Adobe AIR SDK Release Notes

Version	51.0.0.4
Date	19 March 2024
Document ID	HCS19-000287
Owner	Andrew Frost

Table of contents

1 Release Overview

This is a pre-release / beta version of the AIR SDK. Please note that there are a number of features that may not be fully completed, and the maturity of this release is likely to be lower than normal. This version of the AIR SDK should not be used for production applications, instead it is intended for people to explore the new features and APIs, and to provide feedback on this – as well as to check their existing applications against, to guard against regressions.

Build 51.0.0.1 did not make it to public release so these release notes are updated with a few additional changes for 51.0.0.2, but without creating new/colour-coded sections for this. Build 51.0.0.3 is a version with bug fixes for some of the issues found within 51.0.0.2, and information on this is provided in a light blue font. Build 51.0.0.4 includes a number of updated features and fixes, with information provided in orange font.

1.1 Key changes

There are a number of new features/APIs that are available within this release. A better description of these will be posted on the Github discussion board under https://github.com/airsdk/Adobe-Runtime-Support/discussions but a brief list is below:

Single-precision floating point. A new “float” class is introduced with 32-bit data storage for single-precision numeric values, along with some new methods for Stage3D classes to take advantage of vectors based on this type. The new AS3 compiler is required for this, and SWFs generated from source code that uses this data type will not run on any AIR runtime prior to 51.0.

Linux ARM64 support. The Linux variant of the AIR SDK has been updated to include both x86_64 and arm64 binaries. It is possible to switch between these using a ‘configure’ command, which uses symbolic links to switch between the binaries. In 51.0.0.4, ADT has been updated to correctly package and extract ARM-64 based AIR Native Extensions.

Zip archive support. New ActionScript classes are added to allow zip files to be handled directly within ActionScript, including shortcuts to carry out common zip tasks via a background thread. With 51.0.0.4, it is possible to create and save zip files as well as loading and unzipping them.

WebSocket support. New ActionScript classes for handling the WebSocket protocol to allow browser-based applications to connect and communicate with AIR applications. Support is now included for AIR to act as a client, connecting to another websocket server, as well as for AIR to act as the socket server.

Encryption capabilities. Basic encryption and decryption of ByteArray data, along with generation of a random byte stream using the platform’s cryptographic functions.

ByteArray updates. New utility methods to convert between binary and string data formats. This now supports base64 encoding/decoding, and hexadecimal (i.e. base16) encoding/decoding.

Other language features. Minor updates to a number of classes as requested by various developers via the Github forums. This includes String ‘startsWith’ and ‘endsWith’ functions, Array/Vector ‘includes’ and ‘isEmpty’ functions, and tweaks to StageWebView, StageText and TextField APIs.

New ANE functionality. Some additional APIs have been added to the C API (and these will also be provided via the Java API at some point), mostly to aid in the display of external content via a display list object.

Fixes in 51.0.0.3 can be found in section 3.5.2.

Fixes in 51.0.0.4 can be found in section 3.5.3. Note that there are some updated features as well, listed immediately before section 3.5.

1.2 Deployment

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

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

1.3 Limitations

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

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

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

1.4 Feedback

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

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

1.5 Notes

Contributors to the https://airsdk.dev website would be very welcomed: this portal is being built up as the repository of knowledge for AIR and will be taking over from Adobe’s developer websites

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

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	51.0.0.2	51.0.0.3	51.0.0.4
Core Tools	3.0.0	3.0.1	3.0.2
AIR Tools	3.0.0		
Windows platform package	3.0.0	3.0.1	3.0.2
MacOS platform package	3.0.0	3.0.1	3.0.2
Linux platform package	3.0.0	3.0.1	3.0.2
Android platform package	3.0.0	3.0.1	3.0.2
iPhone platform package	3.0.0	3.0.1	3.0.2

