Adobe AIR SDK Release Notes

Version	33.1.1.533
Date	25 June 2021
Document ID	HCS19-000287
Owner	Andrew Frost

Table of contents

1 Purpose of the Release

This is an official release of the Adobe AIR SDK software, provided by HARMAN under the terms of the “AIR SDK License Agreement”. This software may be used to create AIR applications for distribution to end users.

For macOS users on 10.15+, the SDK may not work properly unless the quarantine setting is removed from the SDK: `$ xattr -d -r com.apple.quarantine /path/to/SDK`

There are a large number of changes and fixes in this release, the most notable areas are:

- Linux support, for commercial licensees – the AIR SDK can now be used, and captive-bundled AIR applications created and run, on Linux x86_64 platforms.
- Android: now supporting the requirements for API level 30, including the App Bundle format, and developers can include support for Play Asset Delivery packages.
- iOS: updating to SDK 14.5 and with additional fixes in for code-signing
- Windows: significant improvements in multimedia and reduction in memory leaks/instabilities

For developers who are packaging applications for desktop AIR, there is now a shared AIR runtime that is available for end users to download at https://airsdk.harman.com/runtime. However, we continue to recommend that applications are packaged up with the captive bundle mechanism to include the runtime and remove the dependency upon this shared package.

Any issues found with the SDK should be reported to adobe.support@harman.com or raised on https://github.com/airsdk/Adobe-Runtime-Support/issues.

The website for AIR SDK is available at: https://airsdk.harman.com.

2 Release Information

2.1 Delivery Method

This release shall be delivered via the AIR SDK website: https://airsdk.harman.com/download

2.2 The Content of the Release

2.2.1 Detailed SW Content of the Release

Name	Version
Adobe AIR SDK – for Windows	33.1.1.533
Adobe AIR SDK – for Mac	33.1.1.533
Adobe AIR SDK – for Linux	33.1.1.533
Adobe AIR SDK for Flex Developers – for Windows	33.1.1.533
Adobe AIR SDK for Flex Developers – for Mac	33.1.1.533
Adobe AIR SDK for Flex Developers – for Linux	33.1.1.533

2.2.2 Delivered Documentation

Title	Document Number	Version
Adobe AIR SDK Release Notes	HCS19-000287	33.1.1.533

2.2.3 Build Environment

Platform	**Build Details**	
Android	Target SDK Version:	30
	Minimum SDK Version:	14 (ARMv7, x86); 21 (ARMv8, x86_64)
	Platform Tools:	28.0.3
	Build Tools:	30.0.3
	SDK Platform:	Android-30
iOS	iPhoneOS SDK Version:	14.5
	iPhoneSimulator SDK Version:	14.5
	XCode Version:	12.5
	Minimum iOS Target:	9.0
	Note, files were actually compiled using Xcode 11.2 with the toolchain from Xcode 12.1, in order to ensure continued support for 32-bit binaries.	
MacOS	MacOS SDK Version:	11.3
	XCode Version:	12.5
	Minimum macOS Target:	10.7
Windows	Visual Studio Version:	14.0.25431.01 Update 3
Linux	GCC Version	5.4.0 20160609 (Ubuntu 16.04.5)

2.3 AIR for Linux – Restrictions

The AIR SDK now supports some capabilities on Linux platforms. This is only available to developers with a commercial license to the SDK, and has some restrictions:

- No “shared runtime” support: applications would need to be built as ‘bundle’ packages with the captive runtimes
- Currently only x86_64 support – ARM64 is planned and potentially 32-bit variants if needed
- Packaging into native installers (“native” target type for .deb or .rpm files) is currently not working: please create a “bundle” target and use Linux tools to distribute these

The Linux functionality has not been as widely tested and is provided “as-is” – developers are free to distribute applications built using the SDK, and please report any issues found.

2.4 AIR for Flex users

HARMAN have continued Adobe’s strategy of issuing two AIR SDKs per platform: the first of these (“`AIRSDK_[os].zip`”) contains the newer ActionScript compiler and is a full, self-contained SDK for compiling and packaging AIR applications. The second of these is for combination with the Flex SDK (“`AIRSDK_Flex_[os].zip`”) which doesn’t include a number of the files necessary for ActionScript/MXML compilation. These SDKs should be extracted over the top of an existing, valid Flex SDK.

See instructions at https://helpx.adobe.com/uk/x-productkb/multi/how-overlay-air-sdk-flex-sdk.html.

NOTE when copying an AIR 33.1 SDK over a previous version, there are folders in AIR 33.1 that have the same name as files from previous versions of the SDK (MainWindow.nib and MainWindow-iPad.nib). If there are errors such as

“SDK is missing file objects-13.0+.nib”

then please check that these folders have been properly copied over and contain the objects-13.0+.nib file.

Advance warning: a later version of XCode has reverted this change in format so the same problem may occur in a future release of AIR..

3 Changes and Issues

3.1 Changes in this Release

3.1.1 Runtime

No changes – the below details will be fixed for all “33.1” version numbers:

Namespace: **33.1**

SWF version: **44**

The namespace and SWF version updates are made across all platforms and may be used to access the updated ActionScript APIs that will be introduced in future beta releases of 33.1.

Note that using “33.0” as a namespace is not valid, and is resulting in behaviors such as VerifyErrors being dispatched at application start-up with built-in class names such as ExtensionContext, Context3D, and others.

3.1.2 Build Tools

Android building has been updated to API/platform level 30.

iOS building is still done on Mojave currently so that AIR can continue to support 32-bit iPhone/iPad devices; the build tools and SDKs are now take from Xcode 12 though. The next release is likely to remove support for 32-bit iOS platforms so that this can use the latest Xcode and SDK platforms.

3.1.3 AS3 APIs

Updated AS3 APIs are described in section 12.

AIR-4486: There is a new setting, “flash.system.System.poisonStrings” – this will change the internal mechanism of the ActionScript virtual machine so that any string created when this flag is set will be created as a new string (rather than using the ‘dependency’ mechanism that optimises substrings etc) and will be overwritten by ‘poison’ (0xDD bytes) when garbage collection is called on the object. This is intended to be used when an application is about to request a password, to ensure the entered password doesn’t remain within cached memory.

Additionally, the File class has a new static member, “workingDirectory”, which is valid for desktop platforms and will provide the folder from which a command-line version of the runtime had been launched.

