Adobe AIR SDK Release Notes

Version	50.2.2.4
Date	13 April 2023
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 12

4 Android builds 21
4.1 AAB Target 21
4.2 Play Asset Delivery 21
4.3 Android Text Rendering 22
4.4 Android File System Access 23

5 Windows builds 25

6 MacOS builds 26

7 iOS support 27

8 Splash Screens 28
8.1 Desktop (Windows/macOS) 28
8.2 Android 28
8.3 iOS 28

1 Release Overview

Release 50.2.2.1 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 50.2.2.2 is a minor update to fix some issues, updates to this document are written in olive green font.

Release 50.2.2.3 fixes a few more issues, with some focus on the Android filesystem and permissions handling. Note that there are still some problems and restrictions around this, it's increasingly likely that new ActionScript APIs may be needed to reflect how the Android file access now works (or potentially these APIs may be provided through an ANE). A fuller description of the changes and approaches for file access on Android has been added into section 4.4. Updates relating to 50.2.2.3 are written in a dark orange font.

Release 50.2.2.4 is an Android-only update which fixes a major problem with 50.2.2.3 that was crashing on start-up on a lot of devices due to a JNI problem. There are other fixes still related to the Android file system access. Updates relating to 50.2.2.4 are written in a dark blue font.

1.1 Key changes

The initial support for tvOS added in 50.2.1 has been updated so should now work properly.

Apple SDKs have been updated so this build of the AIR SDK was created with macOS SDK 13.1, iPhoneOS/iPadOS SDKs 16.2, and tvOS SDK 16.1, all using Xcode 14.2.

"ScreenMode" support has been rolled out to all platforms albeit with some limitations in what may be possible on different devices; typically, this just allows access to more information about the current screen or display, without being able to update to a different mode.

Additional access has been provided from Android Java ANEs to allow these to render native (Android) video content to a Stage3D VideoTexture object. This should allow third party video players, such as Exoplayer, to render content directly into the Stage3D graphics pipeline.

A number of bug fixes have been provided as well as other minor updates. For further information please see section 3. Updates for 50.2.2.2 are in section 3.5.2. Updates for 50.2.2.3 are in section 3.5.3. Updates for 50.2.2.4 are in section 3.5.4.

1.2 Deployment

We are hoping to start deploying updates via the AIR SDK Manager now. 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:

https://airsdk.dev/docs/basics/install/macos

https://airsdk.dev/docs/basics/install/windows

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.

Please note also that AIR applications installed on the latest macOS versions on Apple silicon may be unable to run. It appears that Apple are applying different security/code-signing requirements on applications that are running natively as ARM64 based apps: if you encounter this issue, please try setting the application to launch using Rosetta. See below notes for more information.

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	SDK 50.2.2.1	SDK 50.2.2.2	SDK 50.2.2.3	SDK 50.2.2.4
Core Tools	2.4.0	2.4.1	2.4.2	2.4.3
AIR Tools	2.0.1	2.0.1	2.0.1	2.0.1
Windows platform package	2.4.0	2.4.0	2.4.1	2.4.1
MacOS platform package	2.4.0	2.4.0	2.4.1	2.4.1
Linux platform package	2.4.0	2.4.0	2.4.1	2.4.1
Android platform package	2.4.0	2.4.0	2.4.1	2.4.2
iPhone platform package	2.4.0	2.4.0	2.4.1	2.4.1

2.2.2 Delivered Documentation

Title	Document Number	Version
Adobe AIR SDK Release Notes	HCS19-000287	50.2.2

2.2.3 Build Environment

Platform	**Build Details**	
Android	Target SDK Version:	31
	Minimum SDK Version:	16 (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:	16.2
	iPhoneSimulator SDK Version:	16.2
	XCode Version:	14.2
	Minimum iOS Target:	11.0

tvOS	tvOS SDK Version:	16.1
	tvSimulator SDK Version:	16.1
	XCode Version:	14.2
	Minimum tvOS Target:	11.0
MacOS	MacOS SDK Version:	13.1
	XCode Version:	14.2
	Minimum macOS Target:	10.15
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.

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-31 with the default target SDK now set to this level in the generated Android manifest files.

Xcode 14.2 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 14.x, the minimum iOS/tvOS target version was increased to 11.

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-6395**
Title:	`Implementation of screen mode on mobile platforms`
Applies to:	All runtime component
Description:	The flash.display.ScreenMode class is now supported on all platforms, including iOS and Android. There are limitations on these in terms of the capabilities that they have, and the it may not be possible to set the mode (even where multiple modes are reported). The main intention is to enable the support of the "refreshRate" property so that applications can determine what the screen refresh rate is.

