



Adobe AIR SDK Release Notes

Version	50.2.4.5
Date	27 March 2024
Document ID	HCS19-000287
Owner	Andrew Frost

Table of contents

1	Release Overview.....	3
1.1	Key changes	3
1.2	Deployment.....	3
1.3	Limitations	4
1.4	Feedback	4
1.5	Notes.....	4
2	Release Information.....	5
2.1	Delivery Method	5
2.2	The Content of the Release	5
2.3	AIR for Linux – Restrictions.....	6
2.4	AIR for Flex users	6
3	Summary of changes.....	8
3.1	Runtime and namespace version.....	8
3.2	Build Tools	8
3.3	AS3 APIs.....	8
3.4	Features.....	8
3.5	Bug Fixes	9
4	Android builds.....	23
4.1	AAB Target.....	23
4.2	Play Asset Delivery	23
4.3	Android Text Rendering	24
4.4	Android File System Access.....	25
5	Windows builds.....	27
6	MacOS builds	28
7	iOS support.....	29
8	Splash Screens	30
8.1	Desktop (Windows/macOS)	30
8.2	Android.....	30
8.3	iOS	30

1 Release Overview

Release 50.2.4.1 of the AIR SDK is a feature update – a number of new features have been added and bugs have been fixed, although nothing that requires a change to the AIR namespace values or SWF version codes.

Release 50.2.4.2 then adds a number of further important bug fixes for this. For changes specific to 50.2.4.2 please see text in olive green font.

Release 50.2.4.3 is a bug fix update with changes focused around correct packaging of Android APK/AAB files when using Animate, as well as signing updates for IPA files. For changes specific to 50.2.4.3, please see text in orange font.

Release 50.2.4.4 further fixes bugs including some critical problems caused by recent changes, as well as improving the handling of the Android “content” URI schemes via the AS3 “File” class. For changes specific to 50.2.4.4, please see text in light blue font.

Release 50.2.4.5 includes a number of bug fixes that had been provided also within the 51.0 pre-release branch but were also considered useful to release into production at an earlier date, across various different platforms. For changes specific to 50.2.4.5, please see text in a purple font.

1.1 Key changes

The key updates are to build with the latest iPhoneOS, AppleTVOS and macOS SDKs and build tools, to expose the new APIs and so that applications using these platforms will be recognized by Apple as having used the appropriate tools and SDKs.

Note that there is still an issue with the linking of iPhone/iPad applications for publishing to the Apple App Store, where Apple are rejecting binaries that are linked using the LLVM linker that was incorporated into the previous releases. This new release reverts to using the “classic” Apple LD64 linker on macOS; on Windows, it still uses LLVM which should result in binaries that work but that will not be accepted by Apple. We are working on cross-compiling the Apple LD64 linker for Windows, now that they have released the recent updates in their open source build, which should then resolve this issue.

Some Android updates include fixes around black-screen issues, and the ability to use Stage3D when running in “gpu” render mode. There are also a number of minor changes and bug fixes. For details please see section 3.

For bug fixes in 50.2.4.2 please see section 3.5.2.

For bug fixes in 50.2.4.3, please see section 3.5.3.

For bug fixes in 50.2.4.4, please see section 3.5.4.

For bug fixes in 50.2.4.5, please see section 3.5.5.

1.2 Deployment

To obtain the release, it is recommended that developers install the AIR SDK Manager. Whilst the monolithic zip files will still be available from the <https://airsdk.harman.com> website, this may be updated less frequently in the future with only major releases. The goal is for the AIR SDK Manager to help us publish minor updates/fixes with a quicker cadence without resulting in a large amount of effort and data downloads.

The AIR SDK Manager is now available from the <https://airsdk.dev> website, as part of the “getting started” instructions, or directly from github at: <https://github.com/airsdk/airsdkmanager-releases>

1.3 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.

1.4 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>

1.5 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. The latest version of this can be downloaded from <https://github.com/airsdk/airsdkmanager-releases/releases>.

