Adobe AIR SDK Release Notes

Version	**51.2.1.3**
Date	1 May 2025
Document ID	HCS19-000287
Owner	Andrew Frost

Table of contents

1   Release Overview

Release 51.2.1.1 is the first production-ready release in the AIR 51.2 branch. The earlier 51.2.0 betas have been rolled up into this along with the latest 51.1 release, and a further set of bug fixes and updates.

Release 51.2.1.2 is a quick patch release that contains a few fixes for critical errors found in the 51.2.1.1 version. There are only limited changes targeting those specific issues, updates are displayed in a dark red font.

Release 51.2.1.3 includes further critical fixes plus a few updates that allow earlier fixes/features to be configured further. Updates related to 51.2.1.3 are displayed in a light blue font.

This release can be used for application distribution, however two of the newer features are still not yet mature and should not be used unless an application has been carefully tested and verified for the target platforms: these are the ANGLE support on Windows, and the H.264/AAC multimedia capabilities on Linux. Whilst these features may work for some scenarios, there are a significant number of reports of problems still. We will be working to improve the stability of these areas over the next few releases.

1.1   Key changes

Bug fixes in 51.2.1.2 can be found in section 3.5.2.

Bug fixes in 51.2.1.3 can be found in section 3.5.3.

1.2   Deployment

To obtain the release, developers will need to install the AIR SDK Manager, 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

Linux support for the AIR SDK Manager has now been added, so this will be the primary mechanism used for deployment of the AIR SDK now.

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.

Linux runtimes are built using Ubuntu 16 for x86_64 variants in order to provide maximum compatibility; however for arm64, the build environment uses Ubuntu 22 which then restricts usage to similar versions of Linux (i.e. that have glibc version 2.34 or later).

Note that ANGLE support on Windows, and H.264/AAC support on Linux using FFMEG, are both features that are currently causing significant issues and instabilities, and should only be used if a particular app has been tested sufficiently on all the target platforms.

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

The AS3 documentation for AIR is updated and now also available under this site: https://airsdk.dev/reference/actionscript/3.0/

We will continue to provide the shared AIR runtime for Windows/macOS; however, this is not a recommended deployment mechanism, it is prefereably to create native installers based on the "bundle" deployments.

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

The 51.2 releases will only 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	51.2.1.1	51.2.1.2	51.2.1.3
Core Tools	3.5.2	3.5.3	3.5.4
AIR Tools	3.1.0		3.1.1
Windows platform package	3.5.2	3.5.3	3.5.4
MacOS platform package	3.5.2	3.5.3	3.5.4
Linux platform package	3.5.2	3.5.3	3.5.4
Android platform package	3.5.2	3.5.3	3.5.4
iPhone platform package	3.5.2	3.5.3	3.5.4

2.2.2   Delivered Documentation

Title	Document Number	Version
Adobe AIR SDK Release Notes	HCS19-000287	51.2.1

2.2.3   Build Environment

Platform	Build Details	
Android	Target SDK Version:	34
	Minimum SDK Version:	21
	Platform Tools:	28.0.3
	Build Tools:	34.0.0
	SDK Platform:	Android-34
	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:	18.2
	iPhoneSimulator SDK Version:	18.2
	XCode Version:	16.2
	Minimum iOS Target:	12.0

tvOS	tvOS SDK Version:	18.2
	tvSimulator SDK Version:	18.2
	XCode Version:	16.2
	Minimum tvOS Target:	12.0
MacOS	MacOS SDK Version:	15.2
	XCode Version:	16.2
	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 – x86_64)
		11.4.0 (Ubuntu 22.04.3 – arm64)

2.3   AIR for Linux – Restrictions

The AIR SDK now supports both x86_64 and arm64 based Linux platforms. These are only available to developers with a commercial license to the SDK, and have some restrictions:

- No "shared runtime" support: applications would need to be built as 'bundle' packages with the captive runtimes
- 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
- No "StageWebView" component.