Reference:	**AIR-6451**
Title:	`ADT output logging to SDK Manager troubleshooting tab`

Applies to:	Core build tools
Description:	ADT has had logging capabilities for a while, that have to be enabled via a configuration file and then output some debug information to a file in the user's home directory. This new feature additionally outputs such information automatically to the AIR SDK Manager (version 0.3.0 onwards) when the user is viewing the "Troubleshooting" tab.

Reference:	**AIR-6475**
Title:	`AIR ANE - ability to use VideoTexture for Android video players`
Applies to:	Android runtime components

Description:	Two new APIs have been added to the definition of "FREContext":
	
	`public android.view.Surface getSurfaceFromVideoTexture(com.adobe.fre.FREObject);`
	
	This method takes an FREObject that must contain an AS3 "VideoTexture" object, from the flash.display3D.textures package. It returns an Android "Surface" object that can be set as the display output for a media feed – the Surface is created from a SurfaceTexture object, for more info see:
	
	SurfaceTexture | Android Developers
	
	`public void setVideoTextureDimensions(com.adobe.fre.FREObject, int, int);`
	
	This method must be used to set up the video dimensions, in order to start displaying video content. The FREObject must be the AS3 "VideoTexture" object used earlier; the two integer parameters are width and height.

Reference:	**Github-1777 https://github.com/airsdk/Adobe-Runtime-Support/issues/1777**
Title:	`Ensuring iOS cameras support higher resolutions`
Applies to:	iOS runtime components
Description:	The support for iOS camera resolution had been limited to the 1280x720 preset value. This has been updated to support 1920x1080 and 3840x2160 resolutions.

Reference:	**Github-1802** https://github.com/airsdk/Adobe-Runtime-Support/issues/1802
Title:	`Updating camera maximum resolution to 4K`
Applies to:	All runtime components
Description:	The camera resolutions had been limited internally to 1920x1080, this has been updated to support up to 3840x2160 resolutions.

Reference:	**Github-1984** https://github.com/airsdk/Adobe-Runtime-Support/issues/1984
Title:	`Enabling video on linux using ffmpeg`
Applies to:	Linux runtime components
Description:	In order to support H.264 video display on Linux, a decoder adapter has been added that uses FFMPEG (which must be available/installed on the target computer with the appropriate codecs/support for H.264).

Reference:	**Github-2073** https://github.com/airsdk/Adobe-Runtime-Support/discussions/2073
Title:	`Adding support for HTTPS_PROXY environment variable to ADT`
Applies to:	Core build tools
Description:	To ensure that ADT can access the internet (for license checking and usage reporting) it will now read an HTTPS_PROXY environment variable in order to detect this and access the network via a proxy server.

Reference:	**Github-2482** https://github.com/airsdk/Adobe-Runtime-Support/discussions/2482
Title:	`ADT certificate creation supporting 25 year default and beyond 2050`
Applies to:	Core build tools

Description:	The default duration for a self-signed certificate created using ADT had been set to 5 years. It was felt that 25 years would be a better duration, in line with Google's expectations for the Play Store. ADT was not able to generate certificates that had an expiry date beyond 2049; this restriction has been lifted and the required format changes have been made so that certificates cope with all dates.

Reference:	**Github-2502** https://github.com/airsdk/Adobe-Runtime-Support/discussions/2502
Title:	`Adding file version information into .exe file generated by windows bundle packaging`
Applies to:	Windows runtime components
Description:	When creating a Windows bundle, the .exe file created for the new application had missed a lot of the expected file property information. These values are now being set by the runtime during the bundle package creation, using values from the application descriptor file.

Reference:	**Github-2522** https://github.com/airsdk/Adobe-Runtime-Support/discussions/2522
Title:	`Throw an error if 'new Vector()' is called with an invalid argument type`
Applies to:	All runtime components
Description:	Vector constructors are not particularly clear, with the ability to create a vector of a set length via "new Vector()" but the ability to create a vector from an existing array via "Vector()" as a function. If the "new" keyword is used i.e. the Vector constructor, and the first argument (if present) is not a numeric type, this will now throw an argument error so that it's clear a programming error has been made.

Reference:	**Github-2525** https://github.com/airsdk/Adobe-Runtime-Support/discussions/2525
Title:	`Automatically injecting INTERNET permission for debug Android packages`
Applies to:	Core build tools

Description:	Previously, ADT had been adding in the “INTERNET” permission for Android applications, but this was taken out in a previous release. However, this capability is required for debugging with FDB or for access via Adobe Scout. The permission is now being automatically injected again for any “-debug” target type for Android applications.