2.2 The Content of the Release

2.2.1 Detailed SW Content of the Release

Component Name	50.2.4.1	50.2.4.2	50.2.4.3	50.2.4.4	50.2.4.5
Core Tools	2.6.0	2.6.1	2.6.2	2.6.3	2.6.4
AIR Tools	2.0.2				
Windows platform package	2.6.0	2.6.1	2.6.2	2.6.3	2.6.4
MacOS platform package	2.6.0	2.6.1	2.6.2	2.6.3	2.6.4
Linux platform package	2.6.0	2.6.1	2.6.2	2.6.3	2.6.4
Android platform package	2.6.0	2.6.1	2.6.2	2.6.3	2.6.4
iPhone platform package	2.6.0	2.6.1	2.6.2	2.6.3	2.6.4

2.2.2 Delivered Documentation

Title	Document Number	Version
Adobe AIR SDK Release Notes	HCS19-000287	50.2.4

2.2.3 Build Environment

Platform	Build Details
Android	Target SDK Version: 33 Minimum SDK Version: 16 (ARMv7, x86); 21 (ARMv8, x86_64) Platform Tools: 28.0.3 Build Tools: 33.0.2 SDK Platform: Android-33 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: 17.4 iPhoneSimulator SDK Version: 17.4 XCode Version: 15.3 Minimum iOS Target: 12.0

tvOS	tvOS SDK Version: 17.4
	tvSimulator SDK Version: 17.2
	XCode Version: 15.3
	Minimum tvOS Target: 12.0
MacOS	MacOS SDK Version: 14.4
	XCode Version: 15.3
	Minimum macOS Target: 10.13
Windows	Visual Studio Version: 14.0.25431.01 Update 3
Linux	GCC Version 5.4.0 (Ubuntu 16.04.1)

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.

The original instructions from Adobe are at <https://helpx.adobe.com/uk/x-productkb/multi/how-overlay-air-sdk-flex-sdk.html> but a few alterations to this are needed to Step 4 if running on macOS. For this platform, the downloaded AIR SDK zip needs to be expanded to a temporary area and then the copy command needs to copy symbolic links as links rather than resolving them to files. This can be done using a capital ‘R’ rather than lowercase, hence:

```
cp -Rf /tmp/AIRSDK_Flex_MacOS/* /path-to-empty-FLEXSDK-directory
```

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.



Please note that the config files (air-config.xml, airmobile-config.xml, flex-config.xml) may need to be updated to support new features and updates in AIR or in dependencies such as ANEs. For example to ensure the correct SWF version is output, the below line would need to be updated (e.g. to '50' for AIR 50.x, or '44' for AIR 33.1, etc):

```
<swf-version>14</swf-version>
```

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-33 with the default target SDK now set to this level in the generated Android manifest files.

Xcode 15.3 and the latest macOS and iPhoneOS/tvOS SDKs are now being used to build the AIR SDK. Please note when the update was made to use Xcode 15.0, the minimum iOS/tvOS target version was increased to 12. **Additional note: these are the versions that AIR itself is built with. The versions shown in IPA files are manually injected by ADT and don't (yet) take the version codes from the local build environment. See [Issue #3030 \(github.com\)](#). The minimum macOS version had been specified as 10.15 in these release notes; in fact, the code contained different minimum versions for different components, and these have now been consolidated: the minimum macOS version is now set to 10.13 across the AIR framework.**

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-6707
Title:	Setting UDP broadcast settings for *.*.*.255 addresses
Applies to:	All runtime components
Description:	The earlier update to switch to a 'broadcast' mechanism for any address ending in .255 is now rolled out across all platform binaries.

Reference:	AIR-6809
Title:	Building on Sonoma/Xcode 15 for iPhoneOS/tvOS/macOS

Applies to:	iOS and macOS runtime components
Description:	Updates to the code, build settings and Info.plist configuration files in order to build against the latest SDKs and to create applications that show as having been packaged using the latest SDKs and tools.