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
```

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: 51.2

SWF version: 51

There are no new ActionScript APIs in this update; however, there are changes in the application descriptor file definition which means the namespace version has been updated to 51.2.

3.2 Build Tools

The Android build tools and platform used to create the AIR runtime files has been updated to Android-34 with the default target SDK now set to this level in the generated Android manifest files.

Xcode 16.1 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 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

Most of the below features were already mentioned in the 51.2.0 release notes; the key updates are the completion of the Android SecureSocket functionality, and the implementation of a number of configuration settings for the StageWebView constructor on Windows.

Reference:	AIR-6452
Title:	Updating ADT analytics to use airsdk.harman.com and log country/language
Applies to:	Core build tools
Description:	To support our internal analytics, we are now capturing the computer's country / language settings (from the Java runtime) when an application is packaged up using ADT.

Reference:	AIR-7037
Title:	`Adding support for coloured emoji using DirectWrite font support`
Applies to:	Windows runtime component
Description:	When using the existing GDI text rendering mechanism, Windows always provided emoji characters as monochrome. The rendering has been updated to use DirectWrite to display emojis in their correct colours, when using the Flash Text Engine for advanced text display. This mechanism needs a utility library to be present in a location where it can be loaded by the AIR executable. This library is by default put into the runtime's "Resources" folder; it should be automatically loaded in that location via ADL or via a packaged AIR application's executable. Note that currently there is a bug in the implementation which means it will only work if the rendering mode is "cpu", rather than "direct" mode.

Reference:	AIR-7330
Title:	`Android SecureSocket to be implemented via Android's SSLSocket class`
Applies to:	Android runtime component
Description:	In recent Android releases, the SecureSocket implementation (and the secure WebSocket usage) was broken due to issues with OpenSSL and certificate store access. To work around that, the secure sockets are now using standard Android/Java functionality. The Android implementation now correctly handles the 'certificateError' event, as well as providing details of the final certificate used via the 'serverCertificate' and 'serverCertificateStatus' properties.

Reference:	AIR-7397
Title:	`AIR Windows to support ANGLE for OpenGL ES rendering`
Applies to:	Windows runtime component

| Description: | To standardise the rendering mechanisms across platforms, and to work around problems in the Direct3D 11 based graphics within the AIR runtime, a new flag has been added to the application descriptor file: "useAngle", a boolean setting within the "initialWindow" section. If this is set to true, AIR will attempt to load the ANGLE libraries (libEGL.dll, libGLESv2.dll) and if these are found, rendering (and Stage3D support) will then proceed via OpenGL ES mechanisms.

The ANGLE binaries are not provided within the AIR SDK; they can be built from the source code available from Google, or are available from various software components such as Electron-based apps or Google Chrome.

Due to a binary incompatibility between ANGLE and the similar MESA drivers on Windows, a check has been added to ensure that only ANGLE libraries are loaded for this functionality.

Note that a number of issues have been reported with this feature, across a range of different ANGLE library versions and content. This should be considered experimental: HARMAN would be grateful to receive details of sample content and ANGLE library variants that cause instabilities, to try to address this. |
| --- | --- |

Reference:	AIR-7414
Title:	`AIR Linux support for GTK3`
Applies to:	Linux runtime component
Description:	The Linux runtime has been updated so that it uses GTK3 rather than the previous GTK2 variant. This should then help to enable some other updates and fixes to be implemented.

Note that there is still a dependency on the X11 components and so AIR will not work on a desktop running solely Wayland. |

Reference:	AIR-7415
Title:	`Audio/Video on Linux using FFMPEG`
Applies to:	Linux runtime component
Description:	Support for decoding of H.264 and AAC video and audio on Linux has now been added to the runtime, using FFMPEG libraries that would need to be available separately on the target machine.

This feature is currently only working for a subset of content and OS/library combinations and should be considered as experimental only. |

Reference:	AIR-7421
Title:	`AIR updates to shutdown the runtime more cleanly`
Applies to:	All runtime components
Description:	On some platforms the runtime shutdown relied upon the fact that the operating system killed the process. Some updates have been implemented to ensure the runtime is more gracefully closed down, in order to then enable the possibility of re-starting the application (for example, to restart after an update, or to switch rendering modes or other app descriptor settings). An API to enable restarting is planned for AIR 52.0 – however there may be some further restrictions, due to the widespread use of static variables throughout the runtime code. A restart could be considered as just unloading and reloading the 'root' SWF file, but without necessarily removing all settings and definitions that were previously set up.

Reference:	AIR-7430
Title:	`AIR Linux ADT to support 'arch' option for cross-CPU bundling`
Applies to:	Core build tools
Description:	On Linux, when creating an application bundle, it is now possible to use the "-arch" value to generate an appropriate bundle format, regardless of the CPU architecture on which the AIR tools are running. I.e. on an x86_64 machine, it will be possible to generate a bundle that would work on an ARM64 machine, and vice versa.

Reference:	AIR-7440
Title:	`ADT macOS bundles should accept an ICNS file`
Applies to:	MacOS runtime component
Description:	If a MacOS application is being generated, and an ICNS file is provided within the root of the application files, this will then be used for the application icon rather than trying to generate an icon from the provided PNG files based on the 'icon' values in the app descriptor.

Reference:	AIR-7528
Title:	`AIR ANE - API to access the graphics context (OGLES)`
Applies to:	All runtime components
Description:	For applications running using OpenGL ES rendering (i.e. Android and iOS, and Windows when using ANGLE) it is now possible to access the graphics context (i.e. "EGLContext" object) for the AIR runtime. The method is called "FREGetNativeContext3DHandle" and is documented in the FlashRuntimeExtensions.h file. Note that this handle should be used with care and not accessed if there is any change to the application window or Stage3D context. Ideally this should be obtained and then discarded within each function where it is required, and only from the main rendering thread (i.e. where the FREFunction calls are made into the ANE).