3.5 Bug Fixes

3.5.1 Release 50.2.2.1

Reference:	**AIR-6037**
Title:	`Updating iOS event handling to fix screen time impact on audio`
Applies to:	iOS runtime component
Description:	If an app had been restricted by iOS “Screen Time” settings, and was then launched with the “ignore limit” option, the audio playback initially failed and then became chopped. This was due to some incorrect handling of notification events from iOS, and has been updated to improve the behaviour around screen time updates. Note that there are still outstanding issues with choppy sound e.g. when playing back from within a background mode or on some new phones when the display auto-adjustment is happening if there is a lot of ambient light.

Reference:	**AIR-6479**
Title:	`Fixing instabilities in macOS URL handling`
Applies to:	macOS runtime component
Description:	Some problems had been reported around the handling of URL streams where a race condition could cause an instability during the shutdown of an http fetch operation. The code has been refactored to eliminate the access of data that may have been cleaned up from a different thread.

Reference:	**AIR-6486**
Title:	`Fixing crash in Android permission manager from CameraRoll storage request`
Applies to:	Android runtime component

Description:	The recent change in the Android AIR runtime to accommodate 'content://' URLs and permission requests via the Storage Access Framework had an unintended side-effect when permission requests were raised from the camera roll AS3 object. The code has been refactored to eliminate the risk of instabilities in this code.

Reference:	**Github-1830** https://github.com/airsdk/Adobe-Runtime-Support/discussions/1830
Title:	`Updating macho signing to allow replacing of a smaller signature at the end of a file`
Applies to:	Core build tools
Description:	Some iOS frameworks could not be re-signed by the ADT tool (typically only affecting Windows-based packaging) because an existing signature was present and a new signature would not fit into the space available. Where this situation occurs and when the code signature is the last element in the file (which should be most cases!) the tool now just extends the file size in order to fit in the new signature.

Reference:	**Github-2293** https://github.com/airsdk/Adobe-Runtime-Support/issues/2293
Title:	`Add caching of method closures to reduce memory churn`
Applies to:	All runtime component
Description:	When global functions are called (e.g. `flash.utils.getTimer()`), a method closure is created for this. The ActionScript virtual machine was holding weak references to the closure as it would be needed every time the function was called – but the memory management code was immediately cleaning these up, resulting in a large churn of a closure being created for every call. A limited cache has been added to ensure that methods called frequently will be kept alive, reducing the memory overheads.

Reference:	**Github-2339** https://github.com/airsdk/Adobe-Runtime-Support/issues/2339
Title:	`Fixing StageWebView location handling across stages and screens`
Applies to:	Windows runtime components

Description:	The positioning of StageWebView objects on Windows (using both the IE-based WebView and the Edge-based WebView2 components) had had problems when setting viewPort values and when moving items between different windows/stages. There are a number of updates that intend to fix most of the cases, although it's possible that some problems still remain. Note that viewPort information is currently interpreted using window coordinates, rather than stage coordinates (where the stage size doesn't match the window size i.e. if the scaleMode is not "noScale").

Reference:	**Github-2385** https://github.com/airsdk/Adobe-Runtime-Support/issues/2385
Title:	`Ensuring Worker isolates load ANE swf definitions at start-up`
Applies to:	All runtime components
Description:	When a Worker is started up, it does not run through the application bootstrap code in which native extensions are identified and loaded. This meant that any ANE code (from the library.swf files) was not available to a secondary Worker object. A change has been made to go through the loaded ANEs from the primary Worker and to re-load the SWF definitions into the secondary Worker.

Reference:	**Github-2409** https://github.com/airsdk/Adobe-Runtime-Support/discussions/2409
Title:	`Updating build settings for tvOS runtime to avoid missing symbols`
Applies to:	iOS runtime components
Description:	tvOS applications would not build/run with the previous AIR SDK release, several changes have been made to ensure that the appropriate symbols are provided and the necessary linking and runtime support has been updated. Note that tvOS builds are only available for commercially-licensed developers.

Reference:	**Github-2481** https://github.com/airsdk/Adobe-Runtime-Support/issues/2481
Title:	`Fixing performance slowdown when Direct3D window is minimised`
Applies to:	Windows runtime components

Description:	With a mutli-windowed application, if the primary Stage3D window was minimised, there was a performance impact on all secondary windows. This was a result of how the Direct3D integration had been performed and the behaviour of the Direct3D “present” call on a minimised window; this has been eliminated to ensure there is no impact on performance if the main window is minimised.