2.2.2 Delivered Documentation

Title	Document Number	Version
Adobe AIR SDK Release Notes	HCS19-000287	51.0.0

2.2.3 Build Environment

Platform	Build Details	
Android	Target SDK Version:	33
	Minimum SDK Version:	16 (ARMv7, x86); 21 (ARMv8, x86_64)
	Platform Tools:	28.0.3
	Build Tools:	33.0.2
	SDK Platform:	Android-33
	Note – these are the versions we use to build the AIR SDK and runtime, we also recommend developers match the same ‘target SDK’ version as here.	
iOS	iPhoneOS SDK Version:	17.4
	iPhoneSimulator SDK Version:	17.4
	XCode Version:	15.3
	Minimum iOS Target:	12.0

tvOS	tvOS SDK Version: tvSimulator SDK Version: XCode Version: Minimum tvOS Target:	17.4 17.4 15.3 12.0
MacOS	MacOS SDK Version: XCode Version: Minimum macOS Target:	14.4 15.3 10.15
Windows	Visual Studio Version:	14.0.25431.01 Update 3
Linux	GCC Version	11.4.0 (Ubuntu 22.04.1 – 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: 51.0

SWF version: 51

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 51.0. The namespace update is required for opening any SWF file that's got a SWF version of 51, or when using any of the new XML application descriptor flags.

3.2 Build Tools

The Android build tools and platform used to create the AIR runtime files has been updated to Android-33 with the default target SDK now set to this level in the generated Android manifest files.

Xcode 15.2 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 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

For full details, please see What's New (airsdk.dev)

Package/Class	Updated Element	Element name
air.security.Encryption	New Class	Encryption
air.system.License	New Class	License
flash.text.StageTextContentType	New Class	StageTextContentType
air.net.WebSocket	New Class	WebSocket
flash.events.WebSocketEvent	New Class	WebSocketEvent
air.utils.ZipArchive	New Class	ZipArchive
air.utils.ZipEntry	New Class	ZipEntry
flash.display3D.Context3D	New Method	setProgramConstantsFromFloatVector
flash.events.DataEvent	New Property	WEBVIEW_MESSAGE
flash.net.DatagramSocket	New Method	broadcast
air.security.Digest	New Property	SHA512
flash.events.MouseEvent	New Property	MOUSE_WHEEL_HORIZONTAL
flash.events.SecurityErrorEvent	New Property	CERTIFICATE_ERROR

flash.net.Socket	New Property	tcpNoDelay
flash.text.StageText	New Property	contentType
flash.media.StageWebView	Updated Constructor	StageWebView
flash.media.StageWebView	New Method	postMessage
flash.system.System	Updated Method	decryptBlob
flash.text.TextField	New Property	allowedDomains
flash.display3D.VertexBuffer3D	New Method	uploadFromFloatVector
Array	New Methods	includes
Array	New Methods	isEmpty
flash.utils.ByteArray	New Methods	createFromBase64
flash.utils.ByteArray	New Methods	createFromHexString
flash.utils.ByteArray	New Methods	writeBase64
flash.utils.ByteArray	New Methods	writeHexString
flash.utils.ByteArray	New Methods	toBase64
flash.utils.ByteArray	New Methods	toHexString
flash.utils.ByteArray	New Methods	writeRandomBytes
float	New Class	float
String	New Methods	endsWith
String	New Methods	startsWith
Vector	New Methods	includes
Vector	New Methods	isEmpty
Function	New Property	declaration
NetworkInfo	New Property	disableNetworkChanges

3.4 Features

Reference:	**AIR-309**
Title:	`Adding AS3 API for allowing AS3 to handle HTTPS certificate errors`
Applies to:	All runtime components
Description:	The “certificateError” event had been introduced in AIR 50.2.3 as an anonymous event (i.e. the string had to be used manually); this change introduces a definition for it under the SecurityErrorEvent class. A runtime error, code 2128, has been introduced for this.

Reference:	**AIR-5963**

Title:	`Add ANE capabilities to render a Sprite using a MediaBuffer - initial support via BitmapData`
	
	`Allowing ANEs to lock, modify and unlock MediaBuffer content for updating a display object`

Applies to:	All runtime components
Description:	A sprite object can be updated using the air.media.MediaBuffer class, which can be created from a BitmapData object (or from an embedded Bitmap object). The MediaBuffer can be used as the source for rendering, and with the Lock/Unlock methods it can also be updated from the ANE code to result in animation effects.

