Adobe AIR SDK Release Notes

Version	50.2.3.2
Date	10 July 2023
Document ID	HCS19-000287
Owner	Andrew Frost

Table of contents

1 Release Overview

Release 50.2.3.1 of the AIR SDK is a feature update – a number of new features have been added and bugs have been fixed, although nothing that requires a change to the AIR namespace values or SWF version codes.

Release 50.2.3.2 has been provided due to an issue with the Linux runtime in 50.2.3.1. This had been built with a new/updated build environment, but resulted in a strange instability. Having reverted back to the older build environment, this instability appears to be resolved. Further investigations will be needed to determine the root cause but for now, we will continue to use the older build environment for Linux.

Note that this update is only for Linux, other platforms will continue to show as version 50.2.3.1.

1.1 Key changes

The build platforms have been updated, with Android API levels now set with compile and target API levels of 33. The Android gradle plug-in version has also been updated, with some other compatibility changes so that developers can use 7.x or 8.x if required.

A new 'certificateError' event has been created that developers can use to handle errors when connecting a SecureSocket or an URLStream/Loader object to a remote server. This gives the application the chance to override the default behaviour regarding SSL failures caused by self-signed certificates and similar issues and to 'mute' the dialogs that could otherwise be shown to the user. Note that currently this has been implemented without any AS3 API updates, however in AIR 51 the necessary changes will be made to provide the event name within the SecurityEvent class. Feedback on this feature is welcomed, please use https://github.com/airsdk/Adobe-Runtime-Support/issues/1453

The new media classes continue to be slowly implemented with support now for audio output of files/urls on iOS (as well as a/v on Windows output to a NativeWindow). For an example of using this on Windows, see https://github.com/airsdk/Adobe-Runtime-Support/issues/2503#issuecomment-1453906011. An example for iOS will be added under https://github.com/airsdk/Adobe-Runtime-Support/issues/2642. The iOS support is intended to work around the problems seen on iPhone 14 devices where screen brightness controls and ambient lighting conditions can have the effect of adding glitches into audio playback using the SoundChannel mechanism.

There are also a number of minor changes and bug fixes. For details please see section 3.

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.

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

1.4 Feedback

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

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

1.5 Notes

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

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

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

2 Release Information

2.1 Delivery Method

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

The update will also be available via the AIR SDK Manager. The latest version of this can be downloaded from https://github.com/airsdk/airsdkmanager-releases/releases.

2.2 The Content of the Release

2.2.1 Detailed SW Content of the Release

Component Name	**50.2.3.1**	**50.2.3.2**
Core Tools	2.5.0	
AIR Tools	2.0.1	
Windows platform package	2.5.0	
MacOS platform package	2.5.0	
Linux platform package	2.5.0	2.5.1
Android platform package	2.5.0	
iPhone platform package	2.5.0	

2.2.2 Delivered Documentation

Title	Document Number	Version
Adobe AIR SDK Release Notes	HCS19-000287	50.2.3

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:	16.4
	iPhoneSimulator SDK Version:	16.4
	XCode Version:	14.3
	Minimum iOS Target:	11.0

tvOS	tvOS SDK Version:	16.4
	tvSimulator SDK Version:	16.4
	XCode Version:	14.3
	Minimum tvOS Target:	11.0
MacOS	MacOS SDK Version:	13.3
	XCode Version:	14.3
	Minimum macOS Target:	10.15
Windows	Visual Studio Version:	14.0.25431.01 Update 3
Linux	GCC Version	5.4.0 (Ubuntu 16.04.1)

2.3 AIR for Linux – Restrictions

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

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

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

2.4 AIR for Flex users

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

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

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

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

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

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

3 Summary of changes

3.1 Runtime and namespace version

Namespace: 50.2

SWF version: 50

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

3.2 Build Tools

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

Xcode 14.3 and the latest macOS and iphoneOS/tvOS SDKs are now being used to build the AIR SDK. Please note when the update was made to use Xcode 14.x, the minimum iOS/tvOS target version was increased to 11.

The Linux build machine has been updated to Ubuntu 20 with a corresponding update to GCC.

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

3.3 AS3 APIs

No changes

3.4 Features

Reference:	**AIR-6564**
Title:	`AIR Media - basic iOS sound output implementation`
Applies to:	iOS runtime component
Description:	To help work around a problem with the latest iPhones where audio output is glitching, initial support has been added to play back sounds via the new AIR media APIs. Example code will be added into the below Github issue: https://github.com/airsdk/Adobe-Runtime-Support/issues/2642 Currently this is just supported for URLs/file paths, but support for binary data (embedded MP3s etc) will be added later as will capabilities around volume and completion events, looping etc.

