Adobe AIR SDK Release Notes

Version	50.0.0.1
Date	5 September 2022
Document ID	HCS19-000287
Owner	Andrew Frost

Table of contents

1 AIR 50.0 Major update

Release 50.0 of the AIR SDK is a major update that brings a number of changes that we hope will benefit the AIR developer community as well as helping to streamline the development and deployment of the AIR SDK. These release notes are being re-formatted to focus on the changes made in each release, with historical versions being available via the https://airsdk.harman.com website and other updates and guidance being provided under the https://airsdk.dev portal.

1.1 Why AIR 50.0

Adobe's final version of AIR was v32, so when HARMAN took over the maintenance, we moved to v33. For a while now we have used a base version number of 33.1 and just incremented the build – despite the fact that some of the changes have included updates to the XML descriptor format and the ActionScript APIs. Ideally such changes would result in new namespaces (33.1, 33.2, 33.3 etc) and new SWF versions (currently at 44).

With a major set of changes, the normal pattern may be to move to version 34 next – however, there has been some confusion and conflict with the Flash Player releases. Flash Player is still being supported in China and this was released as version 33 and then version 34; the SWF versions have also been incremented. To avoid any conflicts, Adobe and HARMAN had agreed that HARMAN's version of the Flash Player (supported for enterprise customers only) would be pushed to version 50 so that it was easily recognized as coming from HARMAN and not from Adobe or Microsoft.

So the move of AIR to version 50 is intended to match this version. The SWF version code is also being updated to 50 to match this. The initial intention was to keep these the same, as it is often confusing to have to refer back to tables of which SWF versions were associated with which AIR runtime versions, however we are hoping to continue adding new APIs to the ActionScript APIs and it is likely that the SWF version will increase faster than the major version number.

1.2 Component-based releases

One drawback we've found with the AIR SDK deployment method – a set of six monolithic zip files each hundreds of megabytes in size – is that even a tiny bugfix in a common tool that may be fairly urgent to deploy will end up with a lot of effort to build the SDK across the various platforms, go through the QA process and upload to the website. For developers there is then a lot of effort to pull down a new zip file, -extract it and set up all the tools again. The bandwidth being used is fairly significant and seems very wasteful when the bulk of the size is for example the iOS "stub" libraries that don't change until the iOS supported version is changed.

To address this, we are splitting the SDK into a number of smaller, platform-specific components. The "core" tools include the common (typically Java-based) tools plus the templates and other files that are present in all of the SDK variants. There are also some AIR-specific tools that are not needed/wanted if the SDK is for overlaying with a Flex SDK. Then each operating system would require a further package, so you would need one (desktop OS-specific) package for the local build tools and runtimes, and then Android and iOS support if building for those platforms.

If, for example, there is a minor bugfix needed in the ADT tool, this means that only that component will need to be updated. Developers therefore get a smaller fix and the effort and time involved to create this fix is greatly reduced.

To help developers manage this process, an "AIR SDK Manager" tool is being developed that will be able to download the components and collate them into the required SDK folder structure. The monolithic SDK zip files will also be available until the tool is available and widely adopted – we have been using this approach for the last few releases already.

1.3 Release numbering and update strategy

As part of these changes, HARMAN are planning to use a more strict approach to version numbering. The expectation is:

- Any change to the application descriptor format and supported tags will mean that the namespace will need to update. For example the namespace version would move from 50.0 to 50.1 if we add in a new build setting; the templates and XSD files will also be correctly updated.

- Any change to the ActionScript APIs will require an update to the SWF version; this also implies an update is required to the major/minor version number e.g. 50.1 to 50.2, since the runtimes check the SWF versions against the namespace versions for compatibility purposes.

- Any other functional update – i.e. adding/changing code at a feature level but where no change is needed in the ActionScript APIs or the XML descriptor file – will mean the third digit of the release code is changed, for example 50.0.0.1 to 50.0.1.1

- Any pure bug fix would just change the final revision number e.g. 50.0.0.1 to 50.0.0.2. These updates should always be applied and should have minimal risk of any new bugs; the AIR SDK Manager will find and offer to apply such updates to an existing AIR SDK installation.

