Adobe AIR SDK Release Notes

Version	50.2.1.1
Date	17 February 2023
Document ID	HCS19-000287
Owner	Andrew Frost

Table of contents

1 Release Overview

Release 50.2.1 of the AIR SDK is a namespace update i.e. new capabilities have been added into the application descriptor file, requiring a new minor version update.

1.1 Key changes

There are a number of updates included in this release, including the renewal of support for the Apple tvOS platform, and the ability to run the AIR ActionScript virtual machine in a background thread on Android devices to try to eliminate ANR errors.

A few updates have also been made to how ANEs can be configured for Android dependencies, and the APIs that are available for ByteArray manipulation.

For further information please see section 3.

1.2 Limitations

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`

Please note that there is no longer support for 32-bit IPA files, all IPAs will use just 64-bit binaries now so older iPhones/iPads may not be supported.

Android development should now be performed with an installation of Android Studio and the SDK and build tools, so that the new build mechanism (using Gradle and the Android Gradle Plug-in) can use the same set-up as Android Studio.

1.3 Feedback

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

The website for AIR SDK is available at: https://airsdk.harman.com with the developer portal available under https://airsdk.dev

1.4 Notes

Contributors to the https://airsdk.dev website would be very welcomed: this portal is being built up as the repository of knowledge for AIR and will be taking over from Adobe's developer websites. At some point the AS3 documentation will be migrated to this location and this can then be maintained directly by HARMAN (and/or the community) rather than having AS3 API updates listed within these release notes.

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.

On MacOS in particular, the use of the shared AIR runtime to 'install' a .air file will not create a signed application, hence new MacOS versions may block these from running. To ensure a properly signed MacOS application is created, the "bundle" option should be used with native code-signing options (i.e. those appearing after the "-target bundle" option) having a KeychainStore type with the alias being the full certificate name.

2 Release Information

2.1 Delivery Method

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

The update will also be available via the AIR SDK Manager, see https://github.com/airsdk/Adobe-Runtime-Support/discussions/2291

2.2 The Content of the Release

2.2.1 Detailed SW Content of the Release

Component Name	SDK 50.2.1.1
Core Tools	2.3.0
AIR Tools	2.0.1
Windows platform package	2.3.0
MacOS platform package	2.3.0
Linux platform package	2.3.0
Android platform package	2.3.0
iPhone platform package	2.3.0

2.2.2 Delivered Documentation

Title	Document Number	Version
Adobe AIR SDK Release Notes	HCS19-000287	50.2.1

2.2.3 Build Environment

Platform	**Build Details**	
Android	Target SDK Version:	31
	Minimum SDK Version:	14 (ARMv7, x86); 21 (ARMv8, x86_64)
	Platform Tools:	28.0.3
	Build Tools:	31.0.0
	SDK Platform:	Android-31
	Note – these are the versions we use to build the AIR SDK and runtime, we also recommend developers match the same 'target SDK' version as here.	
iOS	iPhoneOS SDK Version:	16.0
	iPhoneSimulator SDK Version:	16.0
	XCode Version:	14.0
	Minimum iOS Target:	11.0

tvOS	tvOS SDK Version:	16.0
	tvSimulator SDK Version:	16.0
	XCode Version:	14.0
	Minimum tvOS Target:	11.0
MacOS	MacOS SDK Version:	13.1
	XCode Version:	14.0
	Minimum macOS Target:	10.12
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 SDK over a previous version, there may be errors relating to “MainWindow.nib” and “MainWindow-iPad.nib”. These were originally files, and then had been turned into folders by a version of Xcode. However these should now be files again hence there may well be problems with overwriting of file types. If you see this error, the best approach is to delete these files/folders from the target location and then perform the copy/extraction again.

3 Summary of changes

3.1 Runtime and namespace version

Namespace: 50.**2**

SWF version: 50

The namespace and SWF version updates are made across all platforms and may be used to access the updated ActionScript APIs that have been introduced with AIR version 50.0. The namespace update is required for opening any SWF file that's got a SWF version of 50, or when using any of the new XML application descriptor flags.

