Adobe AIR SDK Release Notes

Version	33.0.2.246
Date	5 September 2019
Document ID	HCS19-000287
Owner	Andrew Frost

Table of contents

1 Purpose of the Release 3

2 Release Information 4
2.1 Delivery Method 4
2.2 The Content of the Release 4
2.3 AIR for Flex users 4

3 Changes and Issues 5
3.1 Changes in this Release 5
3.2 Known Problems 5
3.3 Previous Changes 6

4 Updating tools/IDEs to support 64-bit ARM 8
4.1 AIR Developer Tool 8
4.2 ADT Architecture Configuration 8
4.3 Flash Builder 8
4.4 Adobe Animate 8
4.5 FlashDevelop 9
4.6 Moonshine 9
4.7 IntelliJ IDEA 9
4.8 FDT 10
4.9 Visual Studio Code 10

5 Configuration File 11

6 Android Applications – Play Store Uploads 12

1 Purpose of the Release

This is an official release of the Adobe AIR SDK software, provided by HARMAN under the terms of the “AIR SDK License Agreement”. This software can be used to package, deploy and distribute Android applications to end users.

The SDK is not yet updated to support iOS, macOS or Windows platforms, so should only be used for Android applications. It supports 32-bit and 64-bit ARM platforms, and 32-bit x86 platforms, and will create a separate APK for each of the different architectures.

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

Build 246 provides some additional functionality and some bug fixes, details can be found in section 3.1; it also provides HARMAN with the metrics and licensing information as per the Privacy section of the AIR SDK License Agreement. There are a number of open issues that are outstanding, some of which are urgent and will be worked on as a priority (multidex failure on old Android builds, and a JIT issue with aarch64 code generation), and we will also be working on the support for other platforms.

The new website for AIR SDK is now live: https://airsdk.harman.com – this is very basic initially but will be updated over the coming weeks to include an FAQ, support information, pricing/purchasing details, etc.

2 Release Information

2.1 Delivery Method

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

2.2 The Content of the Release

2.2.1 Detailed SW Content of the Release

Name	Version	Size
Adobe AIR SDK – for Windows	33.0.2.246	439,925 kb
Adobe AIR SDK – for Mac	33.0.2.246	407,267 kb
Adobe AIR SDK for Flex Developers – for Windows	33.0.2.246	403,750 kb
Adobe AIR SDK for Flex Developers – for Mac	33.0.2.246	371,092 kb

2.2.2 Delivered Documentation

Title	Document Number	Version
Adobe AIR SDK Release Notes	HCS19-000287	33.0.2.246

2.2.3 Build Environment

Platform	**Build Details**	
Android	Target SDK Version:	28
	Minimum SDK Version:	14 (ARMv7, x86); 21 (ARMv8)
	Platform Tools:	28.0.3
	Build Tools:	28.0.3
	SDK Tools:	26.1.1
	SDK Platform:	Android-28

2.3 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.

3 Changes and Issues

3.1 Changes in this Release

3.1.1 Runtime

Namespace: 32.0

SWF version: 43

The default namespace and SWF version are the same as those used by the AIR SDK v32.0, so that applications that are generated using these defaults are able to be run on the desktop tools (e.g. via the AIR Debug Launcher) which have not yet been modified by HARMAN.

3.1.2 Build Tools

Updated smali and baksmali libraries in order to support Java 8 features in Android SDK level 26+

Please note the reqiurement for Java 8 due to the dependency upon Android build tools. Flash Builder users need to update the JRE embedded within this tool, see 4.3.

3.1.3 Features

AIR-198: Add ability to control whether ADT prepends ”air.” to the Android Application ID

AIR-200: Analytics feature to provide information on platforms/tools used when packaging apps

AIR-201: Licensing feature to periodically confirm validity and update the license file

3.1.4 Bug Fixes

AIR-196: Generated license certificate files can be malformed

AIR-203: Drawing a video before Netstream starts to play causes a crash (Gamua-98)

AIR-204: Read “position” property of the async opened FileStream causes hang (Gamua-97)

3.2 Known Problems

AIR-168: AIR content goes all white/blank after AR camera closes (Gamua-67)

AIR-182: Wrong resolution reported on S10 handsets

AIR-183: ANRs caused by inputs not finishing processing in time (Gamua-29)

AIR-197: Youtube videos only play audio when running in Android Webview

AIR-199: AIR 33 doesn’t support Java 8 features for Android extensions

AIR-205: Multidex support fails when running on older Android version

AIR-206: Crash with invalid opcode – JIT issue with aarch64 generation

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

3.3 Previous Changes

3.3.1 AIR 33.0.1.228

AIR-190: AIR SDK scripts on MacOS don't cope with the SDK path containing a space

AIR-192: Black screen when starting AIR (free tier) on older Android versions

3.3.2 AIR 33.0.1.220

AIR-112, Gamua-58: Update ADT so that it doesn't compress certain file types (see 'UncompressedExtensions' config file setting in section 5)

AIR-181: Android 'back' button cannot be handled in ActionScript (Gamua-73)

AIR-184: Camera is not working with ARMv8 binary (Gamua-72)