Reference:	**Github-2493** https://github.com/airsdk/Adobe-Runtime-Support/issues/2493
Title:	`Fixing crash on Linux when exiting fullscreen`
Applies to:	Linux runtime components
Description:	When a sequence of events was triggered including full screen and maximising of windows on Linux, it was possible for a race condition to result in an application crash. A fix has been made to eliminate the multiple calls that had caused the instability.

Reference:	**Github-2496** https://github.com/airsdk/Adobe-Runtime-Support/discussions/2496
Title:	`Ensuring any folder structures are created for mac bundle outputs`
Applies to:	macOS runtime components
Description:	When a macOS application was packaged, if the output bundle was requested to be created in a subfolder, there would be an error if the subfolder did not exist. This change adds the creation of the full file path where necessary so that the output bundle can be copied to this location.

Reference:	**Github-2508** https://github.com/airsdk/Adobe-Runtime-Support/issues/2508
Title:	`Ensuring that dpi-changed resize events are handled during moveWindow`
Applies to:	Windows runtime components
Description:	When a “moveWindow()” method is called to start the moving of a window based on mouse events, the window was not properly resized if it was moved between monitors that had different display scale values. The necessary code handling had been disabled during a moveWindow operation; this has been re-enabled for the situation where the resize request is due to DPI changes.

3.5.2 Release 50.2.2.2

Reference:	**AIR-6494: https://github.com/airsdk/Adobe-Runtime-Support/issues/2531**
Title:	`IPA files need to have correct Xcode/platform tool version codes`
Applies to:	Core build tools
Description:	The AIR runtimes had been updated for iPhoneOS SDK 16.2 and AppleTVOS SDK 16.1, but this hadn't been reflected properly in the IPA files generated by ADT. This update ensures the correct values are injected so that the App Store recognises these as being built appropriately.

Reference:	**Github-2529: https://github.com/airsdk/Adobe-Runtime-Support/issues/2529**
Title:	`Fixing code-signing of frameworks - omitting pkginfo from files2 section`
Applies to:	Core build tools
Description:	Another issue in the code-signing for IPA files when run on Windows: this was a problem where the "PkgInfo" file signature was being provided in both SHA-1 and SHA-256 sections of the signature, whereas it should only be provided in the SHA-1 section.

3.5.3 Release 50.2.2.3

Reference:	**AIR-6525**
Title:	`Generating stub files for swift libraries`
Applies to:	iOS runtime components
Description:	iPhoneOS SDK 16 introduce an additional set of Swift libraries for interoperability and compatibility. The .dylib files for these are now being generated and included in the AIR SDK, as well as a compatibility library that can be linked in for ANEs that require it (libSwiftCompat.a).

Reference:	**AIR-6526**
Title:	`Ensuring ADT does not generate bitcode (for tvOS)`

Applies to:	Core build tools
Description:	Bitcode is no longer required for App Store submissions, and they are starting to reject applications that include it. ADT can inject this on demand but was also defaulting to including bitcode for tvOS applications built for the App Store. This default handling has been switched so that, by default, ADT will never generate bitcode for IPA files.

Reference:	**Github-1984:** https://github.com/airsdk/Adobe-Runtime-Support/issues/1984
Title:	`Removing Linux H.264 video support whilst issues are resolved`
Applies to:	Linux runtime components
Description:	The initial attempt to provide H.264 video decoding capabilities via the use of platform ffmpeg libraries has thrown up issues with compatibility, and how a build of software is able to work against multiple versions of ffmpeg that are binary incompatible with each other. This raised some fundamental issues that need to be resolved, so for now the functionality has been disabled again whilst the various options and solutions are considered.

Reference:	**Github-2326:** https://github.com/airsdk/Adobe-Runtime-Support/issues/2326
Title:	`Workaround for Android ANE functions to run in UI thread`
Applies to:	Android runtime components
Description:	The recent update to allow Android applications to have the runtime in a background thread then caused a problem for some ANE files where Java function calls are being made that must be made from the UI thread. Whilst ANEs should ultimately be updated to handle the multi-threaded possibilities here, a change has been made to allow an application to select some ANE functions that will then be run on the main thread rather than on the runtime thread. To configure this, the fully qualified function class name is required (which can generally be seen in the call stack for a “run on UI thread” failure – the class name will have “.call” after it which is the FREFunction override that is called by AIR). A file (empty/tiny) needs to be created and packaged in the base of the application (so, generally, alongside the main SWF file) that has the exact same name as the class name. During function registration, the runtime checks for such files and if found, the functions will be called via a separate mechanism to run in the UI thread whilst the runtime waits for the result.

