true/>
                        <key>aps-environment</key>
                        <string>development</string>
                        <key>keychain-access-groups</key>
                        <array>
                       <string>KEYCHAIN_ACCESS_GROUP</string>
                        </array>
                ]]>
        </Entitlements>
</iPhone>
```

For submitting the app to App Store with Push Notifications enabled, one needs to set the aps-environment to production. The developer must use the provisioning profile with Push notifications enabled. Cross check the Entitlements of the signed app using the following command.

codesign -d --entitlements - <YourAppName>.app

3) Use the newly added flash.notification class in the application to enable push notification support. For example,

```
_if (_ _[RemoteNotifier]__.supportedNotificationStyles \!= NULL )_
_{_
_var preferredNotificationStyles:Vector.<String> = new Vector.<String>();_
_preferredNotificationStyles.push(__[NotificationStyle]__.ALERT);_
_preferredNotificationStyles.push(__[NotificationStyle]__.SOUND);_
_preferredNotificationStyles.push(__[NotificationStyle]__.BADGE);_
_var subscribeOptions:__[RemoteNotifierSubscribeOptions]_ _= new_
_[RemoteNotifierSubscribeOptions]__();_
_subscribeOptions.notificationStyle = preferredNotificationStyles;_
_var rn:__[RemoteNotifier]_ _= new_ _[RemoteNotifier]__();_
_rn.addEventListener(__[RemoteNotificationEvent]__.TOKEN, tokenEventHandler);_
_rn.addEventListener(__[RemoteNotificationEvent]__.NOTIFICATION, notificationEventHandler);_
_rn.subscribe(subscribeOptions);_
_}_

_function tokenEventHandler(event:__[RemoteNotificationEvent]__):void_
_{_
_// AIR developer can send the tokenId received in event.data to his server via URL request_
_var urlRequest:URLRequest;_
_var urlLoader:URLLoader = new URLLoader();_
_var urlString:String = "_[https://url.com/api/device_tokens/|https://url.com/api/device_tokens/]_" +
event.tokenId;_
_urlRequest = new URLRequest(urlString);_
_urlRequest.authenticate = true;_
_urlRequest.method =_ _[URLRequestMethod]__.PUT;_
_[URLRequestDefaults]__.setLoginCredentialsForHost("url.com",<userId>,<password>) ;_
_urlLoader.load(urlRequest);_
_urlLoader.addEventListener(__[IOErrorEvent]__.IO_ERROR, onError);_
_urlLoader.addEventListener(Event.COMPLETE, onComplete);_
_urlLoader.addEventListener(__[HTTPStatusEvent]__.HTTP_STATUS, onStatus);_
_}_
```

The client application can store in its bundle the alert message strings translated for each localization it supports. The provider specifies the *loc-key* and *loc-args* properties in the aps dictionary of the notification payload. When the device receives the notification, it uses the aps

dictionary properties to find and format the string localized for the current language which it then displays to the user.

Localized strings **must** reside in the file called Localizable.strings in the appropriate <language>.lproj folder. Each entry in this file has a key and a localized string value, the string can have format specifiers for the substitution of of variables values. When an application asks for a particular string, it gets the resource that is localized for the language currently selected by the user.

In order to receive localized remote notifications, an AIR application must have localized key value pairs in Localizable.strings file in the <language>/lproj folder inside the ipa. To add the Localizable.strings in the ipa, the developer just needs to add the respective <language>.lproj folder in the adt command as follows:

adt -package -target ipa-app-store -provisioning-profile <MobileProvisioningCertificate> -storetype pkcs12 -keystore <Certificate.p12> -storepass <PASSWORD> app.xml sample.swf en.lproj es.lproj fr.lproj

Please note that lproj folder needs to be present in the current directory to let the adt merge the files in appropriate lproj folder. If the lproj folder is given as a/b/c/en.lproj ,then adt adds this as a/b/c/en.lproj and hence localized key value pairs won't be accessible to the application.

ADT option to list the attached mobile devices

You can use the following command to get information about the attached iOS/Android devices:

```
adt -devices -platform (android | ios)
```

Using this command produces an output like the following:

```
List of attached devices:
Handle DeviceClass DeviceUUID     DeviceName
   7 iPhone   783fc8af34a6022c924c34fe7025e7e39d9d123b abc's iPhone
```

Licensing support: Flash Player Premium Features for Gaming (Flash Player only)

The combined use of domain memory (used by some third party tools and the Project "Alchemy" C/C++ compiler) with Stage3D hardware acceleration will require a license. When using both features in combination, an informational watermark will be displayed in the Flash Player content debugger (debug player). This licensing requirement does **not** apply to content that uses Stage3D hardware acceleration without domain memory (or domain memory without Stage3D) and **does not apply** to any content deployed using Adobe AIR.

Handling device loss

Unlicensed content will continue to run by automatically switching Stage3D to software rendering. To make sure your content correctly handles loss of the graphics card resource, listen to the Event.CONTEXT3D_CREATE event on the Stage3D object to re-initialize your content's graphics. You can simulate a device loss event by calling context3D.dispose(). (All Stage3D content should handle device loss, which can also occur if a computer is locked or if the screen saver activates.)

Pre-release AIR runtime app distribution

Please do not ship applications using the pre-release runtime. The pre-release program is intended for you to test your app and to share your feedback with us. Under the NDA participants have agreed to, you cannot use the prerelease builds for commercial purposes, and apps cannot be distributed with this software (including apps using captive runtime support).

Reporting issues

Found a bug? Please submit a bug to the Flash Player and Adobe AIR* *bug database**.** Flash Player and AIR may leverage your graphics hardware to decode and play H.264 video. There may be video issues that can only be reproduced with your particular graphics hardware and driver. When reporting an issue involving video, it is essential to note your graphics hardware and driver, along with your operating system and browser (when using Flash Player), so that we can reproduce and investigate issues. Please be sure to include this information as described in Instructions for Reporting Video Playback Issues. Note: Due to the high volume of email we receive, we are unable to respond to every request. Thank you for using Adobe Flash Player and AIR and for taking the time to send us your feedback!

System Requirements

For current Flash Player system requirements visit http://www.adobe.com/products/flashplayer/systemreqs/For current AIR system requirements, visit http://www.adobe.com/products/air/systemreqs/

Flash Player 11.4 has the following minimum system requirements:

	Windows	Macintosh
Processor	2.33 Ghz or faster x86-compatible processor, or Intel® Atom™ 1.6GHz or faster processor for netbook class devices	Intel® Core™ Duo 1.83GHz or faster processor
Operating System	Microsoft® Windows® XP (32-bit), Windows Server® 2003 (32-bit), Windows Server 2008 (32-bit), Windows Vista® (32-bit), Windows 7 (32-bit and 64-bit)	Mac OS® X 10.6, Mac OS X 10.7
Browser	Internet Explorer 7.0 and above, Mozilla Firefox 4.0 and above, Google Chrome, Safari 5.0 and above, Opera 11	Safari 5.0 and above, Mozilla Firefox 4.0 and above, Google Chrome, Opera 11
Memory	128MB of RAM (1GB RAM recommended for netbook class devices), 128MB of graphics memory	256MB of RAM, 128MB of graphics memory

AIR 3.4 has the following minimum system requirements:

	Windows	Macintosh	Android	iOS
Processor / Device Hardware	2.33GHz or faster x86-compatible processor or Intel Atom™ 1.6GHz or faster processor for netbook class devices	Intel® Core™ Duo 1.83GHz or faster processor	ARMv7 processor with Vector FPU, Minimum 550MHz, OpenGL ES2.0, H.264 & AAC H/W Decoders	iPod touch (3rd generation) 32 GB and 64 GB model, iPod touch 4, iPhone 3GS, iPhone 4, iPad, iPad 2
Operating System	Microsoft® Windows® XP, Windows Server® 2003, Windows Server® 2008, Windows Vista® Home Premium, Business, Ultimate, or Enterprise (including 64-bit editions) with Service Pack 2, or Windows 7	Mac OS® X 10.6 and 10.7	Android 2.2, 2.3, 3.0, 3.1, 3.2 & 4.0	iOS 4.2 and higher
RAM	512MB of RAM (1GB recommended)	512MB of RAM (1GB recommended)	256MB RAM	-