3.1.4 Features

Android App Bundle: this is now supported via the creation of a temporary folder structure that is then compiled via Gradle to create the bundle. In order to support this with IDEs that don’t provide the appropriate options, an updated configuration file option is provided, see section 5.

AIR-207: Additional code-signing options can be passed in to ADT When creating a macOS ‘bundle’ package. This would be the ‘native’ signing options, after the ‘-target bundle’ parameters, and would need to be a Keychain store type with the alias being the name of the certificate to use in code-signing. Using this means that the output bundle should be fully structured and code-signed, and ready for notarization.

AIR-3956: Adding support for Asset Pack creation in an App Bundle via the app descriptor. See 7.2;

The command-line support has been extended to macOS so that AIR applications can be launched via ADL with a “-cmd” parameter that will prevent any UI elements from being created. Any references to the Stage, or similar classes, will throw an error, and there is no longer a NativeWindow being created. To help with command-line applications, a “File.workingDirectory” property has been exposed for desktop builds.

Gamua-260: Adding Android support for mouse wheel events (note that middle and right mouse click events may be captured by the OS and interpreted as hardware keys).

AIR-4490: Adding support for ADT license information to be passed on the command line. Currently ADT will look for a file “adt.lic” in the same folder as the adt.jar file; however the license can be provided now via a parameter “-licenseFile” followed by the file path, or via a parameter “-licenseDevID” that should be followed by the developer ID string (ADT will then check online for the validity of this ID). These options should be the first items on the ADT command line.

3.1.5 Bug Fixes

AIR-276: Updating skia library for x64 builds to fix font problem

AIR-319: Fixing architecture/cpu address size capabilities on macOS/iOS (Gamua-859)

AIR-380: Using an old AECM build that works on Android 4.0 (Gamua-278)

AIR-3379: Signing APK files with apksigner for code signature v2

AIR-4200: Swift support signing to correct the plist value in the sha256 hash (Gamua-776)

AIR-4311: Ensuring the resources.arsc file is not compressed

AIR-4469: Updating -XO flag to work with -XO0 overriding -O3 on ad-hoc/app-store builds

AIR-4478: Android App Bundle code-signing support

Gamua-53: Eliminating crash when using Logitech webcams in 64-bit AIR on Windows

Gamua-309: Set contentsScaleFactor per window instead of player wide so the scale factor is correct for windows on different monitors.

Gamua-572: fixing netstream memory leaks on macOS

Gamua-695: Preventing AIR requesting font names that are reserved

Gamua-817: Fixing certificate OID details to properly remove MD2

Gamua-823: Fixing crash in Android window visibility changed handler

Gamua-833: Fixing AIR application installs for apps with Migration signatures

Gamua-843: Ensuring MacOS ANEs with universal binaries work from MacOS-x86-64 platform folder

Gamua-857: Ensuring videos still play if the first frame decoded has a non-zero PTS

Gamua-860: Fixing crash on Netstream.dispose caused by double-delete of YUV plane

Gamua-868: Ensuring video objects can play even when not attached to the stage

Gamua-891: Warning on IPA resources folder, ANE with Linux-x86-64,

Gamua-892: SwiftSupport libraries copied as-is from the input Frameworks

Gamua-917: Ensuring device simulator profiles are included in AIR SDK runtime on Windows

Gamua-919: Updating config file for Platform SDK and Java Home support

3.2 Known Problems

Most H.264/AAC content is now working on desktop platforms; work has now started on revising the multimedia capabilities.

There are currently reports of ANR issues on Android; some investigations have begun into whether the architecture of the AIR runtime on Android can change to an asynchronous/background-threaded approach, and some advanced diagnostics will be created to try to ensure we can find out what is the main cause of the reported ANRs.

For a list of open issues, see https://github.com/airsdk/Adobe-Runtime-Support/issues

3.3 Previous Changes

3.3.1 AIR 33.1.1.476

Gamua-543: Account for TextField horizontal scroll when calculating caret position.

Gamua-754: Ensuring an ANE can have a Mac implementation without native library

Gamua-758: Updating openssl libraries to ensure they are able to load the generated cert bundle

Gamua-777: Adding android.jar library path to d8 command if platformsdk is provided

Gamua-779: multimedia clean-up for memory leaks when handling Video + StageVideo

Gamua-780: Reverting to use android-21 platform for the NDK

AIR-596: Reducing D8/DX command line length when multiple extensions are used

AIR-3226: AIR SDK for MacOS to support M1 i.e. universal binaries

AIR-3801: AIR Android - adding File.applicationRemovableStorageDirectory property

AIR-4183: File/Directory browsing broken on Big Sur

3.3.2 AIR 33.1.1.444

Gamua-521: Fixing code signatures for multiple frameworks in an IPA

Gamua-564: adding a name to images saved to the Android camera roll

Gamua-713: Permission error when requesting file permission on Big Sur

Gamua-569, Gamua-714: Updating linker command line for iPhoneSimulator builds

AIR-551: The curl and openssl libraries used with AIR have been updated, with a view to fixing some of the crashes that had been reported with https access on Android.

AIR-1626 (Gamua-511, Gamua-590, Gamua-676, Gamua-708): Mach-O code signing updates for IPA files

AIR-3591 (Gamua-526): MacOS file open filter does not work on Catalina

AIR-3605: Updating FileReference to use the URLStream idleTimeout value

AIR-3716 (Gamua-696): Adding support for JavaXmx setting in config file for Android builds

3.3.3 AIR 33.1.1.406

AIR-502 (Gamua-532) Support for camera and microphone on MacOS Big Sur.

AIR-662: adding support for a <resdir> element in the application descriptor file.

AIR-1626: updating IPA CodeResources signature format (work in progress for Gamua-590)

AIR-3434 (Gamua-674): InfoAdditions in application xml for macOS.

HTTP Status 307 and 308 handling: these are now correctly identified, and the redirects forwarded with appropriate method verb.

Internal updates within packaging and XML signature verification to allow .air apps to be installed.