1.4 Documentation

The release notes had previously included an ever-increasing historical list of changes, as well as a set of documentation around applying some of the settings and workarounds for building 64-ARM Android platforms within some of the IDEs. This information will be moved to the https://airsdk.dev portal so that it can be more easily referred to and found in searches.

The final part of the release notes had been a list of ActionScript 3 API changes. We have recently published the AIR 33.1 AS3 documentation online at https://airsdk.dev/reference/actionscript/3.0/ and so will be removing this final part of the release notes. A summary of changes will still be present in the release notes document but then the actual reference for new APIs should be found online.

2 Release Overview

Release 50.0.0.1 of the AIR SDK is a pre-release version – it should be treated as a beta release and should not be distributed to end users. There is a built-in timeout wherein the runtime binaries will stop working after three months.

Currently we suggest that developers continue to use the latest AIR 33.1 release for production applications, but would value any feedback from users of the 50.0 pre-release SDK. There are a number of new features that have been requested and we hope to resolve any outstanding critical errors swiftly and move to a production release as early as possible.

2.1 Key changes

Release 50.0 brings a fair number of updates, the key ones are mentioned here:

- New ActionScript “TimeZone” class and ability to set this as the ‘default’ timezone used in local time calculations
- Screen class updates for working out cut-out and ‘safe’ areas
- Enhancements and utility methods for ANEs
- Improvements in security with encryption of SWFs and binary data assets
- Support for multiple instances of AIR applications to be run concurrently
- Support for the macOS “Function” key modifier for menus
- Updates to build configuration mechanisms for application-specific control over settings

2.2 Limitations

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

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

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

2.3 Feedback

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

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

2.4 Notes

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

For developers who are packaging applications for desktop AIR, there is now a shared AIR runtime that is available for end users to download at https://airsdk.harman.com/runtime. However, we continue to recommend that applications are packaged up with the captive bundle mechanism to include the runtime and remove the dependency upon this shared package.

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

3 Release Information

3.1 Delivery Method

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

3.2 The Content of the Release

3.2.1 Detailed SW Content of the Release

Name	Version
Adobe AIR SDK – for Windows	50.0.0.1
Adobe AIR SDK – for Mac	50.0.0.1
Adobe AIR SDK – for Linux	50.0.0.1
Adobe AIR SDK for Flex Developers – for Windows	50.0.0.1
Adobe AIR SDK for Flex Developers – for Mac	50.0.0.1
Adobe AIR SDK for Flex Developers – for Linux	50.0.0.1

3.2.2 Delivered Documentation

Title	Document Number	Version
Adobe AIR SDK Release Notes	HCS19-000287	50.0.0.1

3.2.3 Build Environment

Platform	**Build Details**	
Android	Target SDK Version:	31
	Minimum SDK Version:	14 (ARMv7, x86); 21 (ARMv8, x86_64)
	Platform Tools:	28.0.3
	Build Tools:	31.0.0
	SDK Platform:	Android-31
	Note – these are the versions we use to build the AIR SDK and runtime, we also recommend developers match the same ‘target SDK’ version as here.	
iOS	iPhoneOS SDK Version:	15.0
	iPhoneSimulator SDK Version:	15.0
	XCode Version:	13.0
	Minimum iOS Target:	9.0
MacOS	MacOS SDK Version:	11.3
	XCode Version:	13.0
	Minimum macOS Target:	10.7
Windows	Visual Studio Version:	14.0.25431.01 Update 3

Linux	GCC Version	5.4.0 20160609 (Ubuntu 16.04.5)

3.3 AIR for Linux – Restrictions

The AIR SDK now supports some capabilities on Linux platforms. This is only available to developers with a commercial license to the SDK, and has some restrictions:

- No “shared runtime” support: applications would need to be built as ‘bundle’ packages with the captive runtimes
- Currently only x86_64 support – ARM64 is planned and potentially 32-bit variants if needed
- Packaging into native installers (“native” target type for .deb or .rpm files) is currently not working: please create a “bundle” target and use Linux tools to distribute these

The Linux functionality has not been as widely tested and is provided “as-is” – developers are free to distribute applications built using the SDK, and please report any issues found.