Reference:	**AIR-6012**
Title:	`AS3 API for StageWebView constructor changes`
Applies to:	All runtime components
Description:	The StageWebView constructor has been adjusted to make it more flexible, in order to add new features/capabilities in the future for configuring the platform-specific WebView components. Note that currently this doesn't include any enhancements, but these will be made possible via the new constructor mechanism.

Reference:	**AIR-6051, AIR-6053, AIR-6054**

Title:	`AIR zip support: Basic reading in of zip files to get entry details`
	
	`Create initial AS3 API and framework for zip file support in AIR`
	
	`Adding unzip/inflate support for Zip files`

Applies to:	All runtime components
Description:	New APIs for handling zip files. Currently a zip can be read in – or constructed manually – and extracted into a folder. Writing a compressed zip or creating one from a folder will be added in a future update.

Reference:	**AIR-6063**
Title:	`Updated OpenSSL-based ELS key storage`
Applies to:	All runtime components
Description:	To work around some of the platform-specific problems we had had with storage of keys for the Encrypted Local Store functionality, an OpenSSL-based mechanism has been created in conjunction with platform-specific key store. This change is transparent to the users of the ELS APIs but should result in a better user experience without pop-up dialogs.

Reference:	**AIR-6279**
Title:	`AIR runtime support for float (removing float4 code)`
Applies to:	All runtime components + Core build tools
Description:	Changes have been added into the runtime for a 32-bit “float” data type, which can also be used with the compiler present in The non-Flex versions of the AIR SDK 51.x. Adobe had originally also considered a “float4” structure but this has been excluded.

Reference:	**AIR-6288**
Title:	`AIR AS3 API for encrypting and decrypting a byte array`
Applies to:	All runtime components
Description:	Functionality for encrypting/decryption a byte array object using industry standard mechanisms – via OpenSSL. These APIs are not currently implemented on iOS, for which a non-OpenSSL version will be needed (i.e. using the Apple APIs).

Reference:	**AIR-6425**
Title:	`AS3 API for Socket.tcpNoDelay setting`
Applies to:	All runtime components

Description:	The capability to request ‘no delay’ on a TCP socket had been added in 50.2.1, using a hacked method to set this flag via the target address. This update introduces this properly as a property on the Socket class.

Reference:	**AIR-6579**
Title:	`AS3 String startsWith and endsWith`
Applies to:	All runtime components
Description:	New methods on the String class to provide easier methods to check for comparisons between substrings occurring at the start or end of another string.

Reference:	**AIR-6580**
Title:	`Add 'includes' and 'isEmpty' to Array and Vector classes`
Applies to:	All runtime components
Description:	Utility methods for Array/Vector classes, to see whether a collection contains an element, or to see if it’s empty, without having to iterate or do a length comparison.

Reference:	**AIR-6581**
Title:	`Adding ByteArray conversions to/from base16 and base64`
Applies to:	All runtime components
Description:	String to binary conversions using hex (base16) and base64 string formats.

Reference:	**AIR-6707**
Title:	`Adding DatagramSocket.broadcast() method for UDP broadcasts`

Applies to:	All runtime components
Description:	The broadcast capability had been added recently but with some limitations, this function now allows developers to properly send out broadcast messages to a particular network.

Reference:	**AIR-6752**
Title:	`AIR Digest to include SHA-512 support`
Applies to:	All runtime components
Description:	Addition of the SHA-512 digest/checksum mechanism.

Reference:	**AIR-6991**
Title:	`ByteArray write random bytes`
Applies to:	All runtime components
Description:	Ability to overwrite a ByteArray with cryptographic random data.