Reference:	Github-2885 https://github.com/air sdk/Adobe-Runtime-Support/issues/2885
Title:	Picking up iOS/tvOS platform SDK version from platformsdk path
Applies to:	Core build tools
Description:	The “target SDK” value used in the linker for IPA files will now be picked up from the SDK version in the “-platformsdk” path, if present. For example, if a platform SDK argument is provided that ends with “iPhoneOS16.4.sdk” then the target SDK argument of 16.4 would be used. The default value is set within the ADT tool per these release notes (see section 2.2.3).

Reference:	Github-2911 https://github.com/air sdk/Adobe-Runtime-Support/issues/2911
Title:	Switching IPA linker on macOS to use ld-classic
Applies to:	Core build tools
Description:	To work around issues with the IPA linking process, on a macOS machine AIR will now use the Apple ‘classic’ ld64 binary for linking. (On machines without this binary, it will fall back to the normal ‘ld’ linker, and failing that will use the LLVM binary that’s provided within the AIR SDK).

3.5 Bug Fixes

3.5.1 Release 50.2.4.1

Reference:	Github-1194 https://github.com/air sdk/Adobe-Runtime-Support/issues/1194
Title:	Adjusting Android lifecycle handlers to avoid black screen in Home/Launcher scenario
Applies to:	Android runtime component

Description:	When using AIR applications as Android Home applications, the OS had been launching additional activities within the same process; this needed some updates to lifecycle handling in order to work properly.
--------------	--

Reference:	Github-2810 https://github.com/air sdk/Adobe-Runtime-Support/issues/2810
Title:	Ensuring AIR copes with UIBackgroundModes being a string as well as an array
Applies to:	iOS runtime component
Description:	The UIBackgroundModes value should be an array of strings; however, this update has been made to protect against a scenario where a single string was passed in. AIR now handles this as if it were an array of one string.

Reference:	Github-2869 https://github.com/air sdk/Adobe-Runtime-Support/issues/2869
Title:	Allowing Stage3D contexts to be created in Android gpu rendering mode
Applies to:	Android runtime component
Description:	In order to support GPU acceleration of normal displaylist rendering as well as the Stage3D APIs, a condition has been updated internally so that the OpenGL ES context used for “gpu” rendering mode on Android can also be used to kick off a Stage3D context. There may be some limitations in this mode (particularly a lack of VideoTexture).

Reference:	Github-2888 https://github.com/air sdk/Adobe-Runtime-Support/issues/2888
Title:	Moving Android planeKickCascade function into UI thread to avoid exception
Applies to:	Android runtime component
Description:	When running AIR in a background thread on Android, some issues were found within a “planeKickCascade” method that should not be called from a background thread. This function has been switched to run on the UI thread.

Reference:	Github-2893 https://github.com/air sdk/Adobe-Runtime-Support/issues/2893
Title:	Ensuring BitmapData.decode() works for transparency in PNGs

Applies to:	All runtime components
Description:	An error when setting up a BitmapData object internally had meant that PNGs with transparency were not correctly keeping the transparent colour when decoded using the new BitmapData.decode() method.

Reference:	Github-2923 https://github.com/airSDK/Adobe-Runtime-Support/issues/2923
Title:	Updating LLVM LD64.exe to remove MSVC runtime dependencies
Applies to:	iOS runtime components
Description:	The LLVM build of ld64.exe provided as part of the iOS toolchain on Windows had been built with an assumption that the appropriate Visual Studio redistributable libraries were available on the computer. This has been updated so that the runtime frameworks are built directly into the executable, which should help the compatibility for this component.

3.5.2 Release 50.2.4.2

Reference:	AIR-4740
Title:	Remove use of sun.security classes for code signing - replace with BouncyCastle
Applies to:	Core build tools
Description:	<p>For code-signing, ADT had been using internal sun security classes which had been removed from Java 17 onwards. The updates to Adobe Animate 2024 – where they now by default ship with and use the JRE 17 version – had then meant that macOS builds of IPA files were automatically using native codesign and Windows builds could not be signed.</p> <p>We have now gone through to remove the use of those older classes, and have updated to use the third party “BouncyCastle” library, in order to perform the same code-signing functionality</p>

Reference:	Github-1003 https://github.com/airSDK/Adobe-Runtime-Support/issues/1003
Title:	Ensuring Android surface restore copes with different event ordering
Applies to:	Android runtime component