3.4 AIR for Flex users

HARMAN have continued Adobe’s strategy of issuing two AIR SDKs per platform: the first of these (“`AIRSDK_[os].zip`”) contains the newer ActionScript compiler and is a full, self-contained SDK for compiling and packaging AIR applications. The second of these is for combination with the Flex SDK (“`AIRSDK_Flex_[os].zip`”) which doesn’t include a number of the files necessary for ActionScript/MXML compilation. These SDKs should be extracted over the top of an existing, valid Flex SDK.

See instructions at https://helpx.adobe.com/uk/x-productkb/multi/how-overlay-air-sdk-flex-sdk.html.

NOTE when copying an AIR SDK over a previous version, there may be errors relating to “MainWindow.nib” and “MainWindow-iPad.nib”. These were originally files, and then had been turned into folders by a version of Xcode. However these should now be files again hence there may well be problems with overwriting of file types. If you see this error, the best approach is to delete these files/folders from the target location and then perform the copy/extraction again.

4 Summary of changes

4.1 Runtime and namespace version

Note: baseline for the 50.0.0.1 changes is the latest previous release, 33.1.1.929.

Namespace: 50**.0**

SWF version: **50**

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

4.2 Build Tools

AIR-6084: Updating templates and XSD documents for AIR 50.0 namespace

The templates deployed with the AIR SDK, that some IDEs use when generating new projects, have been updated to include stubs for all of the new application descriptor options that have been added into the 50.0 namespace. The XSD definition documents have also been updated so that tools such as IntelliJ IDEA which cross-check a descriptor against its definition will not throw warnings due to these new features.

github-1300: updating third party libraries used in the Android runtime

For Android builds, the expat library version has been updated to 2.4.7 and the libpng library version has been updated to 1.6.37.

No other changes have yet been made although updates to support the new Xcode and iOS versions will be made soon.

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

Xcode 13 and the latest macOS and iphoneOS SDKs are now being used to build the AIR SDK.

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

4.3 AS3 APIs

Updated APIs are listed below along with the feature/bug reference that appears in subsequent sections, containing more information about usage.

Ref	Change
github-1664	New method on `flash.display.BitmapData` to generate a `BitmapData` object from an image (GIF/JPEG/PNG): `public static native function decode(data:ByteArray):BitmapData;`
github-1894	Removal of unnecessary method on `flash.display.Camera`: `public native function setCursor(value:Boolean):void;`
AIR-6018	New class: `air.security.Digest`

AIR-5960	New method on `flash.external.ExtensionContext` to support dynamic loading of ANE definitions: `public native static function loadExtension(extensionID:String, aneFolder:File) : Boolean;`

AIR-5060	New property on `flash.ui.Keyboard` class for macOS Fn key handling:
	`public static const FUNCTION:uint;`
	Update to `flash.events.KeyboardEvent` constructor to include function key status (new argument '`functionKeyValue:Boolean=false`' added to the end).
	New property accessors on `flash.events.KeyboardEvent` class for function key status:
	`public native function get functionKey():Boolean;`
	`public native function set functionKey(value:Boolean):void;`

github-13	New read-only property on `flash.desktop.NativeApplication` class `public native function get isActive():Boolean;`

github-1425	New read-only property on `flash.display.Screen` class to get the scale factor set for a specific display.
	`public native function get contentsScaleFactor():Number;`
	New method on `flash.display.NativeWindow` class to resize this to cover a screen:
	`public function resizeToScreen(target : Screen) : void`

AIR-5796	New read-only property on `flash.display.Screen` class to access the 'safe' area of a display for mobile phones, taking account of the cut-outs and rounded corners. `public native function get safeArea():Rectangle;`
AIR-6116	New function on the `flash.display.Stage` class to allow a pre-load SWF to replace itself with a new class that has been dynamically loaded in to the runtime: `public native function preloadComplete(mainClass : String) : Boolean;`
AIR-5802	New method on the `flash.system.System` class to decrypted a '`DefineBinaryData`' tag from a SWF file that has been encrypted via the '`embed`' tag: `public static native function decryptBlob(data:Class):ByteArray;`
github-1572	New read-only property on the `flash.system.System` class to check if a loaded SWF has contained any debug instructions: `public static native function get containsDebugInfo():Boolean;`

github-2065	Accessors on the `flash.text.TextField` class to determine whether paragraph marks are treated more like HTML when set within the `htmlText` property:
	`public native function get htmlParagraphBehavior():Boolean;`
	`public native function set htmlParagraphBehavior(val:Boolean):void;`