3.2 Build Tools

The Android build tools and platform used to create the AIR runtime files has been updated to Android-31 with the default target SDK now set to this level in the generated Android manifest files.

Xcode 14.0.1 and the latest macOS and iphoneOS SDKs are now being used to build the AIR SDK.

The build system for this is on a version of macOS that doesn't support 32-bit processes hence we cannot generate the 32-bit versions of the stub files. This means that we can no longer support older 32-bit iPhone/iPad devices.

3.3 AS3 APIs

No changes

3.4 Features

Reference:	**AIR-4357**
Title:	`AIR Android – adding 'runtimeInBackgroundThread'flag to app descriptor`
Applies to:	Android runtime component
Description:	A new app descriptor setting has been introduced under the "`<android>`" tag, "`<runtimeInBackgroundThread>`", which can be set to "true" in order to switch the AIR runtime into a separate (non-UI) thread during application start-up. This means that there is an additional (internal) event queue added which should mean that the Android UI remains responsive to events, eliminating the "App Not Responding" problems particularly during start-up. Note that this does not actually improve performance, but shouldn't adversely affect it. There may need to be changes in any ANE that uses functions that must be run in the main/UI thread of the application.

Reference:	**AIR-6386**
Title:	`Adding handlers for Developer ID Application certs for .air on macOS`
Applies to:	macOS runtime components
Description:	AIR applications (packaged as .air files) can now be signed using a standard Apple Developer ID Application certificate. When the AIR runtime is installing such an application, it will validate the certificate via the OS and use this to declare to the end user that the publisher is verified.

Reference:	**AIR-6424**
Title:	`Adding support for TCP_NODELAY via a host name flag`
Applies to:	All runtime components
Description:	As an interim measure due to a customer request, it is now possible to append "?TCP_NODELAY" to an IPv4 socket address in order to switch the socket into a mode that avoids Nagle’s algorithm. i.e. smaller socket packets are sent with lower latency, but potentially wasting bandwidth. Default behaviour previously depended on the platform and setting.

Reference:	**AIR-6438**
Title:	`Adding support for content:// URIs on Android`
Applies to:	Android runtime components

Description:	To allow users to access files provided via the Storage Access Framework, support is now included for a File to be given a “content://” value as its native path (only on Android). This value will be resolved via the standard application content resolver, in order to open the file and provide normal file property and read/write access. As part of this, the File.browseXXX functions have been updated to use the Storage Access Framework. This means that attempting to browse for a file to open will request the file via the standard Android document open intent; attempting to save a file will use the document save intent, and support has been added for browsing for a directory/folder using the document tree intent. Permissions have changed slightly; the above File.browseXXX functions will always try to grant persistent permissions based on the user’s selection. If the application attempts to access a file for which it has no permission, it will trigger the standard permission error; if the “requestPermission()” method is then called on this File object, it will trigger the ‘open document’ intent which means that the user can grant read access to the file. This is not always useful though, since it is not possible to force the user only to choose the selected file; the runtime does a test to see whether the requested file can be opened following the user’s choice in order to return a ‘granted’ permission status. Calling “requestPermission()” on an empty file now will always immediately return a success status but this does not mean that the application has access to anything beyond its normal scoped storage capabilities.

Reference:	**Github-956** https://github.com/airsdk/Adobe-Runtime-Support/issues/956
Title:	`Adding gradle dependencies sections to Android ANEs`
Applies to:	Core build tools (Android ANEs)
Description:	Rather than using “packagedDependencies” and “packagedResources” sections of an Android platform extension descriptor, it is now possible to add a “gradle” section, within which “repositories” and “dependencies” can be added (as described in the above link for github-956). This allows ANE developers to maintain any dependencies on third party components without having to manually go through too update/re-pack each of these.