Reference:	AIR-7530
Title:	`AIR Diagnostics - app descriptor set-up in the runtime`
Applies to:	All runtime components
Description:	In AIR 51.1 it was possible to enable and configure AIR diagnostics via the use of an AIR native extension library. There is now a mechanism to configure this via the application descriptor, which will create the diagnostics internal implementation and can be used instead of the ANE to enable trace outputs to a file and to set up the configuration for other diagnostic output. The application descriptor can now contain a "diagnostics" entry at the top level, which can contain the following values: • "traceToConsole" – boolean to turn on the output of 'trace' calls to the stdout console output (assuming the SWF file hasn't had 'trace' calls stripped as part of the compilation process). • "traceToFile" – string value to turn on the output of 'trace' calls to the given filename. This will be stored in the application data folder of the operating system. • "categories" – comma-separated list of categories for which to enable diagnostic output.

Reference:	AIR-7546
Title:	`Updating license file generation and handling with validity checks`

Applies to:	All runtime components
Description:	Due to issues that developers have encountered with applications being hacked/copied and distributed with updated versions of the SWF/resource files, the license file mechanism has been updated to try to protect against this behaviour. There should be no impact or change noticed by any developer who is using the normal build tools and process to create their applications.

Reference:	**AIR-7563**
Title:	`ADT to output symbols from IPA production builds via IPASymbolFile setting`
Applies to:	Core build tools
Description:	A new configuration setting has been added to the adt.cfg file that will result in a symbol file being generated (based on the "IPASymbolFile" filename provided). This symbol file can be provided to Harman along with a crash log and will allow Harman to then symbolicate the call stack to determine what method was responsible for the crash.

Reference:	**AIR-7567**
Title:	`ADT configuration to link iOS executables via LLVM and iPhoneOS SDK`
Applies to:	Core build tools
Description:	On Windows, it is now possible to change from using the built-in linker that's provided as part of the AIR SDK, when building IPA files. Instead a "LLVM_HOME" setting can be added to the adt.cfg configuration file which should point to the root of an LLVM installation. If this is present, AIR will use the LLVM_HOME\bin\ld64.lld.exe file in order to link the binaries. Assuming a recent version of LLVM is used, this linker should then support the use of "TBD" files that are provided by Apple within their iPhoneOS, iPadOS or tvOS SDK. Note that Harman are not able to provide the Apple SDKs but if these are made available from the Windows filesystem then the "iOSPlatformSDK" configuration setting (or "-platformsdk" command-line argument) can be used to ensure the IPA is linked using the LLVM linker against an official Apple SDK. This should then resolve the issues found when linking against symbols that can move between libraries based on the different iOS versions.

Reference:	**Github-1854 https://github.com/airsdk/Adobe-Runtime-Support/issues/1854**
	Github-1493 https://github.com/airsdk/Adobe-Runtime-Support/issues/1493
Title:	`Windows only: StageWebView constructor handles userAgent, enableContextMenu`
Applies to:	Windows runtime component
Description:	A number of new capabilities have been added to control the behaviour of the WebVew2 component used on Windows to implement the ActionScript StageWebView feature. The AS3 documentation will be updated with details of the parameters – and which platforms support which features, as these are rolled out across the other operating systems – but currently, if an object argument is used for the StageWebView constructor, it can have the following properties:
	- mediaPlaybackRequiresUserAction: Boolean (default true) that determines whether or not the user would need to click to start any media from playing back on a web page.
	- userAgent: String to be set as the user agent string passed by the browser engine when requesting web content.
	- enableContextMenu: Boolean (default true) that controls whether or not the right-mouse context menu will be enabled for web content.
	- enableKeyboardShortcuts: Boolean (default true) that enables or disables shortcuts such as for navigation or browser functionality. Note that this is unlikely to disable shortcuts for text manipulation when focus is given to a text field (cut/copy/paste etc).
	- enableDevTools: Boolean (default true) that can be used to disable the ability of the user to view the web Development Tools.
	- enableStatusBar: Boolean (default true) that can be used to disable the status bar, typically shown at the bottom of the view and used to display the target for a hyperlink over which the user is hovering.
	- enableZoom: Boolean (default true) that can be used to disable the ability to zoom a web page.
	Note that even on Windows, not all behaviours are supported by all versions of the WebView2 component, so it may depend on what the user has installed. For example the documentation suggests that it would be possible to disable the "swipe navigation" (a two-finger keypad drag to go back to the previous web page), which here is set as part of the 'keyboard shortcuts' behaviour similar to an "Alt + Left" option) – however, this function had no effect on our text environment.

Reference:	Github-3616 https://github.com/airsdk/Adobe-Runtime-Support/issues/3616
Title:	`Optimising memory usage for every-frame events and lists`
Applies to:	All runtime components
Description:	When looking at memory usage in Scout, it was clear that a number of allocations and deallocations each frame are related to the event dispatch for standard events such as 'enterFrame'. Some optimisations have been included here to reduce the memory churn required when looking at the lists of listeners and creating/dispatching these kinds of events.

Reference:	Github-3647 https://github.com/airsdk/Adobe-Runtime-Support/issues/3647
Title:	`Adding IPA code signature checks on start-up`
Applies to:	iOS runtime component
Description:	Some developers had found their applications were being hacked and then re-signed using a mechanism that's not part of the approved Apple distribution process. Additional checks have been added to the runtime on iOS to ensure that the code signature within the main executable is valid and has not been tampered with or re-signed.

3.5   Bug Fixes

3.5.1   Release 51.2.1.1

Note that the below list includes the fixes from 51.2.0.1 and 51.2.0.2: to aid in viewing the more recent fixes, those earlier ones have been written in a grey font.