Reference:	**AIR-6992**
Title:	`Update SWF tag encryption to support custom keys`
Applies to:	All runtime components + Core build tools
Description:	The ‘encrypt’ label in an Embed directive can be set to a custom string, in order to use a different key when encrypting an embedded byte array. The decryption now takes this as an optional key; default behaviour will remain the same but should not be considered secure. The key can be provided on the command-line for the compiler, if required, rather than in the source.

Reference:	**AIR-7018**

Title:	`ADT move all iOS linker inputs into a single folder structure`
Applies to:	Core build tools
Description:	In preparation for the “remote mac linking” operations (to link a binary on a mac computer whilst building/packaging on a Windows one) the output structures have been changed to keep everything in a single temp folder.

Reference:	**Github-216** https://github.com/airsdk/Adobe-Runtime-Support/issues/216
Title:	`Support for WebSocket servers`
Applies to:	All runtime components
Description:	Adding a WebSocket class and implementation of a server, allowing an incoming HTTP connection to be promoted to WebSocket and handling the data transfers. Client functionality (for outgoing connections) is pending.

Reference:	**Github-1242** https://github.com/airsdk/Adobe-Runtime-Support/issues/1242
Title:	`Allow images in HTML text fields via 'allowedDomains' property`
Applies to:	All runtime components
Description:	Web-based images had been blocked from HTML-based text fields, for security reasons. This property allows the developer to determine which domains are considered ‘safe’ to allow such images to be used.

Reference:	**Github-1858** https://github.com/airsdk/Adobe-Runtime-Support/issues/1858
Title:	`Adding horizontal mouse wheel support`
Applies to:	All runtime components
Description:	Horizontal mouse wheel events are now being dispatched by interactive objects, on desktop platforms and on Android where supported.

Reference:	**Github-1936** https://github.com/airsdk/Adobe-Runtime-Support/issues/1936

Title:	`Allow windows.UseWebView2 'exclusive' mode to prevent IE/WebBrowser usage`
Applies to:	Windows runtime components and Core build tools
Description:	The “UseWebView2” application descriptor option for Windows has been updated from a simple true/false to also support “exclusive”, which would mean that if the Edge WebView2 component is not present, the StageWebView constructor would fail (throw an error) rather than falling back to using the earlier IE-based WebView.

Reference:	**Github-2625** https://github.com/airsdk/Adobe-Runtime-Support/issues/2625
Title:	`Adding NetworkInfo.disableNetworkChanges flag to prevent socket disconnects`
Applies to:	Desktop runtime components
Description:	As a workaround to spurious socket disconnects, it is now possible to disable the AIR runtime from monitoring and responding to network information changes on a platform. This setting is initially off (false) and if set to true, will disable the NativeApplication “networkChange” event as well as the internal handling of network changes within the media code.

Reference:	**Github-2742** https://github.com/airsdk/Adobe-Runtime-Support/issues/2742
Title:	`Adding Function.declaration property to find details of a function`
Applies to:	All runtime components
Description:	With a reference to a Function object, it is possible now to request a string representation of the function prototype. This can also be used to determine whether or not the function is a member of a class, or an anonymous function, etc.

Reference:	**Github-3060** https://github.com/airsdk/Adobe-Runtime-Support/issues/3060
Title:	`AS3 StageTextContentType class for StageText support for OTP SMS entry`
Applies to:	All runtime components

Description:	A new class/API is added to allow a StageText object to be given a 'content type' hint. This can be useful to enable users to easily copy/paste from their SMS messages into the StageText object, if they have just received an SMS with a one-time password.

Note: the below updates were made in 51.0.0.3/51.0.0.4 but are considered 'features' rather than bug fixes. Whilst the version is still in a beta/pre-release status, we are updating some features without bumping up the minor build number or milestone number.

Reference:	**Github-1917 https://github.com/airsdk/Adobe-Runtime-Support/issues/1917**
Title:	Updating ADT for ANE support on Linux-ARM64
Applies to:	Core build tools
Description:	A new platform, "Linux-ARM64", has been created for ANEs to use. This can be added to the extension descriptor file and to the command-line for ADT.