Reference:	**Github-1453** https://github.com/airsdk/Adobe-Runtime-Support/issues/1453
Title:	`Adding certificateError event for secure HTTP/socket connections`
Applies to:	All runtime components
Description:	If a connection to an HTTPS server, or to a secure socket, indicates that there is a problem with the server's certificate, previously the behaviour was variable: typically for sockets, the connection was abandoned, and for HTTPS connections, a system dialog was shown to the end user offering them the option to cancel or continue. This feature adds a new event type, "certificateError", that is dispatched when the runtime encounters such a scenario. If there are no listeners for the error then the behaviour will continue as it has done in the past. However, if there is a listener that's added, the default behaviour is then to cancel the connection unless the handler calls the "preventDefault()" method on the error object. Note that this feature can be considered 'experimental' and feedback would be welcomed. There may be some inconsistencies between platforms due to the different ways in which the connectivity is handled, and currently there is no information provided about the reasons or the certificate. Further details will hopefully be available in a future release of AIR where a custom event could be added, potentially also with the ability for the event listener to defer the decision to the end user. Currently the response needs to be immediate (i.e. when the event handler exists, the decision has to be made whether to cancel (default) or continue (preventDefault) the connection). If the application wants to confirm with the user, there are two choices: a) Start presenting the user with the option in the application; meanwhile the connection will be cancelled. If the user selects to continue, the connection can be re-requested and the user's response used to call 'preventDefault' in the event handler. b) The handler could leave the default to cancel the existing connection but then remove itself as an event handler and re-request the connection – this means that the original/default behaviour would occur, i.e. asking the user (for https connections).

3.5 Bug Fixes

3.5.1 Release 50.2.3.1

Reference:	**AIR-4357**

Title:	Removing deferred framebuffer clears for Android runtime in background thread
Applies to:	Android runtime component
Description:	A number of instabilities had been found when using the Android runtime in a background there, which were within the “glClear” function. Some logic had been in place to defer these calls to immediately prior to when they were required, which may have been causing the instability, so this feature has been reverted.

Reference:	**Github-1824** https://github.com/airsdk/Adobe-Runtime-Support/issues/1824
Title:	Ensuring AIR apps can run from the root folder of a Windows drive
Applies to:	Windows runtime component
Description:	A logical error in handling relative path names had prevented AIR applications from working properly if they were based at the root of any Windows drive.

Reference:	**Github-1856** https://github.com/airsdk/Adobe-Runtime-Support/issues/1856
Title:	Fixing URL session closure on macOS for cancelled connections
Applies to:	MacOS runtime component
Description:	When multiple URL requests were made on macOS but were then cancelled before they were completed, the resources were not being correctly released in the operating system and after a while this meant that no new connections were being set up.

Reference:	**Github-1871** https://github.com/airsdk/Adobe-Runtime-Support/issues/1871
Title:	Further updates to support File.openWithDefaultApplication on Android
Applies to:	Android runtime component
Description:	The opening of files using the Android default application has been updated so that this also handles content URLs and doesn’t require additional permissions to be included.

Reference:	**Github-2409** https://github.com/airsdk/Adobe-Runtime-Support/issues/2409
Title:	`Fixing tvOS stub generation and reverting symbol removals`
Applies to:	tvOS runtime component
Description:	The earlier issues relating to symbols not being found has now been resolved properly – there was a logic issue in the 'stub' generation where the AppleTVOS SDK files were processed into the library stubs, which had caused these symbols to be identified as coming from the wrong framework. This has been corrected and the earlier changes have been reverted to re-include the use of those symbols.

Reference:	**Github-2535** https://github.com/airsdk/Adobe-Runtime-Support/issues/2535
Title:	`Don't Activate on _NET_WM_STATE event if the window is being hidden`
Applies to:	Linux runtime component
Description:	When a Linux window was being hidden, a state event was being received that caused the window to be restored again. Additional code logic has been added to detect this condition and ensure windows are not immediately restored when they are minimised.

Reference:	**Github-2603** https://github.com/airsdk/Adobe-Runtime-Support/issues/2603
Title:	`Ensuring Android file chooser ignores non-mime type filters`
Applies to:	Android runtime component
Description:	With the changes to use the Storage Access Framework, the 'File.browseForOpen' method's file filter now needs to be passed a MIME type rather than a file extension. This had resulted in a problem when an invalid filter was passed in (or the default, which resulted in a "*" filter) with Android then throwing an exception. Any filter strings that do not match the expected MIME string format are now being ignored.