Description:	When an Android app is restored from a background state, the different events for surface creation and activity resume had been coming in different orders, that could result in a 'black screen' scenario. An initial workaround that Adobe had targeted for specific devices has now been made more generic in order to cope with this.
--------------	---

Reference:	Github-1199 https://github.com/air sdk/Adobe-Runtime-Support/issues/1199
Title:	Adjusting VideoViewAIR to prevent UnsatisfiedLinkError when calling JNI method too early
Applies to:	Android runtime component
Description:	It could have been feasible for a video surface creation method to have called a JNI function prior to that function being registered, when using the AIR runtime in a background thread. This change attempts to prevent this, although it seems there may also be other causes of this error.

Reference:	Github-2409 https://github.com/air sdk/Adobe-Runtime-Support/issues/2409
Title:	Adding missing media symbols into AppleTV runtime libraries
Applies to:	iOS runtime component
Description:	An earlier change in the iPhoneOS builds had had a side-effect of AppleTVOS builds then having a missing symbol. The necessary files have now been included into the tvos runtime build libraries to avoid this.

Reference:	Github-2615 https://github.com/air sdk/Adobe-Runtime-Support/issues/2615
Title:	Updating Android stagetext/stagewebview functionality to cope better in background threads
Applies to:	Android runtime component
Description:	A number of issues with threading and asynchronous access of StageText and StageWebView objects had been causing problems, particularly during the destruction of these. Various changes have been made to try to work around these problems and ensure appropriate locked access from the different threads.



Reference:	Github-2642 https://github.com/air sdk/Adobe-Runtime-Support/issues/2642
Title:	Adjusting thread handling for iOS audio to avoid glitching
Applies to:	iOS runtime component
Description:	For a while we have had glitches in the audio on iPhone 14s and above, when adaptive brightness has been turned on and particularly when the device is in a bright environment. This had been tracked down to an iOS function that suddenly started to take longer to return. To work around this, the audio output mechanisms have been reworked with additional buffering and threading to avoid this problem.

Reference:	Github-2863 https://github.com/air sdk/Adobe-Runtime-Support/issues/2863
Title:	Ensuring ios webviews are inspectable for debug builds
Applies to:	iOS runtime component
Description:	A change in Apple's policies had broken the "inspectable" capability for web views, so this update restores the capability – but only for builds that have a "debug" capability already (ipa-debug etc).

Reference:	Github-2906 https://github.com/air sdk/Adobe-Runtime-Support/issues/2906
Title:	Allowing non-latin characters in files for Gradle-based builds (and using gradle for apk-signing)
Applies to:	Core build tools
Description:	<p>An internal restriction on resource naming had been preventing non-latin characters from being used in files, however that restriction was due to the use of AAPT in earlier build mechanisms. With Gradle builds, no such restriction is needed so this has been relaxed.</p> <p>Whilst making this change, it was also noticed that Gradle by default will sign an APK file using the v2 signature method, which meant that we could remove our additional code-signing step for these files.</p>

Reference:	Github-2924 https://github.com/air sdk/Adobe-Runtime-Support/issues/2924
Title:	Fixing Android VideoTexture flicker at start of video playback



Applies to:	Android runtime component
Description:	In direct mode, if there was a mixture of normal display content plus a Stage3D VideoTexture, then the texture was somehow displaying a background image prior to the first frame of the video being displayed. To avoid this, the start-up/display mechanism has been adjusted to ensure the video codec output is rendered prior to the first frame, since this seems to have an 'initialise to black' feature.

Reference:	Github-2950 https://github.com/air sdk/Adobe-Runtime-Support/issues/2950
Title:	Ensuring Android ANE .so libraries are packaged in AAB bundles
Applies to:	Core build tools
Description:	With a recent change to allow the use of the 'build configurations' flag to control what CPU versions are packaged in APK files, this had a side-effect that by default meant no .so libraries from Android ANEs were packaged into AAB files. This change fixes that so that by default, all four CPU variants are included, with the 'build configurations' then being used to fine-tune this as required.