Reference:	**Github-3085 https://github.com/airsdk/Adobe-Runtime-Support/issues/3085**
Title:	Adding exception stack trace when Android ANEs throw exceptions
Applies to:	Android runtime component
Description:	A minor change in the ExtensionContext.call() mechanism for Android Java-based ANEs: if the FREFunction call threw an exception, the information about this exception is now printed out via the JNI "ExceptionDescribe" mechanism, which means some warning output will be displayed in logcat.

Reference:	**AIR-6054, AIR-6055, AIR-7046**
Title:	Completing AIR Zip support: Adding support for file modification times for zip entries AIR Zip support for creation and saving of zip archives Adding ZipArchive.load/saveFromByteArray implementations
Applies to:	All runtime components
Description:	Updates to complete this functionality, allowing zip archives to be created manually and saved, or to be created from a set of folders asynchronously.

Reference:	AIR-6866
Title:	Enable NativeWindow class for Android and iOS
Applies to:	Android and iOS runtime components
Description:	As a first step towards supporting various windowing features of the mobile operating systems that Google/Apple are introducing, it is now possible to get access to the 'NativeWindow' object for a stage, for accessing details about the window. Support may be limited for events etc and it is not yet possible to create a new NativeWindow on those platforms.

Reference:	AIR-7018
Title:	Remote IPA linking support: Add an IPALinkFolder build configuration option Creating linker script for running on macOS Add an ADT option for "-use-linker-output"
Applies to:	Core build tools
Description:	A new mechanism is being set up so that the "link" phase of building an IPA file can be carried out on a remote macOS machine, when developing on Windows. At this stage, there are three steps: 1. In the build configuration file (adt.cfg), there is a new setting that can be provided, "IPALinkFolder". If specified, then rather than using/cleaning a temporary location, AIR will create a subfolder under the specified directory and will put all the necessary files for linking the application's executable into this folder. 2. It also generates a script file (.sh). So once ADT has been run, and the subfolder has been created under the IPALinkFolder location, that subfolder can be zipped up and copied over to a macOS computer. An environment variable (AIR_SDK_HOME) needs to point to the root of an AIR SDK installation, and then the linker build script can be run to build the executable file. 3. Once the executable has been created on the macOS computer, it should be copied back onto the Windows PC. ADT can be run again as before, but this time providing an option "-use-linker-output" followed by the path to the executable file from step 2. This will cause ADT to skip the "link" phase and to just treat that file as if it were the output of linking.

Reference:	**AIR-7044**
Title:	AIR TextLine embedded fonts to support COLR and CBDT tables
Applies to:	All runtime components
Description:	In order to support embedded fonts that contain colour emoji characters, some changes have been made to the text rendering and font handling code for “TextLine” (i.e. Flash Text Engine) mechanisms. These do not change any functionality when using a normal TextField object, just when generating flash.text.engine.TextLine objects from a TextBlock, when using a font file that was either embedded with CFF outlines, or loaded in dynamically from an OpenType or TrueType file. The support for “COLR” tables is for v1 - multi-layer bitmaps each with a designated color – so does not support gradient fills and similar from v2. The support for “CBDT” tables should be complete in terms of the colour (PNG-based images) support.

Reference:	**Github-149:** https://github.com/airsdk/Adobe-Runtime-Support/issues/149
Title:	Implementing FontDescription.createFromByteArray to load an OpenType/TrueType font for FTE
Applies to:	All runtime components
Description:	This functionality allows a raw font file (e.g. .ttf) to be loaded in at runtime so that it can be used within the Flash Text Engine framework as an embedded font. FTE supports both TrueType and OpenType font outlines, and now also includes support for colour bitmap characters and colour tables.

Reference:	**Github-216:** https://github.com/airsdk/Adobe-Runtime-Support/issues/216
Title:	WebSocket client connection and handshaking
Applies to:	All runtime components
Description:	Outgoing WebSocket connections are now supported, with the handshaking for promoting an HTTP request into a WebSocket connection.