Reference:	**Github-2615** https://github.com/airsdk/Adobe-Runtime-Support/issues/2615
Title:	`Updating Android StageText to work in a background thread`
Applies to:	Android runtime component

Description:	With the AIR runtime in a background thread, some functionality that caused view updates was then throwing exceptions because it must run in the UI thread. This change brings the StageText implementation into line with this, where the capabilities are updated to have the adding/removing from the stage now happening asynchronously on the UI thread.

Reference:	**Github-2655** https://github.com/airsdk/Adobe-Runtime-Support/issues/2655
Title:	`Fixing the iOS certificate security alert message by moving it out from async thread`
Applies to:	iOS runtime component
Description:	When the iOS runtime showed a security certificate error, the message being displayed could be incorrect as the message was being picked up from the underlying OS which may have had further activities between when the error occurred vs when the message was presented. The message is now being cached when it happens in the background thread, and this cached string is used in the error dialog that is running on the UI thread.

Reference:	**Github-2660** https://github.com/airsdk/Adobe-Runtime-Support/issues/2660
Title:	`Ensuring Android platformsdk is picked up properly on cmdline`
Applies to:	Core build tools
Description:	When building for Android, if no Android platform SDK had been provided in a config file (or found from the default expected path), the command-line's "platformsdk" option should have been checked. A code logic error had prevented this from being properly picked up.

Reference:	**Github-2665** https://github.com/airsdk/Adobe-Runtime-Support/issues/2665
Title:	`Removing memory leakage in Worker when sending strings over MessageChannel`
Applies to:	All runtime components
Description:	Due to earlier changes, an issue had arisen where a string sent between Workers was resulting in a memory leak with the receiving worker not then triggering the appropriate garbage collection. This has been updated so that the strings are collected properly and garbage collection – and Scout metrics – are all reported correctly.

Reference:	**Github-2666** https://github.com/airsdk/Adobe-Runtime-Support/issues/2666

Title:	`Ensuring android CameraUI provider is properly named with air prefix`
Applies to:	Android runtime component
Description:	The CameraUI provider entry that is generated in Android manifest files was always including the “air.” prefix, even when the developer options had been set to not include this in the Android application ID/namespace. This has been updated so it uses the same logic as the application ID.

Reference:	**Github-2667** https://github.com/airsdk/Adobe-Runtime-Support/issues/2667
Title:	`Fixing JNI problems with Android TimeZone.availableTimeZoneNames`
Applies to:	Android runtime component
Description:	The retrieval of time zone names from Java was causing a problem on older devices due to the limitations of local JNI variables. The code logic has been updated to handle these in batches and release the memory between each batch.

Reference:	**Github-2670** https://github.com/airsdk/Adobe-Runtime-Support/issues/2670
Title:	`Ensuring AIR on Android shuts down appropriately on exit() call`
Applies to:	Android runtime component
Description:	When a call to “exit()” had occurred, there were some asynchronous activities that resulted in the process being closed prior to callbacks being triggered by the operating system. This resulted in exceptions being thrown; the logic has been updated so that the application waits for the Android lifecycle to be shut down before the main AIR process is terminated.

Reference:	**Github-2671** https://github.com/airsdk/Adobe-Runtime-Support/issues/2671
Title:	`Preventing Android JNI-detach crash`
Applies to:	Android runtime component
Description:	Some of the handling to run the Android runtime in a background thread meant that other Android threads, when calling into the runtime via JNI, were then being attached and detached. However, some of these threads had already been attached by the operating system hence calling detach was then causing problems. Logic to detect whether threads are already attached has now been added.

Reference:	**Github-2684 https://github.com/airsdk/Adobe-Runtime-Support/issues/2684**
Title:	`Ensuring command-line platformsdk has priority in ADT`
Applies to:	Core build tools
Description:	Previously, the platform SDK provided in a config file (or auto-detected) was being used as the preferred value when running an Android APK/AAB build. This logic change has been made so that priority is instead given to any command-line value that is passed in.

Reference:	**Github-2694 https://github.com/airsdk/Adobe-Runtime-Support/issues/2694**
Title:	`Excluding invalid libc++.so files from Gradle builds`
Applies to:	Android runtime component
Description:	The “libc++.so” files that were being provided for Gradle builds are now being excluded: they weren’t actually binary/library files, but instead just configuration files that had been used by earlier build mechanisms. They are no longer required and had been causing errors when Gradle updated a “strip” warning to an error.

3.5.2 Release 50.2.3.2

Reference:	**Github-2712 https://github.com/airsdk/Adobe-Runtime-Support/issues/2712**
Title:	`Linux runtime rebuild using earlier GCC version`
Applies to:	Linux runtime component
Description:	The recent build using GCC 9.4 had been causing an instability; to resolve this, the earlier build environment with an older GCC version is being used.

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.