Gamua-653: ensuring browseForOpen and browseForSave are not affected by permission updates

3.3.4 AIR 33.1.1.345

Camera.names crash when using OBS

Gamua-170: Fixing DPI change issues causing windows to disappear

Gamua-251: Adding support for audio/AAC playback so MP4s work on Windows

Gamua-461: Adding command-line support for AIR

Gamua-515: Exporting the symbol '__mh_executed_header' to solve third party integration issues

Gamua-525: Fixing JIT issue causing a large number of access errors in ARM64 Android

Gamua-532: Adding support for permissions manager on MacOS Big Sur

Gamua-540: Fixing crash in audio callback during RTMP 'stop'

3.3.5 AIR 33.1.1.300

This update release brings the iOS 14 capabilities to the AIR SDK, with the updated stub libraries and settings being generated in the IPA files based on Xcode 12. It is still possible to target 32-bit iPhone/iPad platforms, as long as the development platform supports 32-bit processes i.e. anything prior to MacOS Catalina. The minimum iOS level now supported is 9.0.

Gamua-170: Fixing scaling/resizing problems between multiple monitors

Gamua-340: Fix for stutter with stationary touch on touchscreen Windows devices

Gamua-392: Fixing y-flip error when rendering to bitmap

Gamua-476: Crash in Texture.fromColor()

Gamua-507: Protecting against async handlers accessing a destroyed activity

AIR-430: Implement safepoint polling for ARM64 (Gamua-286)

AIR-662: Packaging any content in a 'res' folder as if it's a resource (Gamua-163). Following feedback from the last release, this feature has been updated to look for a "-resdir" command-line argument, and to use this as a location for packaging in resource files/folders under the given folder name.

AIR-1024: Supporting 256x256 icon for Windows bundles (Gamua-462)

AIR-1043: Ensure license check copes with multi-instance apps (Gamua-455)

AIR-1046: HLS streaming fails in iOS14 (Gamua-466)

AIR-1105: Fixing Transform.perspectiveProjection to avoid invalid objects being created

AIR-1226: Ensuring enhanced microphone works despite AEC failure (Gamua-487)

3.3.6 AIR 33.1.1.259

Gamua-113 Adding libclang_rt.ios.a to SDK (for convenience..)

Gamua-330: RTMP video streams stuttering/hanging

Gamua-452: Ensuring Android splash screen check uses correct AppID

AIR-446: Audio doesn't restart after phone interruption (Gamua-161)

AIR-662: Packaging any content in a 'res' folder as if it's a resource (Gamua-163)

AIR-931 Ensuring .air files can be generated into installation packages

AIR-938: Updating AIR on Windows to handle up to 6 concurrent HTTP connections per server

3.3.7 AIR 33.1.1.217

Gamua-372: Changing default Android target SDK to 29

Gamua-376: Fixing signing problem when using ANE frameworks and storyboards together

Gamua-392 (partial): Fixing y-flip error when rendering to bitmap (CPU version only so far)

Gamua-398: Updating generated IPA DT_XCODE value to Xcode 11.2.1

AIR-305: Ensuring we release render target D3D11 pointer to avoid GPU memleak (Gamua-20)

AIR-329: Updating ELS to cope with UTF-8 filenames (Gamua-165)

AIR-428: Adjusting order of injected launch storyboard to prioritise over launch images

AIR-479: Changing NEON selection mechanism to avoid SELinux issue (Gamua-372)

AIR-635: Adding support for -simulator option in ADT.

3.3.8 AIR 33.1.1.190

Gamua-371: IPA won't compile with 33.1.1.176 when Fast Packaging is off

Gamua-374: iOS Device not found AIR 33.1.1.176

Gamua-375: java.lang.NullPointerException - AppEntry.dispatchKeyEvent

Gamua-377: WARNING: Unlicensed version of AIR SDK

Gamua 378: Unable to build iOS ipa File with 33.1.1.176

AIR-282: Proximity on IOS blocks event processing when enabled

AIR-568: Encrypted local store - can't create new ELS on Windows (Gamua-205)

3.3.9 AIR 33.1.1.176

AIR-256/Gamua-1: Removing allocation of strings when getting a vector type

AIR-269: Moving SQLite into namespace to avoid OS conflicts (Gamua-218)

AIR-328: Implementation of ELS for Windows + MacOS (Gamua-205)

AIR-349: Ensuring ADT can install onto new simulator devices (see Gamua-201)

AIR-359: Updating ADT to use the normal ‘ld’ command on macOS for linking (see Gamua-113)

AIR-428 Moving iOS splash screen into a Storyboard (see Gamua-333)

AIR-483/Gamua-331: ensuring iOS apps with non-ascii names don’t end up with bad filename

AIR-495: Fixing crash when large number of files are chosen in Windows file browser dialog

Gamua-287: Installer on MacOS now copes with iPhone XR and the newer UDID format.

Gamua-320: Fixing crash when enumerating cameras with Logitech driver

Gamua-338: Updating splash images to avoid crash on launch in Pixel XL devices

Gamua-349: Fix for Chinese-Traditional language code on iOS

Fix for crash in iOS audio disconnection

3.3.10 AIR 33.1.1.98

Gamua-240: Ensuring 'activate' event is sent at startup on Windows after splash screen

Gamua-283: Fixing issue with iOS device installation log confusing IDEs

Gamua-285: Preventing splash screen from distorting on Android

Gamua-285: Preventing splash screen from remaining on the display when debugging

Gamua-286: Partial fix: ensuring AS3 functions don't crash due to ByteArray length sychronisation

Gamua-287: Ensuring we only report real errors from libimobiledevice installation

Gamua-287: Reformatting iIPA output so that it can be parsed by Animate

Gamua-287: Ensuring iIPA.exe works with Flash Builder on Windows

3.3.11 AIR 33.1.1.86

AIR-282: Ensuring proximity sensor doesn't completely block the AIR process when enabled (Gamua-138)

AIR-380: Fixing crash when using AECM microphone on Android ARMv7 (Gamua-226)

AIR-388: Crash in iOS AOT builds due to flash.geom.Transform API update (Gamua-270)

AIR-394: Reverting fix for AIR-168 to avoid context recreating when bringing app to foreground (Gamua-256)

AIR-395: Fix for remote notification event token format in iOS 13.0 (Gamua-263)

AIR-397: Adding synchronization to cache access in Android app directory manager (potential improvement for Gamua-167)

Gamua-112: Enhancing iIPA tool and device connectivity mechanisms

Gamua-240: 33.1.1.50 on Windows 10 only splash screen is displayed and black screen.

Gamua-258: AIR 33.1.1.63 - iOS - Immediately crashing on startup

Gamua-259: [Android IOS] AIR SDK 33.1.1.63 when use ANE crashing on startup

Gamua-269: App area scaled completely wrong on ios.