Reference:	AIR-7631
Title:	`AIR Windows runtime crash when using NAIP`
Applies to:	Windows runtime component
Description:	A problem in the AIR runtime was being exposed when using the native application packager (i.e. when creating a 'bundle' on Windows), resulting in a failure to create the output package. This has been corrected so that bundles can now be successfully generated and run.

Reference:	AIR-7632
Title:	AIR throws error 5016 under ADL
Applies to:	All runtime components
Description:	When launching an application with ADL and when using a 51.2 namespace, an error 5016 could be thrown due to an internal variable not being correctly initialised. This has been fixed so that normal ADL behaviour can be used.

Reference:	AIR-7662
Title:	MediaBuffer ANE API does not properly update
Applies to:	All runtime component
Description:	When testing some sample code, it was discovered that the ANE functions for locking/modifying/unlocking a MediaBuffer object did not always cause a refresh of the Sprite object for which it had been configured. Additional code has been added to ensure that the internal rendering picks up on the fact that this has changed and, so that the updated buffer is then incorporated directly into the next render cycle.

A bug had also been fixed whereby the FRESetRenderSource method could only be called once for a particular Sprite, with subsequent calls resulting in a failure that had been mistaken for memory error. |

Reference:	Github-3274 https://github.com/airsdk/Adobe-Runtime-Support/issues/3274
Title:	Ensuring OSX secure socket is robust for LetsEncrypt
Applies to:	MacOS runtime component
Description:	An issue when connecting a SecureSocket to a server using a certificate from LetsEncrypt showed that the data handling was not fully robust – in particular, not being able to cope with some extra information that was sent through in the SSL handshaking process. This has been updated and should help improve the stability of SecureSocket connections.

Reference:	Github-3394 https://github.com/airsdk/Adobe-Runtime-Support/issues/3394
Title:	Correcting AOT output for unplus (float support)

Applies to:	iOS runtime component
Description:	A problem had been found with the 'unplus' operation related to the floating point mechanism. This has been fixed within the "compile-abc" tools used when generating IPA executables.

Reference:	**Github-3426 https://github.com/airsdk/Adobe-Runtime-Support/issues/3426**
Title:	`Linux camera updates to correctly select mode including FPS`
Applies to:	Linux runtime component
Description:	On Linux, selecting a camera had been based solely on the requested resolution, and then on an internal priority list of pixel formats. This could result in a mode being selected with a bad frame rate purely because the runtime preferred that pixel format.

The logic has been significant improved to now look at the requested FPS value and the argument specifying whether to compromise on frame rate or on resolution. The result prioritises both of these over the pixel format hence providing a better match to the requested settings.

Note that this is implemented for the "Video for Linux version 2" drivers. |

Reference:	**Github-3506 https://github.com/airsdk/Adobe-Runtime-Support/issues/3506**
Title:	`Fixing Matrix3D interpolation calculation`
Applies to:	All runtime components
Description:	A discrepancy had been noticed by Ruffle developers in how the runtime calculated the interpolation between two Matrix3D objects. This led to slightly incorrect results based on the function arguments.

Following discussion, a correction has been applied but will only take effect for applications with a namespace version of 51.2 or later. Developers using this function should check that their applications still behave correctly when switching to this namespace. |

Reference:	**Github-3573 https://github.com/airsdk/Adobe-Runtime-Support/issues/3573**
Title:	`Ensuring Scout on Android connects with complex app timings`

Applies to:	Android runtime component
Description:	Due to a race condition in application start-up, when the Scout mobile helper app is installed on an Android handset, the connection details may not have been retrieved by the time the AIR application needed them. The start-up logic has been changed so that if the Scout mobile helper is installed, the application waits for the settings before proceeding. No impact would occur for applications when the Scout app isn't on the device.

Reference:	Github-3727 https://github.com/airsdk/Adobe-Runtime-Support/issues/3727
Title:	`Updating Win32 timezone cache mechanism`
Applies to:	Windows runtime component
Description:	A cache of the timezone information was being held by the runtime, in order to provide some of the Date functionality. This cache was refreshed periodically or if the user changed the local time, but was not being refreshed if the user changed the timezone that the computer was in. The logic here has been updated so that any change to the local time/timezone should invalidate the cache and ensure that the Date properties are correct.

Reference:	Github-3729 https://github.com/airsdk/Adobe-Runtime-Support/issues/3729
Title:	`Ensuring BitmapData.draw on Android picks up all content`
Applies to:	Android runtime component
Description:	A logic error/assumption within the core rendering code had meant that on Android (and potentially iOS) platforms with direct-mode rendering, some content was not appearing when a "draw to bitmapdata" call was made. The core code has been updated so that the bitmapdata rendering process should now pick up all of the content, essentially switching into a cpu-render mode for the duration of the call.

Reference:	Github-3735 https://github.com/airsdk/Adobe-Runtime-Support/issues/3735
Title:	`Regular Expression did not work correctly with unicode` `characters outside of range 0000-FFFF`
Applies to:	All runtime components

Description:	When a regular expression operated on a string containing values that require surrogate pairs in UTF-16, the match indices were incorrect for any character after such a surrogate pair. This was because the index was based on "characters" and an assumption was then made that each character takes up one slot in a 16-bit string array. The code has been changed now so that the presence of characters that require two slots (i.e. surrogate pairs) will be taken into account when calculating the index of the match.

Reference:	Github-3748 https://github.com/airsdk/Adobe-Runtime-Support/issues/3748
Title:	`AIR getTimer() returns incorrect values on time change on Linux-based OS`
Applies to:	All runtime components other than Windows
Description:	If the "flash.utils.getTimer()" API was used repeatedly, and the local clock was then changed on an OS that uses Linux, the getTimer() result would change based on the clock change i.e. could go down or have a very different value that makes it impossible to determine the elapsed time between calls. The reason for this was in the use of POSIX APIs to get a timestamp: this has been updated to prioritise the use of the "monotonic" clock which should be available on most platforms. If that clock is not available, the original code is used as a fallback.