AIR-5818	New built-in ActionScript class (at the top level) called `TimeZone`
	New accessory methods on the `Date` class to set/retrieve the current time zone:
	`public native static function get localTimeZone() : TimeZone;`
	`public native static function set localTimeZone(tz : TimeZone) : void;`

4.4 Features

AIR Multimedia APIs

A number of new ActionScript classes have been created to form a framework for a new multimedia pipeline. Currently this feature is still work in progress so we do not recommend any of these are used; this will be more fully documented as the implementations are developed across the various platforms.

AIR-5060: Add support for macOS 'function' key modifier

In order to support the “Fn” key on macOS, used in some menu keyboard accelerators (for example Fn-D for dictation), a new property “Function” has been added to the Keyboard class, and a corresponding constructor argument and property “functionKey” has been added to the KeyboardEvent class. Function key modifiers will then work properly with the recent macOS updates.

AIR-5077: Switching .nib folders back to individual files per xcode changes

This change switches the iOS files “MainWindow.nib” and “MainWindow-iPad.nib” back to just being the individual files, rather than the folders that an earlier version of Xcode had been creating. This is only relevant when copying and overlaying the AIR SDK on top of previous SDKs; the ADT tool will now cope with these regardless of whether they are files or folders.

AIR-5796: AIR API for providing cutout information or visible screen areas

To provide built-in support for the “safe” area of a screen – i.e. an area that isn’t obscured by camera cut-outs or rounded corners – a new Screen.safeArea property has been added that is similar to the visibleBounds value, but provides the area with this that is considered ‘safe’ by the operating system.

See also https://github.com/airsdk/Adobe-Runtime-Support/issues/933

AIR-5802: AIR runtime: DefineBinaryData support for encrypted data

The AIR AS3 compiler has been updated so that it can encrypt “BinaryData” SWF tags. These can be created via the “[Embed(…)]” directive, with a MIME type of “application/octet-stream”. To encrypt the data, a new parameter “encrypt="true"” should be used; then at runtime, the data can be retrieved using a new function “System.decryptBlob(data:Class):ByteArray” where the class definition (i.e. the name of the class that followed the Embed directive) is passed in and the decrypted data is returned as a byte array.

AIR-5818: AIR AS3 Timezone support

This change supports the concept of time zones that can be set manually onto a Date object and can be used to convert between UTC and specific timezones (rather than just between UTC and whatever the device local timezone is). A new top level TimeZone class has been added to AS3, and a localTimeZone property added to the Date class so that the “local” (non-UTC) times can be queried for this time zone rather than just the local time.

Note that the list of supported time zones is retrieved dynamically on each platform and will vary between different operating systems.

See also https://github.com/airsdk/Adobe-Runtime-Support/discussions/1396

AIR-5960: Adding delayed-load capability for ANEs

To avoid having to load and initialise everything within the first frame of execution, ANEs can now be loaded in a delayed fashion. To set this up, add a “`delayLoad="true"`” attribute into the “extensionID” tag, and then – prior to accessing any of the AS3 definitions from the ANE’s library – use the new ExtensionContext.loadExtension() method. This method can also be used to load an ANE that’s not listed in the application descriptor file.

AIR-5991: Compiler and configuration files switching to 50.0 release with default as air-config

This updates the default SWF version to 50 when building a SWF using the compiler provided within the full AIR SDK; it also means that the configuration file “air-config.xml” will be picked up by default,

rather than the “flex-config.xml” file. This can be adjusted via the “-load-config” command if a different file is required.