Gamua-277: [Android iOS][33.1.1.63]Display size

3.3.12 AIR 33.1.1.63

AIR-313: Object pooling for geometry APIs – adjusted how these are defined

AIR-379: Problems with AIR 33.1 launch including black screen, invalid splash screen display, crash after short duration (Gamua-240, Gamua-249, Gamua-231)

3.3.13 AIR 33.1.1.50

AIR-354: Crash when changing orientation in background (Gamua-230)

Gamua-231: Splash screen appears even on commercially licensed SDKs

Gamua-234: Packaging tvOS applications failure

3.3.14 AIR 33.1.0.43

AIR-310: Remove Stage3D resource limits for apps using namespace 33.1

AIR-313: Object pooling for geometry APIs

Gamua-227: Crash in loading SWF with embedded resources

3.3.15 AIR 33.1.0.37

AIR-168: AIR content goes all white/blank after AR camera closes (Gamua-67)

AIR-210: Splash screen improvements [pending on iOS]

AIR-263: iIPA process cannot uninstall an application from iOS13

AIR-296: Yet another fix to protect from crash in audio code on Android

AIR-308: Wrap up libimobiledevice for installation on iOS (Gamua-112)

AIR-346: Problems with Android 32-bit ANE development on SDK 33 (Gamua-217)

3.3.16 AIR 33.1.0.16

AIR-296: Adding further fix to protect from crash in audio code

AIR-300: Preventing hang when switching Wi-Fi connection when RTMFP is being used in a Worker (Gamua-96)

AIR-311: Fixing handling of invalid data passed to Font.registerFont() (Gamua-153)

AIR-312: Trying to protect against java.lang.NullPointerException on Android 9 (Gamua-70)

3.3.17 AIR 33.0.2.338

AIR-304: Correcting detection of Catalina OS version

3.3.18 AIR 33.0.2.330

AIR-276: Updating skia used in x64 build to support fonts properly

AIR-283: Updating netstream handling to force a correct seek to start of a file (Gamua-79)

AIR-296: Fixing crash in Android audio mixer due to buffer size/data mismatch

AIR-298: Fixing crash in AIR relating to display objects in stopAllMovieClips

AIR-299: Fixing wrong stage resolution reported after splash screen has been displayed (Gamua-135)

AIR-301: Ensuring we don't have a Java exception if the app is closed during the splash screen (Gamua-157)

AIR-304: Updating IPA packaging to choose between universal 32- and 64- bit, vs 64-bit only on Catalina

AIR-307: Ensuring we cope with a license file in a read-only state

3.3.19 AIR 33.0.2.315

AIR-137: Crash in AIR runtime during requestPermission call

AIR-169: Android App Bundle support – see section 7 for details and limitations.

AIR-265: Crash with null function pointer when trying to pause audio stream

AIR-266: Ensuring output progress messages aren't blocked and async large file writing completes (Gamua-134)

AIR-267: Ensuring local URLs are correctly converted on Windows for Trusted Folder settings

AIR-268: Preventing URLLoader from reading bytes from a URLStream that has been closed already (Gamua-127)

AIR-274: ADT does not recognise new Apple certificates as being production ones (Gamua-137)

AIR-275: Reverting IPA generation to ensure we package both ARMv7 and ARMv8 versions (Gamua-142)

AIR-277: Italic textField cuts off by autoSize property (Gamua-78)

AIR-278: AIR support for Android x86_64 targets

AIR-284: Fixing crash in attachNetStream when the video plane is not on a view (Gamua-146)

3.3.20 AIR 33.0.2.288

AIR-135: Fixing crash in bitmap rendering following corrupt bitmap handling

AIR-250: Ensuring that ADT can still work with Java 7 runtime (as long as minSdkVersion < 26)

AIR-262: Crash in ARMv7 when rendering a bitmap, background thread calls null function pointer

3.3.21 AIR 33.0.2.281

AIR-173: Version/ABI information being output to the Android logcat upon start-up

AIR-199: Adding support for Java 8 features for Android extensions (Gamua-84)

AIR-205: Ensuring multidex support works for older Android devices (Gamua-102)

AIR-206: Fixing nanojit bug that was causing a crash with illegal opcode

AIR-211: New ”-license” option within ADT so that users can check their license status

AIR-221: Ensuring we use '/' notation for package/class to avoid crash-on-start-up problems on some devices (Gamua-1117)

AIR-231: Fixing crash in ADT if the license file has expired and improving the license check process

AIR-236: Improvements in stability within the JIT compiler for armv8

AIR-242: Updating iIPA tool to ensure packages can be installed onto iOS 13 devices [didn’t work]

AIR-246: Fixing StringOutOfBounds exception in getHardwareInfo

AIR-249: Fixing crash in Android audio loop creation due to race condition

AIR-251: Fixing crash in sha1_block_data_order by updating openssl for armv8

3.3.22 AIR 33.0.2.246

AIR-196: Generated license certificate files can be malformed

AIR-198: Add ability to control whether ADT prepends ”air.” to the Android Application ID

AIR-200: Analytics feature to provide information on platforms/tools used when packaging apps

AIR-201: Licensing feature to periodically confirm validity and update the license file

AIR-203: Drawing a video before Netstream starts to play causes a crash (Gamua-98)

AIR-204: Read “position” property of the async opened FileStream causes hang (Gamua-97)

3.3.23 AIR 33.0.1.228

AIR-190: AIR SDK scripts on MacOS don’t cope with the SDK path containing a space

AIR-192: Black screen when starting AIR (free tier) on older Android versions

3.3.24 AIR 33.0.1.220

AIR-112, Gamua-58: Update ADT so that it doesn’t compress certain file types (see ’UncompressedExtensions’ config file setting in section 5)

AIR-181: Android ‘back’ button cannot be handled in ActionScript (Gamua-73)

AIR-184: Camera is not working with ARMv8 binary (Gamua-72)

AIR-186: Camera hangs when video.attachCamera(null) is called in frame handler (Gamua-54)

3.3.25 AIR 33.0.0.212

HARMAN Ref AIR-159: Soft keyboard not appearing when an input text field has focus

HARMAN Ref AIR-160: Config file doesn’t take effect unless “DebugOut” setting is present

HARMAN Ref AIR-161: ADT packaging of ANEs doesn’t handle the use of a config file to override the architecture