Reference:	**Github-2409:** https://github.com/airsdk/Adobe-Runtime-Support/issues/2409

Title:	Removing tvOS invalid reference
Applies to:	iOS runtime components
Description:	AppleTV applications were crashing at start-up with an undefined symbol. This symbol should be present in AppleTV OS according to the documentation but its use has been removed, at least temporarily, to try to work around this runtime failure.

Reference:	**Github-2486:** https://github.com/airsdk/Adobe-Runtime-Support/issues/2486
Title:	Ensuring only one maximize event is sent on macOS
Applies to:	MacOS runtime components
Description:	When a window was maximised on macOS, there were sometimes two events being sent – an additional, initial event was sent due to the reconfiguration of the window into the larger area, but then some core code was also triggering a repeat of the event. This has been adjusted so that during one operation, only one event can be sent.

Reference:	**Github-2517:** https://github.com/airsdk/Adobe-Runtime-Support/issues/2517
Title:	Check for intent handling before requesting SAF file permissions
Applies to:	Android runtime components
Description:	One issue was reported where the Android system reported that it could not find an intent for a file permission request (on Chromecast platforms?) – this situation was not being handled properly resulting in a lack of response back to the AS3 code, so there is now a check being made for the availability of the intent handler, with a fallback to the earlier permission behaviour in case no intent can be found.

Reference:	**Github-2532:** https://github.com/airsdk/Adobe-Runtime-Support/issues/2532
Title:	Fixing argumenterror thrown when closing a window after StageWebView.dispose() call
Applies to:	All runtime components

Description:	One of the recent changes to ensuring StageWebView objects could be moved between windows meant that a listener was added into the StageWebView object itself, checking for when the window on which it was placed was being closed down. However, the checks in here were then throwing an exception if the 'dispose' method had been called; to prevent this, the event listeners are now removed manually when the object is disposed rather than waiting for garbage collection.

Reference:	**Github-2533:** https://github.com/airsdk/Adobe-Runtime-Support/issues/2533
Title:	`Attempting to resolve Android content files and launch in default app; Updating Android file handling functions to use Java to fix permission issues; Correcting File.resolvePath() for Android content URLs`
Applies to:	Android runtime components
Description:	A number of changes have been made to how the Android code handles file requests using 'content://' URIs. One of the recent changes had called a Java API from within C++ but without the normal version checking which caused problems for apps running on old Android versions (KitKat). There were also problems with how "`File.resolvePath()`" worked when using "content://" URIs, plus other fixes have been made. There may still be some issues to be corrected here, but it is also worth noting that newer Android versions will not allow the same level of file system access that had been possible in the past, so applications should use the filesystem within the expected approach that Android are promoting. For more information see section 4.4.

Reference:	**Github-2547:** https://github.com/airsdk/Adobe-Runtime-Support/issues/2547
Title:	`Ensure StateChange Events are sent for maximise events on Linux even if the window has already been resized.`
Applies to:	Linux runtime components
Description:	There were some conditions when a window was being maximised on Linux where the AS3 StateChange event was not being dispatched. This has been updated to ensure more consistency between the activity and the event handling across the platforms.

3.5.4 Release 50.2.2.4

Reference:	**AIR-6568**
Title:	`Fix permission handling for MANAGE_EXTERNAL_STORAGE`

Applies to:	Android runtime components
Description:	The ‘MANAGE_EXTERNAL_STORAGE’ permission is one that is unlikely to be granted by a Play Store review, but for apps that are distributed outside of this, it could be a very useful option. However, the appropriate intent was not being created for this, which meant that the permission request was not properly shown to the user. With this change, if a ‘File.requestPermission()’ call is made and this permission is listed in the manifest, then the app file access permission screen is shown to the user and the user can determine whether or not this full access has been granted.

Reference:	**Github-2559: https://github.com/airsdk/Adobe-Runtime-Support/issues/2559**
Title:	`Fixing issues with content URIs and permissions on Android`
Applies to:	Android runtime components
Description:	The earlier fixes still had problems depending on the approach used for requestion permissions, and the format of the resulting content URI (some URIs were ‘tree’ ones, some were ‘document’ ones, both under the ‘content’ schema). These should (mostly) be handled properly now with the appropriate conversions happening within the runtime so that the permissions that had been granted by the user are correctly applied to the access request for a file from ActionScript.

Reference:	**Github-2568: https://github.com/airsdk/Adobe-Runtime-Support/issues/2568**
Title:	`Ensuring Android JNI file util classes can be loaded`
Applies to:	Android runtime components
Description:	The new JNI classes that had been referenced from the Android AIR runtime were not correctly loaded in from the C++ code, which meant that applications would crash when starting up on a lot of devices. The referencing has been fixed and applications should now be stable again.

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>
```

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://" URIs, 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.