Reference:	**Github-2357** https://github.com/airsdk/Adobe-Runtime-Support/issues/2357
Title:	`Adding 'onRequestPermissionsResult' mechanism for Android ANEs to use`
Applies to:	Android runtime components

Description:	The AIR Android Activity Wrapper class has been updated so that ANEs can request permissions via the main activity, and receive a result in a callback. The mechanism is very similar to the ActivityResultCallback capability (which has also been updated so that the interface is public) – note that a non-zero permission request code must be used, zero is reserved for internal use.

Reference:	**Github-2369 https://github.com/airsdk/Adobe-Runtime-Support/issues/2369**
Title:	`Adding a package error if the macOS captive runtime is malformed`
Applies to:	Core build tools (packaging macOS applications)
Description:	Minor update to generate an appropriate packaging error if the macOS captive runtime or any framework files are invalid or malformed (e.g. files when symlinks are expected).

Reference:	**Github-2417 https://github.com/airsdk/Adobe-Runtime-Support/issues/2417**
Title:	`Building support for Apple tvOS applications`
Applies to:	iPhone runtime components
Description:	The “Apple tvOS” build has been re-created and updated to use the appletvos/appletvsimulator 16.0 SDK. Functionality and maturity should be equivalent to the last version that Adobe had created for this platform.

Reference:	**Github-2435 https://github.com/airsdk/Adobe-Runtime-Support/issues/2435**
Title:	`Adding configuration file 'iPhoneSimulator' entry`
Applies to:	Core build tools (iPhone development on macOS)
Description:	A new option has been added into the adt.cfg file so that the iPhone simulator name can be set up, overriding the default (which had previously been set to “iPhone 6” but has also been updated, to “iPhone 11”).

Reference:	**Github-2467 https://github.com/airsdk/Adobe-Runtime-Support/issues/2467**

Title:	`Adding Java FREByteArray setLength method`
Applies to:	Core build tools (Android ANE library)
Description:	A new method “setLength” has been added to the Java FREByteArray class, allowing the length of a byte array object to be set directly. It can also be set by passing a length to the static FREByteArray.newByteArray() method. Note that the C++ function, FRESetArrayLength, has also been updated as part of this so that it can take a ByteArray object as well as Array and Vector objects.

Reference:	**Github-2469** **https://github.com/airsdk/Adobe-Runtime-Support/issues/2469**
Title:	`Removing default Android INTERNET permission injection`
Applies to:	Android runtime component
Description:	By default, ADT had been adding in the “INTERNET” permission for Android applications. This was left over from earlier functionality and was no longer needed, so has been removed. Applications requiring access to the internet should list this in their Android permissions in the Application descriptor file.

3.5 Bug Fixes

3.5.1 Release 50.2.1.1

Reference:	**AIR-5846**
Title:	`Remove A2712Enabler from SDK/runtime`
Applies to:	macOS runtime component
Description:	Removing an unnecessary file from the Adobe AIR framework on macOS

Reference:	**Github-2208** **https://github.com/airsdk/Adobe-Runtime-Support/issues/2208**
Title:	`Allow activation of windows on Linux even if they aren’t owner/owned`
Applies to:	Linux runtime component

Description:	The ‘NativeActivation.activate()’ method was behaving differently on Linux and was not causing a window to be activated in certain situations.

Reference:	**Github-2339** https://github.com/airsdk/Adobe-Runtime-Support/issues/2339
Title:	`Ensuring Win32 Webview classes move properly between stages`
Applies to:	Windows runtime component
Description:	The Win32 Webview (both IE-based and Edge-based) were not moving properly between different NativeWindow instances when their ‘stage’ properties were being updated. This change fixes the behaviour so that it’s possible to move an instance of a StageWebView control from one stage to another.

Reference:	**Github-2372** https://github.com/airsdk/Adobe-Runtime-Support/issues/2372
Title:	`Adding a default string for NSLocationAlwaysAndWhenInUseUsageDescription in IPA info plist file`
Applies to:	iPhone runtime component
Description:	This new usage description had been required by Apple, so a default string has been added alongside the existing NSLocation…UsageDescription values. By default these will be generated in any IPA file.