3.5.3 Release 50.2.4.3

Reference:	AIR-6479
Title:	Thread protection around credential handling in macOS URL stream
Applies to:	MacOS runtime component
Description:	Instabilities were reported in macOS HTTPS handling, which happened within the handling of credentials for SSH authentication. A change has been made to remove the potential of a use-after-free scenario that may have been causing the crash here.

Reference:	AIR-6931
Title:	Ensure macOS apps installed via AIR shared runtime are always x64
Applies to:	AIR shared runtime for macOS

Description:	Apple's enhanced security checks mean that an ARM64-based binary, running on Apple Silicon, needs to be code-signing in order to run. But with applications that are distributed as .air files, it is not possible to sign these when they are installed. This change means that applications generated by the AIR shared runtime, when installing a .air file, will be installed as x64 based applications, which currently gets round the Apple security requirement.
--------------	---

Reference:	AIR-6964
Title:	AIR Android manifest additional features should override any from the template
Applies to:	Core build tools
Description:	If a 'uses-feature' manifest addition is provided in an application descriptor file, this should override any templated feature that is specified by AIR. Previously this would have just ended up duplicating the feature, resulting in an error when building.

Reference:	Github-2010 https://github.com/air sdk/Adobe-Runtime-Support/issues/2010
Title:	Re-implementing fix to allow Rectangle ..ToOutput methods to use 'this' as output
Applies to:	All runtime components
Description:	A fix had been implemented previously to ensure the new "...ToOutput" geometry methods could handle a scenario where an object is provided as the output for its own function call. That fix had been lost during an incorrect documentation update, so has been re-implemented here.

Reference:	Github-2625 https://github.com/air sdk/Adobe-Runtime-Support/issues/2625
Title:	Updating RTMPS code to accept self-signed certificates
Applies to:	All runtime components
Description:	The security handling for RTMPS had been rejecting remote servers that had used self-signed certificates. This change is an interim update to allow the connections by default: ultimately the goal would be to use the new 'certificateError' event to allow developers to have control over this behaviour.



Reference:	Github-2978 https://github.com/air sdk/Adobe-Runtime-Support/issues/2978
Title:	Ensuring APK files are code-signed with a release configuration
Applies to:	Core build tools
Description:	Release APK files had not been correctly signed following the fixes to handle code-signing via the Gradle tools rather than the (failing) internal code signature. This update ensures the provided certificate details are correctly configured within the Gradle settings for release builds.

Reference:	Github-2988 https://github.com/air sdk/Adobe-Runtime-Support/issues/2988
Title:	Ensuring URL schemes conforming to RFC3986 are accepted in AIR
Applies to:	All runtime components
Description:	When the “navigateToURL” method is called, AIR checks that the URL scheme used is valid. However, the check had been based on an internal Adobe specification which rejected schemes with a “.” character. This check has been updated to conform to the standards for scheme definitions.

Reference:	Github-2989 https://github.com/air sdk/Adobe-Runtime-Support/issues/2989
Title:	Correcting DER-encoding of macho entitlements for IPA code signature
Applies to:	Core build tools
Description:	The updated code-signing had been incorrectly wrapping an array of data from within a macho entitlement into an octet-string. This has been corrected so that the DER-encoding of the application entitlements now match the expectation from Apple’s security validation.

Reference:	Github-2992 https://github.com/air sdk/Adobe-Runtime-Support/issues/2992
Title:	Ensuring AABS built using Animate have all the native libraries included
Applies to:	Core build tools

Description:	When packaging from Animate, their use of the “armv7” command-line argument when combined with the override for switching from APK to AAB had resulted in a bundle without the correct native libraries. This update fixes the generation of bundles to ensure it includes all CPU formats, unless overridden by the application descriptor file preferences.
---------------------	---

Reference:	Github-2993 https://github.com/air sdk/Adobe-Runtime-Support/issues/2998
Title:	Adding thread-safety to new audio code to avoid segfault
Applies to:	iPhoneOS runtime components
Description:	The updated audio threads to fix glitching in iPhone audio had been calling a method to decode from MP3 data which could conflict with the removal of a sound channel from a different thread. The appropriate critical sections have been added to avoid this scenario to prevent the crashing.