AIR-186: Camera hangs when video.attachCamera(null) is called in frame handler (Gamua-54)

3.3.3 AIR 33.0.0.212

HARMAN Ref AIR-159: Soft keyboard not appearing when an input text field has focus

HARMAN Ref AIR-160: Config file doesn't take effect unless "DebugOut" setting is present

HARMAN Ref AIR-161: ADT packaging of ANEs doesn't handle the use of a config file to override the architecture

3.3.4 AIR 33.0.0.182

ADOBE Ref AIR-4198749: AIR crashes on latest Anrdoid Q Preview

HARMAN Ref AIR-144: Performance hit on 64-bit ARM Android runtime

HARMAN Ref AIR-149: AIR SDK cannot package an app with google_play_services included in the manifest

HARMAN Ref AIR-153: Swf-Version built from Adobe Animate is set to 44 and does not work with ADL

HARMAN Ref AIR-156: ADT copyright output is affecting IDEA integration

HARMAN Ref AIR-157: Cannot export release build from FB on second attempt

HARMAN Ref AIR-158: AIR SDK package failures due to incorrect target SDK version

GAMUA Ref #55: Including Support-v4 28.0.0.ANE results in compile error

3.3.5 AIR 33.0.0.175

HARMAN Ref AIR-138: ADT shouldn't compress raw/binary files when creating APK

HARMAN Ref AIR-139: ADT needs a mechanism to set the default target architecture

HARMAN Ref AIR-140: Building with new airglobal.swc file fails

HARMAN Ref AIR-142: air-sdk-description.xml isn't updated

HARMAN Ref AIR-143: ADT -version should only print the version and not the copyright notice

HARMAN Ref AIR-145: Crash in AIR runtime on ARMv7 builds

HARMAN Ref AIR-146: ADT should use "armv8" for consistency

HARMAN Ref AIR-151: ADT doesn't work with Java 8: dx tool failed

3.3.6 AIR 33.0.0.168

Adobe Ref AIR-4198789: 64-bit ARM support for Android.

Adobe Ref AIR-4198749: Text relocations on Android Q

HARMAN Ref AIR-82: System.Capabilities.supports64BitProcesses incorrect with 64-bit AIR builds

HARMAN Ref AIR-96: Remove reliance on deprecated “MODE_WORLD_READABLE” flag

4 Updating tools/IDEs to support 64-bit ARM

4.1 AIR Developer Tool

To package an android application with the armv8 binary, the "-arch armv8" option must be used on the ADT command line. By default, the packager will use armv7 unless a configuration file is provided – see below.

4.2 ADT Architecture Configuration

The default architecture used by ADT can be adjusted via the configuration file as described in section 5.

For example, to ensure that the packages created by ADT will always embed the 64-bit runtime, the configuration file should contain:

```
DefaultArch=armv8
OverrideArch=armv8
```

Using this configuration file, a developer can package their applications for ARMv8 targets using existing versions of Adobe Animate, FDT etc.

4.3 Flash Builder

The new AIR SDK should be updated using standard instructions found on Adobe's forums:

https://helpx.adobe.com/uk/flash-builder/kb/overlay-air-sdk-flash-builder.html

or for updating the Flex SDK: https://helpx.adobe.com/uk/x-productkb/multi/how-overlay-air-sdk-flex-sdk.html

If you find an issue with the AS3 not compiling, this can be addressed by https://forums.adobe.com/thread/1483159

Exporting a release build must be set to use the captive runtime.

To update the architecture, open the Project Properties and expand the ActionScript Build Packaging item to select "Google Android"

Click on "Customize Launch", "Add Parameter..." and give a name of "-arch" and value "armv8". Place this after the "-target" option.

Please note that AIR SDK now requires Java version 8, in line with Google's requirements for the latest Android build tools, and that Flash Builder's internal JRE needs to be updated accordingly: please see http://blogs.adobe.com/flashplayer/2018/02/running-adobe-flash-builder-on-mac-with-java-78.html

Also please note an issue which may cause problems when adding "-arch armv8" (or "-arch x86") to the launch parameters: https://forums.adobe.com/thread/1505072

4.4 Adobe Animate

To add support for the new AIR SDK, use the "Help | Manage Adobe AIR SDK…" option from Animate. Click on the "+" icon and select the folder into which you have extracted the SDK. This should show in the list of SDKs with the correct version number.

To select the target architecture, if this is not ARMv7 or x86 then please use the configuration file mechanism described above.

4.5 FlashDevelop

The packaging script asks the user which option to use for creating a mobile package (Android/iOS etc) but there is no way currently in this to specify an architecture (even for x86).