3.3.26 AIR 33.0.0.182

ADOBE Ref AIR-4198749: AIR crashes on latest Anrdoid Q Preview

HARMAN Ref AIR-144: Performance hit on 64-bit ARM Android runtime

HARMAN Ref AIR-149: AIR SDK cannot package an app with google_play_services included in the manifest

HARMAN Ref AIR-153: Swf-Version built from Adobe Animate is set to 44 and does not work with ADL

HARMAN Ref AIR-156: ADT copyright output is affecting IDEA integration

HARMAN Ref AIR-157: Cannot export release build from FB on second attempt

HARMAN Ref AIR-158: AIR SDK package failures due to incorrect target SDK version

GAMUA Ref #55: Including Support-v4 28.0.0.ANE results in compile error

3.3.27 AIR 33.0.0.175

HARMAN Ref AIR-138: ADT shouldn’t compress raw/binary files when creating APK

HARMAN Ref AIR-139: ADT needs a mechanism to set the default target architecture

HARMAN Ref AIR-140: Building with new airglobal.swc file fails

HARMAN Ref AIR-142: air-sdk-description.xml isn't updated

HARMAN Ref AIR-143: ADT -version should only print the version and not the copyright notice

HARMAN Ref AIR-145: Crash in AIR runtime on ARMv7 builds

HARMAN Ref AIR-146: ADT should use "armv8" for consistency

HARMAN Ref AIR-151: ADT doesn't work with Java 8: dx tool failed

3.3.28 AIR 33.0.0.168

Adobe Ref AIR-4198789: 64-bit ARM support for Android.

Adobe Ref AIR-4198749: Text relocations on Android Q

HARMAN Ref AIR-82: System.Capabilities.supports64BitProcesses incorrect with 64-bit AIR builds

HARMAN Ref AIR-96: Remove reliance on deprecated "MODE_WORLD_READABLE" flag

4 Updating tools/IDEs to support 64-bit ARM

4.1 AIR Developer Tool

To package an android application with the armv8 binary, the "-arch armv8" option must be used on the ADT command line. By default, the packager will use armv7 unless a configuration file is provided – see below.

4.2 ADT Architecture Configuration

The default architecture used by ADT can be adjusted via the configuration file as described in section 5.

For example, to ensure that the packages created by ADT will always embed the 64-bit runtime, the configuration file should contain:

```
DefaultArch=armv8
OverrideArch=armv8
```

Using this configuration file, a developer can package their applications for ARMv8 targets using existing versions of Adobe Animate, FDT etc.

4.3 Flash Builder

The new AIR SDK should be updated using standard instructions found on Adobe's forums:

https://helpx.adobe.com/uk/flash-builder/kb/overlay-air-sdk-flash-builder.html

or for updating the Flex SDK: https://helpx.adobe.com/uk/x-productkb/multi/how-overlay-air-sdk-flex-sdk.html

If you find an issue with the AS3 not compiling, this can be addressed by https://forums.adobe.com/thread/1483159

Exporting a release build must be set to use the captive runtime.

To update the architecture, open the Project Properties and expand the ActionScript Build Packaging item to select "Google Android"

Click on "Customize Launch", "Add Parameter..." and give a name of "-arch" and value "armv8". Place this after the "-target" option.

Please note that AIR SDK now requires Java version 8, in line with Google's requirements for the latest Android build tools, and that Flash Builder's internal JRE needs to be updated accordingly: please see http://blogs.adobe.com/flashplayer/2018/02/running-adobe-flash-builder-on-mac-with-java-78.html

Also please note an issue which may cause problems when adding "-arch armv8" (or "-arch x86") to the launch parameters: https://forums.adobe.com/thread/1505072

4.4 Adobe Animate

To add support for the new AIR SDK, use the "Help | Manage Adobe AIR SDK…" option from Animate. Click on the "+" icon and select the folder into which you have extracted the SDK. This should show in the list of SDKs with the correct version number.

Animate 20.0 includes support for ARMv8 in the UI now; however, the configuration file mechanism can still be used to generate x86_64-based APKs or for users of older versions of Animate.

4.5 FlashDevelop

The packaging script asks the user which option to use for creating a mobile package (Android/iOS etc) but there is no way currently in this to specify an architecture (even for x86).

An extra section can be added to the "Packager.bat" script that will allow the user to be queried on the target ABI to be used in the package. The "Packager.bat" script can then be provided into FlashDevelop's project folder so that this is used for all new projects:

FlashDevelop\Projects\190 ActionScript 3 - AIR Mobile AS3 App\bat\Packager.bat

The extra choice needs to be added within the "android-config" section, prior to the "goto start" command:

```
:: which architecture?
echo.
echo Please select your target architecture for Android:
echo.
echo  [1] armv7       ^(32-bit ARM devices^)
echo  [2] x86         ^(Intel products^)
echo  [3] armv8       ^(64-bit ARM devices^)
echo.
set /P ARCH=[Choice:]
echo.
if "%ARCH%"=="1" set OPTIONS=%OPTIONS% -arch armv7
if "%ARCH%"=="2" set OPTIONS=%OPTIONS% -arch x86
if "%ARCH%"=="3" set OPTIONS=%OPTIONS% -arch armv8
```

4.6 Moonshine

Moonshine has a build.xml file which is used to call the ADT packaging tool:

```
<target name="compileAPKProject" depends="compileSWF">
      <java jar="${ADT_PATH}" fork="true" failonerror="true">
            <arg line="-package" />
            <arg line="-target apk-captive-runtime"/>
            ..
            <arg line="${SWF_FILE_PATH}" />
            <!-- Add folders to be bundled in the AIR file here -->
      </java>
</target>
```

An additional 'arg' can be added in order to select the ABI:

```
            <arg line="-arch armv8" />
```

4.7 IntelliJ IDEA

The new SDK should be incorporated into IntelliJ IDEA using the standard process documented at: https://www.jetbrains.com/help/idea/preparing-for-actionscript-flex-or-air-application-development.html

To build and package the application for the armv8 architecture, an option is being provided in the latest release of IDEA. This update to the “Package AIR Application Dialog” will now give the user the full set of target architecture options.

4.8 FDT

With FDT currently the same mechanism should be used as for Adobe Animate, with a configuration file being used to force a target architecture.

Please note that new applications created using FDT will pick up an incorrect namespace, and the application descriptor file needs to be manually changed back to 32.0.