3.5.4 Release 50.2.4.4

Reference:	AIR-6979
Title:	AIR JPEG decoding to take account of macOS display colour profile
Applies to:	MacOS runtime component
Description:	When a display on macOS was set to a colour profile with a different range than the standard sRGB palette, colours were being distorted resulting in high saturation levels. The change here corrects the JPEG decoding to follow the same mechanism as appears to be used by WebKit/Safari.

Reference:	Github-2972 https://github.com/air sdk/Adobe-Runtime-Support/issues/2972
Title:	Ensuring correct toplevels are used for user-specified ANEs to correct delayed-loading
Applies to:	All runtime components
Description:	When delayed-load is used for ANE libraries, the internal “toplevel” object had caused a mis-match, resulting in a failure to load in the ANE later on. With this change, the initial set-up uses the user-space toplevel rather than the internal one, which means manual loading later on is now working.



Reference:	Github-3000 https://github.com/air sdk/Adobe-Runtime-Support/issues/3000
Title:	Ensuring Android StageText elements can be re-added to stage
Applies to:	Android runtime components
Description:	When changing the code to cope with the asynchronous runtime, StageText objects had ended up with state handling within the “removed from stage” handler that could result in some later Java exceptions as well as preventing the StageText element from being re-added to the stage.

Reference:	Github-3034 https://github.com/air sdk/Adobe-Runtime-Support/issues/3034
Title:	Preventing iPhone webView crash in debug builds pre-ios16.4
Applies to:	iPhoneOS runtime components
Description:	The recent change to make a webView “inspectable” on iOS needed to have conditional execution flags around it, as the use of this property was causing a crash on iOS versions before 16.4. As well as this change, the setting should now take effect for ipa-test target types as well as ipa-debug.

Reference:	Github-3043 https://github.com/air sdk/Adobe-Runtime-Support/issues/3043
Title:	Updating handling of content URIs and file paths in Android
Applies to:	Android runtime components
Description:	<p>The handling of “content” URIs in Android has been causing problems for a while, and the updates here are to improve the handling and behaviour for File objects that are created when using the Storage Access Framework to get permissions for a folder, and then using the “resolvePath” mechanisms to create other File objects that are under this tree.</p> <p>The mechanism has been updated now so that primarily, an incoming URI will be checked to see if the OS recognises us as having permissions for it: if that is the case, then the URI is refactored so that it is based upon this permitted location. This then appears to ensure the ContentResolver class is able to handle the URI and knows that the permissions are granted.</p> <p>A fall-back to a normal ‘file’ based URI has also been added, but should only be used if the more normal Android mechanism has failed. This file path detection has limitations and variances depending on the device.</p>



Reference:	Github-3046 https://github.com/air sdk/Adobe-Runtime-Support/issues/3046
Title:	Fixing null reference error in ADT on Android manifest 'uses-feature' entry
Applies to:	Core build tools
Description:	The change to allow a user's feature requests to override those from the AIR template made the assumption that a value would always be provided for whether the feature is required or not. This resulted in a null reference error if the manifestAddition descriptor left the field blank to pick up the default value. This change copes with the lack of the 'required' attribute and uses the default 'true' if it's missing.

Reference:	Github-3058 https://github.com/air sdk/Adobe-Runtime-Support/issues/3058
Title:	Enabling native AAB signing with Java providerName argument
Applies to:	Core build tools
Description:	With some hardware-based code signing certificates, there will be a requirement to configure the Java security file so that the certificate vendor's settings can be found, in which case rather than a "keystore" parameter, a "providerName" parameter should be used for ADT. This should already have worked for signing .air files, but was failing for .aab files where a keystore value was expected. Note that further work is needed to ensure that signing via Gradle builds can handle such a scenario.

3.5.5 Release 50.2.4.5

Reference:	AIR-7028
Title:	AIR Android file permission callbacks not always called
Applies to:	Android runtime component
Description:	When adding an event listener to an Android File permission manager, the listener method was not always called back, due to a garbage collection issue within the runtime. This code has been updated to ensure that the reference counts are maintained properly.