Reference:	**Github-PAD20** https://github.com/airsdk/ANE-PlayAssetDelivery/issues/20
Title:	`Updating NetStream.play() to accept IDataInput argument`
Applies to:	All runtime components
Description:	The “play()” argument for NetStream, as well as taking a URL or file path (or null for appending bytes), is now able to take an IDataInput argument. This could be a ByteArray, or it could be an AssetFile (from the Play Asset Delivery ANE). The NetStream object will read this and buffer it in the same manner as it would read a local file path that had been passed in. This means that it can also support MP4 file containers.

3.5 Bug Fixes

3.5.1 Release 51.0.0.1 and 51.0.0.2

Reference:	**AIR-6840**
Title:	`FileReference.upload() to cope with binary file responses (Windows)`
Applies to:	Windows runtime components
Description:	The FileReference.upload() method has assumed a simple text-based HTTP response. For Windows this has been extended now to handle a binary response, which will be made available via the “data” parameter of the DataEvent received in the DataEvent.UPLOAD_COMPLETE_DATA handler.

Reference:	**Github-2318** https://github.com/airsdk/Adobe-Runtime-Support/issues/2318
Title:	`Removing StageVideo viewport coordinate limits for AIR 51 apps`
Applies to:	All runtime components
Description:	For an application with the 51.0 namespace, the viewport coordinates will no longer be limited for a StageVideo object. It is then up to the developer to test and ensure behaviour is appropriate on the target platforms.

Reference:	**Github-3024** https://github.com/airsdk/Adobe-Runtime-Support/issues/3024
Title:	`Removing StageWebView viewport coordination limits for AIR 51 apps`

Applies to:	All runtime components
Description:	For an application with the 51.0 namespace, the viewport coordinates will no longer be limited for a StageWebView object. It is then up to the developer to test and ensure behaviour is appropriate on the target platforms.

Reference:	**Github-3062 https://github.com/airsdk/Adobe-Runtime-Support/issues/3062**
Title:	`Updating Win32 camera handling to include better fallbacks where direct connect fails`
Applies to:	Windows runtime component
Description:	If a camera source could not be directly connected to the AIR DirectShow filter, or via a conversion filter, the resulting situation meant that no mode information was available (negative width/height) and that no camera previews were available. The fallbacks have been updated to ensure that if the “capture graph builder” is able to make a connection, then the camera is now supported properly. Specifically this now works for EOS Webcam utility, but the OBS virtual camera is still not working..

3.5.2 Release 51.0.0.3

Reference:	**AIR-6279**
Title:	`Fixing compile-abc crash when parsing ABC v47.16`
Applies to:	iOS runtime component
Description:	Some IPA packaging commands had been causing a problem within the compile-abc tool (AOT compilation of ActionScript Byte Code into macho object code). This was caused by incorrect handling of the updated constant pool for ABC version 47.16, used for storing float objects.

Reference:	**AIR-7028**
Title:	`AIR Android file permission callbacks not always called`
Applies to:	Android runtime component

Description:	If a new File object was created, a permission event handler added, and then the “requestPermission()” method called, then the event handler did not always get the callback. This could happen when the File was just a local reference, and then went out of scope. Although we had been protecting against the mark-and-sweep clean-up of an in-progress event handler, this also needed additional reference counting.

Reference:	**AIR-7029**
Title:	`AIR Android applicationDirectory files may not be accessible`
Applies to:	Android runtime component
Description:	An APK file had been provide by a customer whereby it wasn’t possible to copy any files out from File.applicationDirectory subfolders, due to a lack of defensive programming that meant the APK file hosting the application was not properly parsed.

Reference:	**AIR-7030**
Title:	`ADT cannot use 50.2 xml descriptor elements when using 51.0 namespace`
Applies to:	Core build tools
Description:	The validator classes for application and extension descriptor files had been derived from 50.1 (due to when they were created); this meant that changes within 50.2 were not recognised as valid features/updates. The classes have been corrected to now extend the 50.2 versions.