AIR-6015: Moving standard configuration settings into user folder

The “adt.cfg” file had been created initially to work around restrictions in the IDEs where new features available in the ADT command line were not reflected in the IDEs such as Animate, Flash Builder etc. The file was located in the “lib” folder for the AIR SDK (i.e. alongside the “adt.jar” file) but this needed setting up each time a new SDK was installed. The settings have now been moved to a user-specific folder (“~/.airsdk/adt.cfg”) so that a single file covers all SDK installations. Note that if a particular setting is not found in this user file, the ADT software still looks for a setting in the SDK’s “lib” folder, so this can be used for customisations/exceptions that may be SDK-specific.

AIR-6018: AIR security - digest capability

A utility method has been added to the ActionScript APIs in order to generate a digest/checksum of some data. A new package “air.security” defines a class “Digest” that can be passed data via the IDataInput interface, and then returns a digest (SHA-1, SHA-256 or MD5) as a ByteArray. There is also a utility method to perform a digest in a single function call that takes the algorithm plus an input ByteArray.

AIR-6057: App descriptor additions for Android gradle build settings

To allow developers to tweak the settings used in the Android gradle build mechanism, new application descriptor values are available under the “<android>” section. These are “<multidexVersion>”, “<gradleVersion>” and “<androidGradlePluginVersion>”; version codes should be provided as the standard x.y.z notation. This is likely to be required only when customising the application and build process with ANEs that have particular requirements, or perhaps where build systems need to be maintained with different defaults and tools than the AIR SDK is expecting.

AIR-6106: App descriptor files to include more Android build settings

In a similar vein, some of the configuration settings that had been present in the “adt.cfg” file are now available within the application descriptor file. These values have the highest priority and will override any setting in an adt.cfg file, and can be used for application-specific overrides to any default values. The available options – that should be contained within the “<android>” section of the application descriptor – are: “<addAirToAppID>” (true/false), “<buildArchitectures>” (comma-separated list of architectures armv7/armv8/x86/x64), “<createAppBundle>” (true/false), “<buildLegacyAPK>” (just a capitalisation change), and “<uncompressedExtensions>” (comma-separated list of extensions). The “buildArchitectures” are used in two instances: (a) when generating an Android App Bundle in order to limit which AIR runtime libraries are included; (b) when generating an APK file, if only one option is listed here, this will be used as an override to any command-line request provided by the IDE.

AIR-6116: Adding stage.preloadComplete() method to load a new root display object

This method is intended to help with the use of preloaders to ensure that ActionScript definitions loaded in via secondary SWF files (or in later frames in a moveiclip-based SWF) can be appropriately instantiated and added to the stage. If the preloader calls “preloadComplete” on its stage object, passing in the fully qualified name of the ActionScript class for the main application, then AIR will reset itself as if it was being started up again and the new class will be created with an initial display object already on the stage. The original preloader object will be removed completely from the display list and will be cleaned up via standard garbage collection rules.

github-13: Adding NativeApplication.isActive property

This property has been added so that applications can check whether they have the operating system ‘focus’; it is updated immediately prior to dispatching the “activate” and “deactivate” events.

github-175: Adding new 'FRENewByteArray' ANE method for creating ByteArray AS3 objects

This method is a short-cut/utility to simplify the creation of ByteArray objects from native C/C++ code in an ANE. An FREByteArray object can be passed in and an FREObject handle to the new

ActionScript object is returned. Note that this function will copy the input data so the user remains responsible for cleaning up the original data that is passed in.

github-236: Adding FREAcquire/ReleaseNativeWindowHandle methods for ANEs

In order to support advanced functionality, it is now possible to get the native window handle from the operating system, that is the equivalent of the NativeWindow object. Note that this is currently only supported on Windows but will be expanded to other desktop platforms.

github-494: Incorporating support for encrypted SWF formats

This is a mechanism to support SWF files where the contents of the file have been encrypted (using a new tool that will be provided as part of the AIR SDK, ultimately this is planned to be an option for the new ActionScript compiler as well). Note that the SWF is decrypted in memory and so it may still be possible for users to access the decrypted SWF data if they can do a memory dump of a process.

github-1305: Allowing multiple instances of an AIR application via XML app descriptor flag