Reference:	AIR-7029
Title:	AIR Android applicationDirectory files may not be accessible
Applies to:	Android runtime component
Description:	Some APK files appeared to contain a root path in the zip entry list which had caused the internal processing of files within the APK (i.e. that would appear under File.applicationDirectory) to abort early. As a result, these files were not then available to the application. This code has been made more robust.

Reference:	AIR-7035
Title:	String.fromCharCode() should support all unicode code points
Applies to:	All runtime components
Description:	The "fromCharCode" function had taken an integer value but was truncating this to a 16-bit value, and interpreting it as a single UCS16 character. Instead, the code now checks if the value is above that limit, and if so it will encode it into two UTF-16 characters.

Reference:	AIR-7059
Title:	Fixing AIR crash on iOS around network authentication
Applies to:	iOS runtime component
Description:	A problem with a race condition within some network authentication handling had caused a problem that was earlier found on macOS (see AIR-6479). It transpired that the same code was used in the source code for the iOS implementation, and had caused an instability, hence this update has been merged across into the iOS variant.

Reference:	Github-2610 https://github.com/air sdk/Adobe-Runtime-Support/issues/2610
Title:	Ensuring win32 timezone retrieval works for default tz when not dynamic
Applies to:	Windows runtime component



Description:	The default timezone was not being retrieved properly on Windows when run from some locations, if the “default” timezone for that location did not have a dynamic entry in the Windows timezone database. Some fallback code has been added to ensure that we also check for non-dynamic default timezone entries in case the initial call returned null.
--------------	---

Reference:	Github-2807 https://github.com/airSDK/Adobe-Runtime-Support/issues/2807
Title:	Removing ANRs caused by access of nativeGetTextBoundingBoxes from wrong thread
Applies to:	Android runtime component
Description:	When using the AIR runtime in a background thread, some touch handlers had caused problems when they tried to check whether the touch event was in a text box. This could result in some deadlocks and therefore ANRs. To work around this, we now assume (for AIR running in the background mode) that the touch event is not in a text field; this means that long-press handling within those text fields will not be available.

Reference:	Github-2903 https://github.com/airSDK/Adobe-Runtime-Support/issues/2903
Title:	Fixing instability when breaking into a debugger on uncaught error
Applies to:	All runtime components
Description:	Some logic within the debugger, where an uncaught error was being trapped by the debugger and then the debugger started querying the local variable stack, caused a null pointer dereference within the AIR runtime. The code here has been made more robust to ensure that these null values cause the debugger to continue without the instability.

Reference:	Github-3049 https://github.com/airSDK/Adobe-Runtime-Support/issues/3049
Title:	Eliminating instability in GC following socket thread querying the app descriptor
Applies to:	All runtime components
Description:	On some platforms and in some conditions, a query was being made within a background socket thread in order to check the API version of the runtime (i.e. based on the app descriptor namespace). However, there was an instability caused by the thread not having the appropriate garbage collection configuration and call stack in place.



Reference:	Github-3062 https://github.com/air sdk/Adobe-Runtime-Support/issues/3062
Title:	Updating win32 camera handling to include better fallbacks where direct connect fails
Applies to:	Windows runtime component
Description:	It was found that a virtual camera (for Canon) was not accepting the test connection that AIR was making in order to determine the available profiles/modes that the camera supported, resulting in a camera listed that you could not then connect to. An additional fallback option has been added to try a different connection mechanism should the existing attempts fail, which has allowed a connection to be made to this virtual camera.

Reference:	Github-3087 https://github.com/air sdk/Adobe-Runtime-Support/issues/3087
Title:	Correcting invalid scheme detection to prevent false-flagging of relative paths
Applies to:	All runtime components
Description:	A previous change had tried to ensure that URLs were correctly checked against the official format; however, this code was also executed for relative path names and this caused a problem for paths starting with numbers. The code has been corrected to treat these differently now; a URL scheme cannot start with a number, but a relative path can.