Reference:	**Github-1917** https://github.com/airsdk/Adobe-Runtime-Support/issues/1917
Title:	`Ensuring FlashRuntimeExtensions.so Linux library works for arm64`
Applies to:	Linux runtime component
Description:	The ARM64 version of FlashRuntimeExtensions.so had not been distributed within the Linux SDK; the packaging scripts have been updated so that this is included and configured appropriately.

Reference:	**Github-2492** https://github.com/airsdk/Adobe-Runtime-Support/issues/2492

Title:	Updating Stage.displayState documentation to clarify it cannot be changed when handling FullScreenEvent
Applies to:	Documentation
Description:	Adding comments to clarify that if you're handling a FullScreenEvent due to changing the displayState, you cannot then change the displayState again.

Reference:	**Github-2610 https://github.com/airsdk/Adobe-Runtime-Support/issues/2610**
Title:	Ensuring Win32 timezone retrieval works for default tz when not dynamic
Applies to:	Windows runtime component
Description:	If the default timezone was requested on Windows, this only actually worked if the user's default timezone was "dynamic" i.e. had the potential to change the dates on which daylight savings occurs. The update checks for this situation and uses a secondary Win32 API if necessary to retrieve the current timezone information.

Reference:	**Github-2809 https://github.com/airsdk/Adobe-Runtime-Support/issues/2809**
Title:	Adding permission documentation for Geolocation class
Applies to:	Documentation
Description:	The documentation for the Geolocation class had no descriptions around the permissionStatus and requestPermission members. These have been added and a change submitted to airsdk.dev.

Reference:	**Github-3060 https://github.com/airsdk/Adobe-Runtime-Support/issues/3060**
Title:	AS3 StageTextContentType implementations for Android and iOS
Applies to:	Android and iOS runtime components
Description:	The 'StageTextContentType' value is now being converted into an appropriate platform-specific value and passed to the OS in order to provide hinting/keyboard updates as supported. This includes the capability to bring an SMS one-time passcode into the text field from the messaging app.

Reference:	**Github-3081 https://github.com/airsdk/Adobe-Runtime-Support/issues/3081**
Title:	Updating FlashRuntimeExtensions.h from source tree
Applies to:	All runtime components
Description:	The header file for FlashRuntimeExtensions had not been updated in the SDK folders; this is now available so that developers can use the new ANE methods.

Reference:	**Github-3083 https://github.com/airsdk/Adobe-Runtime-Support/issues/3083**
Title:	Updating descriptor-sample namespace to fix Animate bug
Applies to:	Core build tools
Description:	Animate had been stuck on the 50.2 namespace for application descriptor files, due to the way in which it takes the value from the “descriptor-sample” XML file in the AIR SDK. This file has now been updated.

Reference:	**Github-3084 https://github.com/airsdk/Adobe-Runtime-Support/issues/3084**
Title:	Updating airsdk.xml file for 51.0 namespace/swf version
Applies to:	Core build tools
Description:	The mapping in airsdk.xml had not been updated in the SDK folders, this has now been corrected.

Reference:	**Github-3086 https://github.com/airsdk/Adobe-Runtime-Support/issues/3086**
Title:	Updating swf version in the XML configuration and compiler to default 51
Applies to:	Core build tools
Description:	The air-config.xml configuration, and other confg files, have been updated to handle the 5.0 major/minor player version. The compiler has been updated to ensure it can handle this and future player version values and will base the SWF version on the major player version number.

Reference:	**Github-3087 https://github.com/airsdk/Adobe-Runtime-Support/issues/3087**
Title:	Correcting invalid scheme detection to prevent false-flagging of relative paths
Applies to:	All runtime components
Description:	An earlier update had caused a bug when trying to open a URL using a local relative file path. The code did not distinguish between relative paths and schemes, and thus rejected a relative path that started with a number.

3.5.3 Release 51.0.0.4

Reference:	**AIR-6743**
Title:	Update icon list in descriptor XSD
Applies to:	Core build tools
Description:	The list of permitted icon sizes was not correct in the XML schema definition, which can cause issues in some IDEs.