Reference:	Github-3755 https://github.com/airsdk/Adobe-Runtime-Support/issues/3755
Title:	`Fixing ANR caused by nativeShowOriginalRect being called from UI thread`
Applies to:	Android runtime component
Description:	An earlier change to reduce the ANRs (in 51.1.3.10) exposed this next issue where a call is being made from the UI thread that is blocked by a call from the runtime thread, if they are different. This results in deadlock and an ANR crash. The "nativeShowOriginalRect" is now being pushed onto the rendering thread and handled asynchronously to avoid the UI thread being locked.

3.5.2   Release 51.2.1.2

Reference:	Github-3673 https://github.com/airsdk/Adobe-Runtime-Support/issues/3673
Title:	`Ensuring stage scalemode is updated on macOS`
Applies to:	MacOS runtime component
Description:	When an application was moved between two screens that had different pixel scalings, the Stage.scaleMode value did not update and the application then could draw in a zoomed in/out fashion after resizing it to force a full redraw. This has been fixed so that the scale mode is updated and the application automatically redraws when this changes.

Reference:	Github-3765 https://github.com/airsdk/Adobe-Runtime-Support/issues/3765
Title:	`Ensuring StageWebView constructor works on iOS`
Applies to:	iOS runtime component
Description:	The recent changes to StageWebView on Windows needed updates in other areas and the iOS implementation had an override that was then not being called, resulting in a failure to construct the object. This has been corrected.

Reference:	Github-3766 https://github.com/airsdk/Adobe-Runtime-Support/issues/3766
Title:	`Ensuring command-line apps on Linux do not crash with GTK3`
Applies to:	Linux runtime component
Description:	After the GTK3 updates, some of the library calls that had been used for timings no longer worked if there wasn't a display connect. The code here has been updated to remove the dependency on GTK and just use Glib.

Reference:	Github-3767 https://github.com/airsdk/Adobe-Runtime-Support/issues/3767
Title:	`Updating timestamp mechanism to use SHA-256 for message imprint`
Applies to:	Core build tools

Description:	The DigiCert timestamping service, used as a default when signing files such as .air packages, appears to no longer support message imprints that are hashed using the SHA-1 algorithm. This has therefore been updated to use the SHA-256 algorithm, and the signature verification functionality has been checked to ensure that it accepts this mechanism.

3.5.3 Release 51.2.1.3

Reference:	AIR-7546
Title:	AIR security - license file validation opt-out
Applies to:	All runtime components
Description:	With the updated mechanisms for license file checking, package validation meant that tools to adjust an APK or similar after its creation by ADT could cause the application not to run. For development/debugging purposes, we have added a flag so that this option can be disabled. This is currently a setting in the build configuration file (adt.cfg, in the ~/.airsdk folder): "PackageValidation=always|never"

Reference:	AIR-7677
Title:	AIR Linux to support Wayland via GDK_BACKEND
Applies to:	Linux runtime component
Description:	The AIR runtime is still dependent upon X11 APIs which had meant that users needed to ensure they were using "Wayland with Xorg" on recent versions of Ubuntu, or with similar workarounds on other distributions. In the short term we have added an automatic environment variable setting within AIR so that the initialisation code will use XWayland, in order to ensure that AIR applications work regardless of the user's desktop option.

Reference:	AIR-7682
Title:	AIR configuration for JIT verbose output
Applies to:	All runtime components

Description:	AIR already included a variety of configuration settings that can be used to adjust the runtime behaviour; a flag to turn on 'verbose mode' was present, but the AIR runtime didn't then read 'verbose only' options which can limit the outputs to only a specified set of functions. With this fix, the omission is rectified which allows the runtime to output information on the verification and JIT-compilation of specific methods.

Reference:	AIR-7683
Title:	AIR Windows crash when opening SWFInvestigator
Applies to:	Windows runtime component
Description:	A crash had been found when running the SWF Investigator tool; it's not clear how prevalent this crash might be but it was related to the inclusion of the new font rendering code. This has been fixed with defensive coding.

Reference:	Github-3723 https://github.com/airsdk/Adobe-Runtime-Support/issues/3723
Title:	Fixing diagnostics error checking for Context3D OpenGL ES failures
Applies to:	Android/iOS runtime components
Description:	When using OpenGL ES rendering, the diagnostics mechanism had been set up to notify on any GL errors – however the information provided was not accurate and was therefore of no use. This has been fixed so that the data is valid and can be used by Harman to identify the source of an error.

Reference:	Github-3729 https://github.com/airsdk/Adobe-Runtime-Support/issues/3729
Title:	Fixing Android text rendering to avoid BitmapData.draw issue
Applies to:	Android runtime component
Description:	When rendering content to a BitmapData object, objects added to the display list as a sibling but after/above a text field would not be captured. This was a side-effect of the new text rendering code added for an earlier Android update, where the text is rendered to a separate buffer and later copied back. The algorithm has been updated to ensure that the copying of a text field completes prior to the rendering of its higher siblings.

Reference:	Github-3751 https://github.com/airsdk/Adobe-Runtime-Support/issues/3751
Title:	Support individual surrogate-pair values in Strings (cf JavaScript)
Applies to:	AIR tools
Description:	The compiler was interpreting a single character code from the surrogate pair range in the standard way for Java applications, i.e. it was not a valid character and resulted in an "?" character. For example: "\uD83E" This has been updated so that the compiler treats this in the same manner as JavaScript, i.e. the character code is retained even though it's invalid as a UTF-16 string on its own.

Reference:	Github-3772 https://github.com/airsdk/Adobe-Runtime-Support/issues/3772
Title:	Adding some JIT optimisations to reduce floating point maths
Applies to:	All runtime components
Description:	When arithmetic operations are being converted into machine code, they default to using floating point maths, but if the result is then assigned into an integer type and the arguments are also integers, then the JIT code was changing the operation to use integer operations. However, an omission here meant that this was not chained i.e. this only worked if there were two operands based on variables or literal values; it did not work if one operand was itself the result of a mathematic operation. This has been fixed so that the maths operations are inspected as a chain, which should ensure we limit the floating point operations that aren't needed.