4.9 Visual Studio Code

`asconfig.json` already supports android packaging options including the "arch" value. For targeting armv8, this needs to be updated:

```
"airOptions": {
    "android": {
        "arch": "armv8"
    }
}
```

See https://github.com/BowlerHatLLC/vscode-as3mxml/wiki/asconfig.json#android-options

5 Configuration File

ADT uses an optional configuration file to change some of its behaviour. To create a configuration file (there is not one by default within the SDK), create a new text file and save this with the name “adt.cfg” in the SDK’s “lib” folder (i.e. alongside the ‘adt.jar’ file). The configuration file is in the standard ‘ini file’ format with separate lines for each option, written as “setting=value”. Current options are listed below:

Setting	**Explanation**
DefaultArch	Used as a default architecture if there is no “-arch” parameter provided to ADT. Values may be ‘armv8’, ‘armv8’, ‘x86’ or ‘x64’.
OverrideArch	Used where an architecture value is being provided to ADT using the ‘-arch’ parameter, this configuration setting will override such parameter with the value given here. Values may be ‘armv8’, ‘armv8’, ‘x86’ or ‘x64’.
DebugOut	If set to “true”, results in additional output being generated into a local file which can aid in debugging problems within ADT (including the use of third party tools from the Android SDK). Values may be ‘true’ or ‘false’, default is ‘false’.
UncompressedExtensions	A comma-separated list of file extensions that should not be compressed when such files are found in the list of assets to be packaged into the APK file. For example: “UncompressedExtensions=jpg,wav”
AddAirToAppID	Configures whether or not the “air.” prefix is added to an application’s ID when it is packaged into the APK. Values may be ‘true’ or ‘false’, default is ‘true’.
JavaXmx	Adjusts the maximum heap size available to the Java processes used when packaging Android apps (dx/d8, and javac). Default value is 1024m although this is automatically overridden by any environment variable or value passed to the originating application. If this config setting is present, e.g. ‘2048m’, then it takes priority over all other mechanisms.
CreateAndroidAppBundle	Overrides any usage of ADT with an APK target type, and instead generates an Android App Bundle. Note that the output filename is not adjusted so this may result in generation of a file with “.apk” extension even though it contains an App Bundle. Values may be ‘true’ or ‘false’, default is ‘false’.
KeepAndroidStudioOutput	When generating an Android App Bundle, rather than using a temporary folder structure and cleaning this up, this option will generate the Android Studio file structure under the current folder and will leave this in place). Values may be ‘true’ or ‘false’, default is ‘false’.

AndroidPlatformSDK	A path to the Android SDK, that can be used instead of the “-platformsdk” command line parameter. Note that on Windows, the path should contain either double-backslashes (“c:\\folder”) or forwardslashes (“c:/folder”).
iOSPlatformSDK	A path to the iOS/iPhone/iPhoneSimulator SDK, that can be used instead of the “-platformsdk” command line parameter.
JAVA_HOME	This can be set as an override or alternative to the system environment variable that is read when ADT needs to use Java (e.g. when creating an Android App Bundle). Note that on Windows, the path should contain either double-backslashes (“c:\\folder”) or forwardslashes (“c:/folder”).

6 Android Applications – Play Store Uploads

New applications now need to have a 64-bit version of native code as well as a 32-bit version, as per the blog post from Google:

https://android-developers.googleblog.com/2019/01/get-your-apps-ready-for-64-bit.html

It will shortly be a requirement to use Android App Bundles to provide the necessary files for the Play Store to then generate an APK file specific to a user's device: please see section 7 for details. The below information is left in for reference but is less likely to be used now.

The guidelines and requirements for the use of multiple APK files can be found at:

https://developer.android.com/google/play/publishing/multiple-apks

Please note in particular the following requirement:

Each APK **must have a different version code**, specified by the `android:versionCode` attribute

Currently the ADT packaging tool will generate the `android:versionCode` attribute based on the version number provided in your AIR Application Descriptor File (which is generated by the likes of Adobe Animate from within the version given in the target settings, i.e. "AIR for Android Settings" dialog box). In the XML-based application descriptor, this is the "`versionNumber`" value.

The version is a dot-separated series of up to three numbers, for example "10.2" or "15.123.5". Internally this is translated into the `android:versionCode` value by splitting the numbers into millions, thousands, and units (if there are less than three parts to the version number, these are assumed to be zero, i.e. "10.2" is the equivalent of "10.2.0").

Hence "10.2" will become 10 million 2 thousand, 10002000; "15.123.5" will become 15 million 123 thousand and 5, 15123005.

To create a set of APKs that can be uploaded to the Play Store that will cover both 32-bit and 64-bit ARM devices, a developer would therefore need to create two APK files using two different version numbers. Due to the way in which the Play Store determines which APK to serve to which customer, the 64-bit version needs to be at the higher version level (because the 32-bit version could also run on a 64-bit OS, so if that had a higher version then it would completely overshadow the 64-bit APK).

The workflow should therefore be:

1) Create a first APK file for 32-bit ARM (armv7)
2) Update the version number by as small as increment as possible
3) Create a second APK file for 64-bit ARM (armv8)
4) Upload both APK files to the Play Store.

The same process can be applied to x86/x86_64 binaries, with the x86_64 version requiring a higher version code and both of these would need to be different from the versions used by the ARM-based APKs.

7 Android App Bundle

7.1 AAB Target

Google introduced a new format for packaging up the necessary files and resources for an application intended for uploading to the Play Store, called the Android App Bundle. Information on this can be found at https://developer.android.com/guide/app-bundle

AIR now supports the App Bundle by creating an Android Studio project folder structure and using Gradle to build this. It requires an Android SDK to be present and for the path to this to be passed in to ADT via the "-platformsdk" option. It also needs to have a JDK present and available via the JAVA_HOME environment variable.

To generate an Android App Bundle file, the ADT syntax is similar to the "apk" usage:

```
adt -package -target aab <signing options> output.aab <app descriptor and files> [-extdir <folder>]
-platformsdk <path_to_android_sdk>
```

No "-arch" option can be provided, as the tool will automatically include all of the architecture types. Signing options are optional for an App Bundle.

Note that the creation of an Android App Bundle involves a few steps and can take significantly longer than creating an APK file. We recommend that APK generation is still used during development and testing, and the AAB output can be used when packaging up an application for upload to the Play Store.