Reference:	**AIR-7031**
Title:	AIRSDK Android dependency lists
Applies to:	Android runtime component
Description:	A list of dependencies used by the Android runtime APK file has been revised/updated to remove out-dated and unnecessary dependencies.

Reference:	**AIR-7035**
Title:	String.fromCharCode() should support all unicode code points
Applies to:	All runtime components
Description:	The “String.fromCharCode()” method used to truncate any input value at 16 bits. This code has been updated to support higher code points 0x10000 and above, through the use of UTF-16 encoding with surrogate pairs.

Reference:	AIR-7036
Title:	Adding platform-specific fallback fonts for Emoji character ranges in Flash Text Engine
Applies to:	All runtime components
Description:	The Flash Text Engine code maintains a list of fallback fonts that are queried for glyphs in case the chosen FontDescription object does not contain a glyph for a particular Unicode code point. This has been extended to add support for the emoticon Unicode ranges with fallback fonts chosen based on each operating system's standard emoji font file.

Reference:	AIR-7059
Title:	Fixing AIR crash on iOS around network authentication (see AIR-6479)
Applies to:	iOS runtime component
Description:	Some HTTPS authentication/certificate verification handling was previously causing an instability on macOS, and had been fixed recently (under AIR-6479). This update brings the same fix into the iOS code.

Reference:	Github-1917: https://github.com/airsdk/Adobe-Runtime-Support/issues/1917
Title:	Correcting Linux bundle creation to support ARM64 ANEs
Applies to:	Linux runtime component
Description:	When a Linux bundle was created via ADT, that included an ANE, the wrong type of binaries were being extracted into the generated bundle. This fix ensures that the appropriate binaries for the operating system are used when building a bundle structure.

Reference:	Github-2807: https://github.com/airsdk/Adobe-Runtime-Support/issues/2807
Title:	Removing ANRs caused by access of nativeGetTextBoxBounds from wrong thread
Applies to:	Android runtime component

Description:	A number of ANRs were caused by hit testing that was checking for the touch point being within a text field – the problem coming when the runtime was in a background thread vs the touch request in the UI thread. This change protects against this scenario, at the expense of long-press events no longer working in text fields when the runtime is in a background thread.

Reference:	**Github-2871:** https://github.com/airsdk/Adobe-Runtime-Support/issues/2871
Title:	`Switching to a 64-bit version of the LLVM ld64 linker`
Applies to:	iOS runtime component
Description:	The Windows "ld64.exe" process was a 32-bit one but this had caused memory issues for some large applications when compiling the IPA files. Given that iOS executables are always 64-bit now, and the compile-abc executable is also 64-bit, we are switching this binary over to the 64-bit build of LLVM's macho linker, in order to resolve these memory issues.

Reference:	**Github-3098:** https://github.com/airsdk/Adobe-Runtime-Support/issues/3098
Title:	`Allowing a/v data access for NetStream in data generation mode`
Applies to:	All runtime components
Description:	Security errors were being thrown when trying to access a/v decoded data from a NetStream object that was in 'data generation' mode i.e. where the a/v stream was being passed in via NetStream.appendBytes(). However, given the fact that countless tools exist that could perform this same activity, this has been adjusted to allow AIR applications to also create such snapshots from the audio/video data.

Reference:	**Github-3102:** https://github.com/airsdk/Adobe-Runtime-Support/issues/3102
Title:	`Preventing crash when using Workers in a beta/prerelease build`
Applies to:	All runtime components
Description:	An instability was found when a Worker was launched in the 51.0 pre-release version. This is caused by the Worker attempting to display the "beta release" pop-up message, which should not be done from a background thread. Workers now will skip this step.

Reference:	**Github-3106:** https://github.com/airsdk/Adobe-Runtime-Support/issues/3106
Title:	`MacOS EncryptedLocalStore updating key storage mechanisms`
Applies to:	MacOS runtime components
Description:	The Encrypted Local Store functionality on macOS was missing an element in the new mechanisms to handle key storage (trying to avoid the need for any user passwords). This has now been correct and fully implemented.

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.