Reference:	Github-3773 https://github.com/airsdk/Adobe-Runtime-Support/issues/3773
Title:	Avoid MovieClips from advancing during splash screen time
Applies to:	All runtime components
Description:	This issue showed up on Android when an application that used Animate / MovieClip animations was displaying a splash screen. In 51.2 an updated mechanism for the splash display meant that the movie clips advanced whilst the splash screen was present, resulting in skipped/out-of-order ActionScript code rather than the expected application start-up. A change has been made to ensure no frame processing or advances happen during the splash screen period.

Reference:	Github-3677 https://github.com/airsdk/Adobe-Runtime-Support/issues/3777
Title:	Updating GTK dependency information in ANE header file
Applies to:	Core build tools
Description:	The documentation within the FlashRuntimeExtensions.h file has been updated with the update to using GTK3, in order for ANE developers to use the correct compiler/linker settings.

Reference:	Github-3779 https://github.com/airsdk/Adobe-Runtime-Support/issues/3779
Title:	Ensuring reused enterFrame event objects are reset fully
Applies to:	All runtime components
Description:	An earlier optimisation meant that the 'enterFrame' Event object dispatched on every frame is now reused between calls; however, this had not been properly reset between calling each event handler, so any handlers that called "stopPropagation" or "preventDefault" would have an impact on subsequent event listeners and frames.

The object is now being properly reset between calls to each target so that these are not impacting each other. |

Reference:	Github-3787 https://github.com/airsdk/Adobe-Runtime-Support/issues/3787
Title:	Fixing Android NativeApplication.exit() mechanism
Applies to:	Android runtime component
Description:	Improvements to the shut-down mechanism in Android had not been able to fully shut down the AIR runtime and cause it to be unloaded, which meant that when an application then was restarted, it did not properly initialise.

The earlier mechanism included a 'kill process' stage and this has been added in again at the end of the shut-down process, to ensure that applications can be properly restarted using a different process ID. |

4   Configuration File

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

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

KeepAndroidStudioOutput	When generating an Android App Bundle, rather than using a temporary folder structure and cleaning this up, this option will generate the Android Studio file structure under the current folder and will leave this in place). Values may be 'true' or 'false', default is 'false'.
AndroidPlatformSDK	A path to the Android SDK, that can be used instead of the "-platformsdk" command line parameter. Note that on Windows, the path should contain either double-backslashes ("c:\\folder") or forwardslashes ("c:/folder").
iOSPlatformSDK	A path to the iOS/iPhone/iPhoneSimulator SDK, that can be used instead of the "-platformsdk" command line parameter.
JAVA_HOME	This can be set as an override or alternative to the system environment variable that is read when ADT needs to use Java (e.g. when creating an Android App Bundle). Note that on Windows, the path should contain either double-backslashes ("c:\\folder") or forwardslashes ("c:/folder").
UseNativeCodesign	On macOS, this will mean that the IPA binary is signed using the "codesign" process rather than using internal Java sun security classes within ADT. This is "false" by default, unless ADT detects that the sun security Java classes are not available.
SignSwiftFiles	By default, any swift libraries that are included in an IPA payload are signed in the normal way. This can be turned off by setting this value to "false".
OnlyIncludeSwiftUsedArchsInSupport	If this is set to "true" then for ipa-app-store builds that include a "SwiftSupport" folder, the swift libraries will be updated via lipo to only include architectures that are used by the application (e.g. armv7 and arm64, omitting armv7s and arm64e).
OnlyIncludeSwiftUsedArchsInPayload	This is similar to the above flag but applies to the versions of the swift libraries that are included in the "Payload" folder within the IPA package. This (and the above) are now defaulting to "false" so that the swift libraries are just copied into position, but to get the legacy behaviour this should be set to "true".
iosSimulator	The name of a simulator to use when installing or running an IPA file on an iPhone simulator on mac. Note that this value will be overridden by any command-line option or by an environment variable should this be set as well (i.e. AIR_IOS_SIMULATOR_DEVICE).

IPASymbolFile	To aid in debugging iPhoneOS/tvOS issues, this setting has been introduced which should give the filename of a symbol file that will be generated as part of the iOS build process. This isn't a human-readable file, but if a crash log is produced from an AIR application on iOS/tvOS, this file can be provided to HARMAN along with the crash log in order for us to investigate the crash location and call stack.
LLVM_HOME	[Windows only, currently] Specifies the installation directory for the LLVM toolchain. If this entry is present, ADT will use the LLVM linker called "ld64.lld.exe" situated in the "bin'" folder of the LLVM_HOME location. When switching to the LLVM implementation of the linker, it is then possible to use the "iOSPlatformSDK" setting (or the "-platformsdk" command-line argument to reference the actual Apple iPhoneOS SDK which means linking will take place against the "TBD" files, and Apple's newer dynamic linking/loading mechanisms should then work across the different iOS versions. This mechanism should result in more stable binaries than when linking against the "stub" SDK files provided in the AIR SDK. These stub files will be removed in the future, with LLVM becoming the standard mechanism for linking on non-macOS platforms.
PackageValidation	Whether or not the application should validate the package contents at start-up. With AIR 51.2, license files include information about the package that is created by ADT, and the runtimes will validate that these have not been significantly tampered with. This check can be disabled if this flag is set to "never" – default is "always". In future we may change how this flag works e.g. for only specific applications, or for only debug-type packages.

5   Android builds

5.1   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.

5.2   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

5.3   Android Text Rendering

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

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

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