AAB files cannot be installed onto a handset: instead, an appropriate APK needs to be generated and installed, which can be done via the "bundletool" utility. Instructions for this are available at https://developer.android.com/studio/command-line/bundletool#deploy_with_bundletool, essentially the below lines can be used:

```
java -jar bundletool.jar build-apks --bundle output.aab --output output.apks --connected-device

java -jar bundletool.jar install-apks --apks=output.apks
```

The first step generates an "apks" file which is a collection of different APK files that are appropriate for the connected device; the second step then installs this onto a handset. Note that the APK generation here will use a default/debug keystore; additional command-line parameters can be used if the output APK needs to be signed with a particular certificate.

7.2 Play Asset Delivery

As part of an App Bundle, developers can create "asset packs" that are delivered to devices separately from the main application, via the Play Store. For information on these, please refer to the below link:

https://developer.android.com/guide/playcore/asset-delivery

In order to create asset packs, the application XML file needs to be modified within the <android> section, to list the asset packs and their delivery mechanism, and to tell ADT which of the files/folders being packaged should be put into which asset pack.

For example:

```
<assetPacks>
    <assetPack id="ImageAssetPack" delivery="on-demand" folder="AP_Images"/>
</assetPacks>
```

This instruction would mean that any file found in the ”AP_Images” folder would be redirected into an asset pack with a name ”ImageAssetPack”. The delivery mechanisms can be ”on-demand”, ”fast-follow” or ”install-time” per the Android specifications.

Note that assets should be placed directly into the asset pack folder as required, rather than adding an additional ”src/main/assets” folder structure that the Android documentation requires. This folder structure is created automatically by ADT during the creation of the Android App Bundle.

The asset pack folder needs to be provided as a normal part of the command line for the files that should be included in a package. So for example if the asset pack folder was ”AP_Images” and this was located in the root folder of your project, the command line would be:

```
adt -package -target aab MyBundle.aab application.xml MyApp.swf AP_Images [then other files, -platformsdk directive, etc]
```

If there were a number of asset packs and all of the relevant folders were found under an ”AssetPacks” folder in the root of the project, the command line woudl be:

```
adt -package -target aab MyBundle.aab application.xml MyApp.swf -C AssetsPacks . [then other files, -platformsdk directive, etc]
```

To access the asset packs via the Android Asset Pack Manager functionality, an ANE is being developed.

8 Windows builds

The SDK now includes support for Windows platforms, 32-bit and 64-bit. We recommend that developers use the “bundle” option to create an output folder that contains the target application. This needs to be packaged up using a third party installer mechanism, in order to provide something that can be easily distributed to and installed by end users. HARMAN are looking at adapting the previous AIR installer so that it would be possible for the AIR Developer Tool to perform this step, i.e. allowing developers to create installation MSI files for Windows apps in a single step.

Instructions for creating bundle packages are at:

https://help.adobe.com/en_US/air/build/WSfffb011ac560372f709e16db131e43659b9-8000.html

Note that 64-bit applications can be created using the “-arch x64” command-line option, to be added following the “-target bundle” option.

9 MacOS builds

MacOS builds are provided only as 64-bit versions. A limited shared runtime option is being prepared so that existing AIR applications can be used on Catalina, but the expectation for new/updated applications is to also use the “bundle” option to distribute the runtime along with the application, as per the above Windows section.

Note that Adobe’s AIR 32 SDK can be used on Catalina if the SDK is taken out of ‘quarantine’ status. For instructions please see an online guide such as:

https://www.soccertutor.com/tacticsmanager/Resolve-Adobe-AIR-Error-on-MacOS-Catalina.pdf

AIR SDK now supports MacOS Big Sur including on the new ARM-based M1 hardware: applications will be generated with ‘universal binaries’ and most of the SDK tools are now likewise built as universal apps.

10 iOS support

For deployment of AIR apps on iOS devices, the AIR Developer Tool will use the provided tools to extract the ActionScript Byte Code from the SWF files, and compile this into machine code that is then linked with the AIR runtime and embedded into the IPA file. The process of ahead-of-time compilation depends upon a utility that has to run with the same processor address size as the target architecture: hence to generate a 32-bit output file, it needs to run a 32-bit compilation process. This causes a problem on MacOS Catalina where 32-bit binaries will not run.

For this reason, it is not possible to create a universal binary (embedding both 32-bit and 64-bit ARM code) using MacOS Catalina (10.15) or later. The tools will only run the 64-bit version and the IPA files will be limited such that they only install onto 64-bit devices. To summarise the process/requirements:

- Using MacOS versions up to and including 10.14, an IPA will be created that contains a universal binary with both 32-bit and 64-bit code in it. This can be installed onto older iOS devices that do not support 64-bit, as well as newer devices.

- Using MacOS 10.15 upwards, with the restriction where we cannot run 32-bit executables, the IPA file will only contain the 64-bit code. These IPA files can only be installed onto 64-bit devices; there is an updated device requirements setting that is added automatically into the Info.plist file in this case.

- Using Windows, developers can continue to support 32-bit and 64-bit generation as before.

11 Splash Screens

For our ‘free tier’ users, a splash screen is injected into the start-up of the AIR process, displaying the HARMAN and AIR logos for around 2 seconds whilst the start-up continues in the background. There are different mechanisms used for this on different platforms, the current systems are described below.

11.1 Desktop (Windows/macOS)

Splash screens are displayed in a separate window centred on the main display, while the start-up continues behind these. The processing of ActionScript is delayed until after the splash screen has been removed.

11.2 Android

The splash screen is displayed during start-up and happens immediately the runtime library has been loaded. After a slight delay the initial SWF file is loaded in and when processing for this starts, the splash screen is removed.

11.3 iOS

The splash screen is implemented as a launch storyboard with the binary storyboard and related assets included in the SDK. This has implications for those who are providing their own storyboards or images in an Assets.car file:

- If you are on the ‘free tier’ then the AIR developer tool will ignore any launch storyboard you have specified within your application descriptor file, or provided within the file set for packaging into the IPA file.
- If you are creating an Assets.car file, then you need to add in the AIR splash images from the SDK which are in the “lib/aot/res” folder. These should be copied and pasted into your “.xcassets” folder in the Xcode project that you are using for creation of your assets.

Troubleshooting:

Message from ADT: “`Warning: free tier version of AIR SDK will use the HARMAN launch storyboard`” – this will be displayed if a <UILaunchStoryboardName> tag has been added via the AIR application descriptor file. The tag will be ignored and the Storyboard from the SDK will be used instead.