Reference:	**Github-2375** https://github.com/airsdk/Adobe-Runtime-Support/issues/2375
Title:	`Fixing framework code resources in IPA signature`
Applies to:	iPhone runtime component
Description:	An error was found when signing a framework that included string bundles, where they were correctly added into the sha-1 signatures but not into the sha-256 signatures that Apple now use.

Reference:	**Github-2385** https://github.com/airsdk/Adobe-Runtime-Support/issues/2385
Title:	`Reverting FDB Worker workaround from github-399`

Applies to:	All runtime components
Description:	Changes from the fix for https://github.com/airsdk/Adobe-Runtime-Support/issues/399 had caused a problem in Workers when the debugger was active, causing ANE to then fail. The original issue was re-checked and a better fix has been implemented without this side-effect.

Reference:	**Github-2441 https://github.com/airsdk/Adobe-Runtime-Support/issues/2441**
Title:	`Ensuring IPA framework packaging handles universal binaries without armv7`
Applies to:	iPhone runtime components
Description:	The code to check whether a binary was “universal” (aka “fat”) or “thin” was just looking to see whether both arm64 and armv7 variants were present, so was mis-identifying a fat file that didn’t include armv7. This has been updated to check for the actual file type so that it can cope with all variants.

4 Android builds

4.1 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 (or set via a config file – it also checks in the default SDK download location). It also needs to have a JDK present and available, and will attempt to find this either from configuration files or via the JAVA_HOME environment variable (or if there is an Android Studio installation present in the default location, using the JDK provided by that).

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.

ADT allows an AAB file to be installed onto a handset using the "`-installApp`" command, which wraps up the necessary bundletool commands that generate an APKS file (that contains a set of APK files suitable for a particular device) and then installs it. If developers want to do this manually, 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
```

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.

4.2 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 would 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 available via the AIR Package Manager tool. See https://github.com/airsdk/ANE-PlayAssetDelivery/wiki

4.3 Android Text Rendering

Previously, the rendering of text on Android had been handled via a native library built into the C++-based AIR runtime file. This had some restrictions and issues with handling fonts, which caused major problems with Android 12 when the font fallback mechanism was changed and the native code no longer coped with this. To resovle this, a new text rendering mechanism has been implemented that uses public Android APIs in order to set up the fonts and to render the text.

The new mechanism uses JNI to communicate between the AIR runtime and the Android graphics classes for this, and has some differences with the legacy version. One of the changes that has been made is to correct the display of non-colorized text elements when rendering to bitmap data: in earlier builds, if some text included an emoji with a fixed color (e.g. ”flames” that are always yellow/orange even if you request a green font color) then these characters appeared blue, due to the different pixel formats used by Android vs the AIR BitmapData objects. With the new mechanism, AIR correctly renders these characters to BitmapData (although the problem still remains when rendering device text to a ’direct’ mode display list).

Some developers may not want to switch to this new mechanism yet, and others may want their applications to always use it. Some would perhaps want it only when absolutely necessary i.e. from Android 12 onwards. To cope with this request, there is a new application descriptor setting that can be used: ”<newFontRenderingFromAPI>” which shoudl be placed within the <android> section of the descriptor XML. The property of this can be used to set the API version on which the new rendering mechanism takes place. The default value is API level 31 which corresponds to Android 12.0 (see https://source.android.com/setup/start/build-numbers). So for example if you always want devices to use the new mechanism, you can add:

```
      <newFontRenderingFromAPI>0</newFontRenderingFromAPI>
```

whereas if you never want devices to use this, you could add:

```
      <newFontRenderingFromAPI>99999</newFontRenderingFromAPI>
```

5 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.

6 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.

7 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.

Additionally, due to the generation of stub files from the iPhone SDK that are used in the linking process – which are created in a similar, platform-specific way – it is not possible to create armv7-based stub files when using Catalina or later. From release 33.1.1.620, the stub files are based on iOS15 and are purely 64-bit. This means that no 32-bit IPAs can be generated, even when running on older macOS versions or on Windows.

8 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.

8.1 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.

8.2 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.

8.3 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.