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

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

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

5.4   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.

6   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.

Document Id: HCS19-000287

7   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.

8   iOS support

8.1   32-bit vs 64-bit

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.

8.2   MacOS remote linking from Windows

Due to a number of updates from Apple around the mach-o linker, and the movement of symbols between different component libraries, it has become increasingly problematic to link Apple binaries on a Windows computer. Originally, Adobe had cross-compiled the "ld64" Apple linker, but without support for the "TBD" format that Apple use for the iPhoneOS/AppleTVOS SDKs. To work around this limitation, the AIR SDK includes "stub" binaries for the SDKs – but it is not then possible to support the movement of symbols i.e. where a particular symbol is found in different frameworks for different iOS versions.

Using LLVM's linker, which supports the mach-o format, it was also found that Apple restrictions had been preventing some applications from being published via the App Store due to a difference in how symbols were found/stored, and the known/unsupported issues in LLVM meant that this is also not a completely viable solution.

The solution that we will work with now is to use a mac machine to perform the link stage of the build process. The rest of the development and build process can still occur on Windows but linking the AIR application's object files against the iPhone / AppleTV SDKs should be done on a mac.

There are two ways to achieve this: initially a manual mechanism to allow files to be pushed to a macOS machine, linked via a script, and then the result copied back to the Windows machine where the packaging command needs to be run again to pick up the binary. And with the release of 51.0.1 this is now possible to handle automatically within a single run of ADT, following some initial machine configuration. Details on these two methods follow.

8.2.1   Manual copying and linking

There are a number of steps to the build process in this scenario.

1. Configure ADT to use a specific folder, into which all linker inputs will be placed.

To do this, edit the "adt.cfg" file (in your home folder under an ".airsdk" subfolder) and add a line: "`IPALinkFolder=c:/path/to/link/folder`". This must be the name of an existing folder, under which subfolders will be created for each run of ADT. Note that you need to use forward-slashes, or escaped backslashes ("\\"), due to how Java reads in property files.

2. Run your normal link command via ADT.

This will then generate a subfolder under your "IPALinkFolder" location, which contains a script file and all the input files needed for the Apple linker.

3. Copy this link folder to your macOS computer.

This can be done with SFTP/SCP or similar tools, or potentially you could have a network shared folder set up.

4. On the macOS computer, run the linker.

Using a terminal window, you will first need to set an environment variable, "AIR_SDK_HOME", and then run the script that was generated by ADT. For example:

> export AIR_SDK_HOME=/Users/username/Downloads/AIR_SDK/AIRSDK_51.0.1
>
> ./linkerscript.sh

5. Copy the resulting file back onto the Windows PC.

The file should be called "linkerOutput" and should be an arm64 macho executable file.

6. Call ADT again, this time providing the linked file.

To do this, add the arguments "`-use-linker-output path_to_linkerOutput`"; this can go within the normal input files list, or at the end of this (similar to "-extdir").

ADT will then ignore the normal command to link the binary, and will use the provided executable in order to package and sign the IPA file.

7. Clean up.

The folder that's created under the "IPALinkFolder" location, as well as the linkerOutput file (and of course the files that have been copied to the macOS machine) are not automatically deleted. So these should be periodically cleaned up manually to avoid wasting disk space.

8.2.2   Programmatic remote linking

In order to automatically allow the Windows machine to connect to the macOS machine and to copy files and drive the linker, a password-less mechanism will need to be set up to allow remote access without any user interaction. This requires the use of SSH keys: unless a key-pair is created that doesn't have a passphrase, it will be necessary to use "ssh-agent" to store the passphrase and associate this with the user's Windows credentials.

To set this up (one time only):

1. Create a new key-pair (unless you want to use an existing pair).

On Windows, run "`ssh-keygen`" and provide a filename – the default is "id_rsa" but in this walkthrough we shall use "adt_access". It then prompts for a passphrase: if you leave this blank, you will not need to follow the "ssh agent" steps below, but the recommendation would be to create a suitably secure passphrase for this. You should then have two files, "adt_access" and "adt_access.pub".

2. Install the public key on the mac machine.

You can use sftp/scp for this. The key should be added into your ".ssh" folder – note that you need the username of the mac machine, which we shall assume is just "user". You will then need to configure SSH to allow this public key to be used for connections: if you remote in (or just open a terminal) on the mac, go into the ".ssh" folder, and run: "`cat adt_access.pub >> authorized_keys`". This adds the new key onto the end of the authorized keys list.

3. Set up ssh agent to provide the passphrase.

Firstly you will need to check that ssh-agent is running: open "Services" on the computer, and find an entry with name "OpenSSH Authentication Agent". This should be changed to "Automatic", or "Automatic (Delayed Start)" if you prefer, and if necessary, also started manually. The "Status" column should show that this is running.

Then in a Windows console, run "`ssh-add adt_access`" and provide your passphrase.

Note that if you get an error message "Permissions for 'private-key.ppk' are too open", you will need to ensure that only the current user is able to access the private key file ("adt_access"). This means adjusting the "Security" properties on this file, changing the owner of the file to the current local user account, removing inheritance and inherited permissions, and removing all permissions for users/groups other than the current local user. For more details, see the below link:

[Windows SSH: Permissions for 'private-key' are too open - Super User](#)

You can then test the connection by running "`ssh -i adt_access user@mac_ip_address`", which should then log on automatically without further prompting.

4. Provide the configuration to ADT.

Now that you have the connectivity set up, you need to create a configuration file for AIR. You will need to add two entries into the "adt.cfg" file that is in your "c:\users\username\.airsdk\" folder:

```
IPALinkFolder=c:/path/to/link/folder

RemoteLinkMachine=mac_ip_address
```

The first setting is to provide a location into which the linker will output all of the files. This is not strictly necessary but will aid in debugging problems.

The second provides the network location of the remote machine onto which you've put the public ssh key.

You will then need to create a configuration file with the name of this "mac_ip_address" network address, with an ".cfg" extension, and put this into a subfolder "remote_link_configs" under the .airsdk directory. For example:

```
C:\Users\username\.airsdk\remote_link_configs\192.168.1.3.cfg
```

The contents of this file should be:

```
CertPath=C:/path/to/private/key/adt_access

Username=user

SdkFolder=/Users/user/Documents/AIR_SDKs/AIRSDK_51.0.1
```

The "CertPath" value points to the private key that we've named "adt_access", again please note the use of forward-slashes or double-backslashes in the Windows path. "Username" is the user associated with the key from when this was added to "authorized_keys". And "SdkFolder" is the path on the remote mac machine where an AIR SDK can be found. This path is only used for the runtime libraries i.e. "libRuntimeHMAOT.arm-air.a" and "builtin_abc.arm64-air.o", the linker won't use this for the actual link binary (ld64) or the stub files; instead, the remote script picks up your iPhoneOS SDK using the "xcrun" utility.

Once that is all set up, you can use ADT as normal for IPA builds, and the remote linking will happen in the background. If there are issues, please check the adt.log (or use AIR SDK Manager's "Troubleshooting" tab) and report an issue via Github.

Please do note that the link folders are not (currently) cleaned up with this approach, so the location under the "IPALinkFolder", and its copy that is pushed to the remote Mac device (with the same name, within the user's home folder) will still exist after the ADT process has completed. This will help with debugging any issues, but we expect to change this in the future.

9   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.

9.1   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.

9.2   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.

9.3   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.

10   AIR Diagnostics

10.1   Purpose

The goal of the AIR diagnostics implementation is to allow both developers and HARMAN to benefit from additional metrics around an appication for debugging purposes. One of the key goals is to allow errors that occur in the field to be detected and reported back, with an initial focus being around the Android "Application Not Responding" issues that are relatively common and can trigger the 'bad behaviour' labels from the Google Play Store.

There have also been a number of situations where HARMAN are unable to reproduce issues, and where additional logging has been added to the AIR runtime for developers to then reproduce a problem and report back. With the framework in place for AIR diagnostics, such logging could then start using this mechanism, and could then be left in place and become part of the generic runtimes rather than needing customer-specific builds.

10.2   Mechanism

Implementing a mechanism to capture diagnostics has to also consider the performance of the runtime, as we do not want to significantly impact performance (or memory footprint) of the deployed applications. It is important therefore that any checks as to whether a particular diagnostic should be captured/reported should be as minimal as possible, and no processing of data specific to this should occur if the relevant category of diagnostic has not been enabled.

Internally, we have used ANEs as the basis of the mechanism to enable the diagnostics, to select which categories to enable, and to receive feedback from the runtime. The ANE native implementation is built into the runtime, but needs to be enabled through the inclusion of an ANE, or more accurately a SWC library that provides the API for this and that then communicates with the runtime.

To enable diagnostics then, an application will need to add the extension ID to their application descriptor file: "`com.harman.air.AIRDiagnostics`". The application can then configure the diagnostics to specify a reporting folder, or to check for existing reports left from previous runs of the application, or to get more details on a report. It can add listeners for feedback for particular situations and can configure the categories of diagnostics that it wants to listen for.

The standard case for diagnostics should be that the AIR runtime writes relevant information (asynchronously!) to log files, and these can then be interpreted to generate reports of the data. The data shoudl be machine-readable so different structures and schemas will be defined for these as relevant. One of the benefits of using an ANE mechanism is that this can then be adapted and extended more rapidly than if we used a built-in ActionScript API (as well as keeping all of this logic outside of the runtime and only included on-demand).

Typically when the application exits, the diagnostic reports that are being generated are then removed. This obviously helps to limit the size of the storage needed for diagnostics, but also means that an application can check on start-up for the existance of a report: and if it's found, it implies that the application may have had an uncontrolled exit the last time it was used. If that was, for example, caused by an Android ANR with the OS shutting down the application, it's possible that the "long function" diagnostic may contain the clues as to the cause of this behaviour.

10.3   Categories

The number of categories will be expanded as time goes by, so this list will be kept in sync with the availability of each category within the relevant runtime version.

10.3.1   Long-running functions

ANR problems can happen if a call into the AIR runtime blocks the UI thread for too long. To try to find if there are functions that generally run for longer than expected, this category has been added to try to help identify the culprit. The functions that are tracked are:

- Processing a frame (i.e. executing all 'enter frame' type event handlers and normal frame advance behaviours)
- Rendering a frame (i.e. the drawing / graphics code)
- GC: marking non-stack roots
- GC: marking queue and stack
- GC: sweeping

Functions are checked every second to see if they are still running. This is an excessive amount of time and so will be logged. If a function subsequently completes, but takes over 2 seconds, then a notification event is sent out from the diagnostics ANE.

If the runtime is killed by the OS then a report should be available that contains information about which functions have taken a lot of time, to see if this information shows a pattern of a particular function that may have been starting to increase in duration.

10.3.2   Garbage Collection activity

This is often an area that is considered problemmatic particularly in the final phase of collection. AIR runs garbage collection on a frame-by-frame basis (using reference counting) as well as on a mark-and-sweep basis (using roots and finding objects that are not then reachable from these). This category focuses on the mark-and-sweep approach, and will notify of the start of an incremental marking session (meaning that some condition within the runtime has triggered the start of garbage collection), the end of incremental marking, the start and end of the final stack-based marking, and the start and end of the 'sweep' phase where object destructors are called and memory clean-up and consolodation happens. The metrics include memory usage at each stage so this may also help to see whether there had been any benefit in collection at this point, which may help inform any tweaks that may be needed to the garbage collection policy.

Note that if the final stack marking and sweeping takes too long, this will also be notified as a long-running function.

10.4   Diagnostic API and guide

At the time of writing, the API is still being finalised; this will be released shortly and the actual API and documentation will be provided at that time.

In the meantime, there is a sample application that demonsrates how the ANE can be used to request some diagnostics and check the results of this: this will be updated periodically, and the latest ANE can be downloaded from the 'ane' subfolder:

https://github.com/airsdk/Adobe-Runtime-Support/tree/master/samples/air_diagnostics

10.5   FAQs

How do I get information off the device?

Currently this will have to be done by the application logic. The API includes some ways to get at the data and this could be wrapped into calls to a back-end service. HARMAN are considering providing a service here that could receive an application's diagnostics and make this available to both the application developers and to ourselves, to help in remote debugging; however, at this point in time it woudl be up to the application developer to somehow detect the presence of a report and send the information somehow.

What are the privacy concerns?

We are not intending to collect customer data, or any information that could allow a specific customer to be identified. Information should be solely related to the application itself, as well as some general details about the device (OS/version/CPU/etc).

It is expected that developers will be providing a privacy policy to their end users, and this should mention the collection of information in order to improve the application or service, in order to cover the use of this diagnostics mechanism.

Why do we not just extend the capabilities of Adobe Scout?

We had considered adding additional capabilities to Scout, in particular around the memory and GC mechanisms. But the real issue is that we want to collect data from applications deployed in the field, with end users who will not have any development tools or debugging expertise. So the diagnostics system is set up to be self-contained within an application, with the end user not having to do anything themselves.

How can I request different categories for extra debugging?

If there are specific areas of concern or requirements for debugging, please raise a ticket on the Github system: https://github.com/airsdk/Adobe-Runtime-Support/issues

If you have an existing issue open that you believe would benefit from this approach, please add a comment to the ticket and raise this as a possibility.