Message from ADT: “`Warning: removing user-included storyboard "[name]"`” will be displayed if there was a Storyboardc file that had been included in the list of files to package: this will be removed.

Message from ADT: "`Warning: free tier version of AIR SDK must use the HARMAN launch storyboard`" – this will be displayed if the Storyboardc file in the SDK has been replaced by a user-generated one.

If a white screen is shown during start-up: check that the HARMAN splash images are included in your assets.car file. Note that the runtime may shut down if it doesn’t detect the appropriate splash images.

The runtime may also shut down for customers with a commercial license if a storyboard has been specified within the AIR descriptor file but not added via the list of files to package into the IPA file.

12 ActionScript API Updates

Changes have been made to the AS3 APIs so the documentation hosted by Adobe is now out of date in the below cases. HARMAN will be taking over the hosting of the AS3 documentation at some point during 2021 at which point we will update the online documentation and remove this section.

12.1 Geometry Object Pooling

New APIs have been added to some of the geometry classes, following requests to support object pooling and reduce the memory wastage and overheads due to creation of new objects. To access these APIs, SWF version 44 is required. The parameter “output” can be used to pass in an object that should be operated on, rather than the runtime generating a new object for this. To keep some similarities with the previous equivalent functions, these then return the ‘output’ object from the function.

Class	API Updates
flash.geom.Matrix	`public function deltaTransformPointToOutput(point:Point, output:Point):Point`
	`public function transformPointToOutput(point:Point, output:Point):Point`
flash.geom.Matrix3D	`public function decomposeToOutput(orientationStyle:String = "eulerAngles", output:Vector.<Vector3D> = null):Vector.<Vector3D>`
	`public function transformVectorToOutput(v:Vector3D, output:Vector3D):Vector3D;`
	`public function deltaTransformVectorToOutput(v:Vector3D, output:Vector3D):Vector3D;`
	`public static function interpolateToOutput(thisMat:Matrix3D, toMat:Matrix3D, percent:Number, output:Matrix3D):Matrix3D;`
flash.geom.PerspectiveProjection	`public function toMatrix3DToOutput(output:Matrix3D):Matrix3D;`
flash.geom.Point	`public static function interpolateToOutput(pt1:Point, pt2:Point, f:Number, output:Point):Point`
	`public function subtractToOutput(v:Point, output:Point):Point`
	`public function addToOutput(v:Point, output:Point):Point`
	`public static function polarToOutput(len:Number, angle:Number, output:Point):Point`
flash.geom.Rectangle	`public function intersectionToOutput(toIntersect:Rectangle, output:Rectangle):Rectangle`
	`public function unionToOutput(toUnion:Rectangle, output:Rectangle):Rectangle`

flash.geom.Transform	public function copyConcatenatedMatrixToOutput(output:Matrix):void;
	public function copyConcatenatedColorTransformToOutput(output:ColorTransform):void;
	public function copyPixelBoundsToOutput(output:Rectangle):void;
	public function getRelativeMatrix3DToOutput(relativeTo:DisplayObject, output:Matrix3D):Matrix3D;
	public function copyPerspectiveProjectionToOutput(output:PerspectiveProjection):void;
flash.geom.Utils3D	public static function projectVectorToOutput(m:Matrix3D, v:Vector3D, output:Vector3D):Vector3D;
	public static function pointTowardsToOutput(percent:Number, mat:Matrix3D, pos:Vector3D, at:Vector3D=null, up:Vector3D=null, output:Matrix3D=null):Matrix3D;
flash.geom.Vector3D	public function crossProductToOutput(a:Vector3D, output:Vector3D):Vector3D
	public function addToOutput(a:Vector3D, output:Vector3D):Vector3D
	public function subtractToOutput(a:Vector3D, output:Vector3D):Vector3D

12.2 System class

In order to support the use of AIR on a command line, without using the Stage or display objects, additional functions have been added to the System class:

A further update is to add a ‘poisonStrings’ setting: any String object created whilst this setting is ‘true’ will then be overwritten by 0xDD bytes when the string is later garbage collected (even if the setting has since been switched back off). There is a minor memory/performance impact when using this, in that AIR will create new string objects even if you use “substring()” methods etc, while this value is set to ‘true’.

Class	**API Updates**
flash.system.System	public static function output(outString:String):void Writes the ‘outString’ value to the console, via stdout.
	public static function input(format:String):String Accepts up to 256 characters of input from stdin and returns a new string containing these characters. The ‘format’ parameter is currently ignored.
	Public static function set poisonStrings(value:Boolean):void Sets AIR into a mode where the memory used for new String objects will be poisoned upon clean-up.

12.3 PermissionManager class

In order to support MacOS privacy requirements, developers now have to request permissions on the Camera and Microphone classes, rather than retrieving a Camera or Microphone object and then calling “requestPermission” on that object. To support this, a new class is defined called “PermissionManager” within the flash.permissions package, and an object of this type is created on-demand when the Camera or Microphone class has its new “permissionManager” property inspected. With the PermissionManager, the application can then check for the status and request this via a standard pattern with an event listener being called once the permission status has been confirmed.

Class	API Updates
flash.media.Camera	public static function get permissionManager():flash.permissions.PermissionManager
flash.media.Microphone	public static function get permissionManager():flash.permissions.PermissionManager
flash.permissions.PermissionManager	Inherits from: flash.events.EventDispatcher Properties: • resourceType : String. Which resource this PermissionManager instance is responsible for. • permissionStatus : String. Determine whether the application has been granted the permission to use resource. See flash.permissions.PermissionStatus. Methods: • requestPermission() : void. Requests permission for the application to use the resource. See PermissionEvent. An event of type “PermissionEvent.PERMISSION_STATUS” will be dispatched when the permission has been granted or denied.

12.4 File class

A new property has been added to the File class, currently with an implementation just on Android devices. This provides the path to the application's external storage folder i.e. the folder that should be used for application storage if this must be on an external/removable drive, rather than Android's mapping of internal storage. This is provided as a static property that will be null if there is currently no removable storage device mounted.

For desktop builds, another new property is the "workingDirectory" value that provides the current working folder for the AIR/ADL executable.

Class	API Updates
flash.filesystem.File	`public static function get applicationRemovableStorageDirectory ():flash.filesystem.File`
	`Public static function get workingDirectory():flash.filesystem.File`