An extra section can be added to the "Packager.bat" script that will allow the user to be queried on the target ABI to be used in the package. The "Packager.bat" script can then be provided into FlashDevelop's project folder so that this is used for all new projects:

FlashDevelop\Projects\190 ActionScript 3 - AIR Mobile AS3 App\bat\Packager.bat

The extra choice needs to be added within the "android-config" section, prior to the "goto start" command:

```
:: which architecture?
echo.
echo Please select your target architecture for Android:
echo.
echo  [1] armv7       ^(32-bit ARM devices^)
echo  [2] x86         ^(Intel products^)
echo  [3] armv8       ^(64-bit ARM devices^)
echo.
set /P ARCH=[Choice:]
echo.
if "%ARCH%"=="1" set OPTIONS=%OPTIONS% -arch armv7
if "%ARCH%"=="2" set OPTIONS=%OPTIONS% -arch x86
if "%ARCH%"=="3" set OPTIONS=%OPTIONS% -arch armv8
```

4.6 Moonshine

Moonshine has a build.xml file which is used to call the ADT packaging tool:

```
<target name="compileAPKProject" depends="compileSWF">
      <java jar="${ADT_PATH}" fork="true" failonerror="true">
            <arg line="-package" />
            <arg line="-target apk-captive-runtime"/>
            ..
            <arg line="${SWF_FILE_PATH}" />
            <!-- Add folders to be bundled in the AIR file here -->
      </java>
</target>
```

An additional 'arg' can be added in order to select the ABI:

```
            <arg line="-arch armv8" />
```

4.7 IntelliJ IDEA

The new SDK should be incorporated into IntelliJ IDEA using the standard process documented at: https://www.jetbrains.com/help/idea/preparing-for-actionscript-flex-or-air-application-development.html

To build and package the application for the armv8 architecture, an option is being provided in the latest release of IDEA. This update to the “Package AIR Application Dialog” will now give the user the full set of target architecture options.

4.8 FDT

With FDT currently the same mechanism should be used as for Adobe Animate, with a configuration file being used to force a target architecture.

Please note that new applications created using FDT will pick up an incorrect namespace, and the application descriptor file needs to be manually changed back to 32.0.

4.9 Visual Studio Code

`asconfig.json` already supports android packaging options including the "arch" value. For targeting armv8, this needs to be updated:

```
"airOptions": {
    "android": {
        "arch": "armv8"
    }
}
```

See https://github.com/BowlerHatLLC/vscode-as3mxml/wiki/asconfig.json#android-options

5 Configuration File

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

Current options are listed below:

Setting	**Explanation**
DefaultArch	Used as a default architecture if there is no “-arch” parameter provided to ADT. Values may be ‘armv8’, ‘armv8’, or ‘x86’.
OverrideArch	Used where an architecture value is being provided to ADT using the ‘-arch’ parameter, this configuration setting will override such parameter with the value given here. Values may be ‘armv8’, ‘armv8’, or ‘x86’.
DebugOut	If set to “true”, results in additional output being generated into a local file which can aid in debugging problems within ADT (including the use of third party tools from the Android SDK). Values may be ‘true’ or ‘false’, default is ‘false’.
UncompressedExtensions	A comma-separated list of file extensions that should not be compressed when such files are found in the list of assets to be packaged into the APK file. For example: “UncompressedExtensions=jpg,wav”
AddAirToAppID	Configures whether or not the “air.” prefix is added to an application’s ID when it is packaged into the APK. Values may be ‘true’ or ‘false’, default is ‘true’.

6 Android Applications – Play Store Uploads

New applications now need to have a 64-bit version of native code as well as a 32-bit version, as per the blog post from Google:

https://android-developers.googleblog.com/2019/01/get-your-apps-ready-for-64-bit.html

Currently the AIR SDK doesn't support Android App Bundle, or the concept of packaging support for multiple ABIs into a single APK file. To target multiple ABIs, developers therefore need to create multiple APK files. The guidelines and requirements for this are found at:

https://developer.android.com/google/play/publishing/multiple-apks

Please note in particular the following requirement:

Each APK **must have a different version code**, specified by the `android:versionCode` attribute

Currently the ADT packaging tool will generate the `android:versionCode` attribute based on the version number provided in your AIR Application Descriptor File (which is generated by the likes of Adobe Animate from within the version given in the target settings, i.e. "AIR for Android Settings" dialog box). In the XML-based application descriptor, this is the "`versionNumber`" value.

The version is a dot-separated series of up to three numbers, for example "10.2" or "15.123.5". Internally this is translated into the `android:versionCode` value by splitting the numbers into millions, thousands, and units (if there are less than three parts to the version number, these are assumed to be zero, i.e. "10.2" is the equivalent of "10.2.0").

Hence "10.2" will become 10 million 2 thousand, 10002000; "15.123.5" will become 15 million 123 thousand and 5, 15123005.

To create a set of APKs that can be uploaded to the Play Store that will cover both 32-bit and 64-bit ARM devices, a developer would therefore need to create two APK files using two different version numbers. Due to the way in which the Play Store determines which APK to serve to which customer, the 64-bit version needs to be at the higher version level (because the 32-bit version could also run on a 64-bit OS, so if that had a higher version then it would completely overshadow the 64-bit APK).

The workflow should therefore be:

1) Create a first APK file for 32-bit ARM (armv7)
2) Update the version number by as small as increment as possible
3) Create a second APK file for 64-bit ARM (armv8)
4) Upload both APK files to the Play Store.

There will still be a warning about the lack of use of the Android App Bundle, and the resulting inefficiencies, but this can be ignored.