Reference:	Github-3098 https://github.com/air sdk/Adobe-Runtime-Support/issues/3098
Title:	Allowing a/v data access for NetStream in data generation mode
Applies to:	All runtime components
Description:	When NetStream is used with the “appendBytes()” mechanism, it was protecting the stream from being accessed by the user to try to extract a frame of video from this. However, this seems like an unnecessary restriction hence it is now possible to capture frames from videos when they are sourced from the “NetStream.appendBytes” method.

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 APK 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/air sdk/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 resolve 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-colored 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 should 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>
```


4.4 Android File System Access

In the earlier versions of Android, it was possible to use the filesystem in a similar way to a Linux computer, but with a set of restrictions that had a fairly high-level granularity:

- It was possible to read/write to an application's private storage location. AIR exposes this via `File.applicationStorageDirectory`.
- If the app requested the 'read/write storage' permission, the app could then read and write in the user's shared storage location and to removable storage. The main home folder was accessible via `File.userDirectory` or `File.documentsDirectory`, and later AIR 33.1 added `File.applicationRemovableStorageDirectory`.
- Later, this was updated such that the user had to also grant permission via a system pop-up message. To trigger this pop-up, AIR developers could use `File.requestPermission()`

With the introduction of "scoped storage" however, a lot of this has changed. Android files are treated in a similar way to other resources, with URLs using the "content://" schema which can refer either to filesystem-backed files, or to transient resources, or elements within other storage mechanisms such as databases and libraries. Permission to access each resource depends upon the creator of that resource, and by default it's not possible for an application to open a file that another application had created. Permissions for the top-level internal storage (i.e. `File.documentsDirectory`) have been changed so that applications cannot create entries here but must use sub-folders of these (a set of standard sub-folders is generally created by the OS).

Within AIR, we have been attempting to add support for the "content://" URLs, and to switch the File class "browseForXXX" functions so that they use the new intent-based mechanisms for selecting files to open and save, or to select a folder. Within these calls, we are also requesting the appropriate read/write permissions (and persisting these so that they can be used in the future). This means that it should be possible to call `browseForOpen()` and allow the user to select a shared file that can then always be opened (for reading). Equally a `browseForDirectory()` call should mean that an application then has read/write access into the selected directory and its sub-tree.

Requesting file system permissions has to be handled in a similar way, with permissions either granted for a file or for a folder tree. The `File.requestPermission()` function therefore looks at the native path of the File object this is called on, and decides whether to show a file open intent (if there's a normal path or URL in the `nativePath` property), or to show a folder selection intent (if the path ends in a forward-slash), or whether to just ignore the call with a 'granted' response and then wait for later permission requests for individual files (if the File object has not had a `nativePath` set). This last option is intended to allow apps to work across different Android versions and is the recommended option: early in the application lifecycle, create a new File and call `requestPermissions()`: if the app is running on an earlier Android version, the permission pop-up will appear, otherwise the app will need to request specific file access later on via the "browseForXXX" functions or by requesting permission for a specific file. Sadly it isn't possible to ensure that the user only gives a yes/no response for these file/folder open intents, they are able to browse for other files, so it may be that the file the user selects is not the one you are trying to open. If this is detected, the permission status event will show as 'denied', so if you are happy for the user to choose a different file, use `browseForOpen()` rather than `requestPermission()`.

There is an exception to having to use scoped storage and the storage access framework, which is if an application has the "MANAGE_EXTERNAL_FILES" permission. This permission is intended for utilities such as file manager apps and anti-virus scanners that have a legitimate need to access all the (shared storage) files on the device, but if an app requests this permission and is submitted to the Play Store, but doesn't justify itself, then the submission is likely to be rejected.

Some applications are not distributed via the Play Store though, at which point this permission can be used to turn the behaviour back to how it used to be in earlier Android versions. The



“`File.requestPermission()`” capability has been overridden in the cases where AIR detects this permission has been requested in the manifest, and it will now display the appropriate dialog to ask the user to turn on the ‘all files’ access for this app. Once this has been granted (asynchronously), it would then be possible to create, read and write files and folders including in the root storage device.



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.