A new application descriptor setting, “<allowMultipleInstances>”, has been added at the top level (under the “<application>” section). This will mean that desktop applications will by default have their standard behaviour for running multiple processes – i.e. a new instance of the application will be spawned. It will then be possible to have several versions of the same application running simultaneously; note that there may be implications for LocalConnection settings, although we believe that file and resource access is all synchronised so there should not be any problem with multiple application instances reading/writing to the same file simultaneously (e.g. ELS, local storage etc).

github-1425: Adding scaling information for windows and NativeWindow.resizeToScreen() method

The Screen class has a new property called “contentsScaleFactor” since this value can be different for different physical monitors. A utility function has also been added so that a NativeWindow object can be automatically resized to fit the full area of the Screen object that is passed in.

github-1572: Adding System.containsDebugInfo property to identify SWF files that contain debugfile instructions

A utility method has been added so that an application can tell at runtime whether it was built with debug instructions included. Note that this property is set to true as soon as any debug instruction is found in a function that the runtime has processed/entered, which means that any loaded SWF files may also affect this value.

github-1664: Synchronous decoding for png/jpeg/gif images via BitmapData.decode method

A new mechanism has been added to BitmapData objects so that they can be created from PNG, JPEG and GIF objects. This can then be used synchronously although developers should be aware of any delays in how long this operation may take, vs the use of the Loader object to asynchronously load and decode an image.

github-1873: Adding FREGetFREContextFromExtensionContext method to C ANE interface

This utility method allows the C/C++ implementation of an ANE to retrieve an FREContext handle based on an ExtensionContext object handle This could be used for communication between ANEs. Note however that the handle may become invalidated inbetween different ANE calls, so this handle should not be cached between different calls out to an ANE.

github-1966: correct handling of XML signature validation and publisher information

This change means that AIR applications installed from appropriately signed .air packages will be correctly shown as “trusted” and the publisher information will be shown as verified. This change was also pulled into a recent runtime-only update, AIR 33.1.1.744 (with webkit) and AIR 33.1.1.932, which do not have a corresponding SDK release.

github-1979: Adding windows.clipboardFullHTML flag to obtain full HTML code from the clipboard

This is an application descriptor update where a new tag, “<clipboardFullHTML>” (true/false) can be set under the “<windows>” section, that will change the behaviour of the clipboard operations when

retrieving HTML text on Windows. Previously (and the default still, or if the flag is false) the “fragment” of HTML had been provided to the AIR app rather than the full data that was present on the clipboard. Setting this to true will result in the full Windows clipboard HTML information being provided, including all style settings, per the Windows clipboard documentation.

github-2065: Adding TextField.htmlParagraphBehavior property to fix newline behaviour in htmlText

This setting can be used to change the behaviour of paragraph marks (…) within the htmlText property of a TextField. If this is set to true, the behaviour is updated so that it becomes more like that of normal HTML – where both the start and end of a paragraph are marked with newline characters. To ensure the behaviour seems sensible, any newline at the start or end of the text field will be ignored – hence this setting can be used to avoid the behavioural differences if a text field just has a single line that has .. formatting, where in ‘input’ mode there will be a newline at the end but in ‘dynamic’ mode this is removed.

4.5 Bug Fixes

AIR-6005: ADL should check for a license in the user folder first

This fix should prevent the issues that are being seen due to AIR SDK license files being present both in the AIR SDK “lib” folder and in the user-specific “~/.airsdk” folder. ADT updates the latter of these but ADL had been checking first for a file in the SDK “lib” folder which could then be out of date, resulting in a splash screen being shown when debugging but not when an application was packaged.

github-1274: System.output needs to appear on the active console when packaged as an .exe file

The output method had been working only when running via ADL, but when an application was generated via ADT the console connection no longer worked. This has now been rectified so that command-line applications can be converted into standalone .exe files on Windows.

github-1894: Removing unnecessary Camera.setCursor function

This method was no longer used/supported so the definition has been removed to avoid confusion.

github-2025: Ensuring AOT compilation copes with switch statement optimisations

Previously, an empty case option within a switch statement could cause problems with the AOT compiler due to an optimisation that condensed cases that were the same.

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>
```

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.

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

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.

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.