# Ad SDK Integration for Mobile Game Publishers

Media and Advertising

| Table of Contents | |
|---|---|


## Executive Summary

This paper discusses integration of Ad SDK in mobile advertising. It highlights integration and monetization of mobile game apps; which are built using Unity, ANE, and Corona; through third-party Ad SDK.

## Introduction

The mobile gaming industry is on the rise for both gamers and publishers. The market share of mobile gaming stands at 51% of the total gaming revenue worldwide, which is estimated at $ 137.9 Bn with smartphones amounting to 80% of this and tablets being 20% (Source: Newzoo). In the US, mobile gaming revenue is estimated at $30.4 Bn. Game engines such as Unity and Unreal rule the cross-platform and mobile gaming, and power some of the most popular games around the globe.

With growing acceptance or mobile gaming, more and more ad networks and ad exchanges partner with various mobile app and game publishers to offer them ad inventory. Many such ad networks manage the demand from their advertiser partners and some of them also source such demand from third-party ad networks.

The publisher partners integrate Ad SDK into their mobile app for enabling such ad networks or exchanges to serve and place ads on the publishers' apps.

The publishers also include game publishers or developers, who generally build their mobile gaming app on top of game engine platforms such as Unity, ANE, and Corona.

This whitepaper intends to overcome common challenges and understand changes that such game publishers will need to make in their SDK for seamless integration with any Ad SDK. This whitepaper also captures the details around implementation and integration of wrappers or plug-ins that can be used by the game publishers to integrate with mobile Ad SDKs.

***What is a game engine?***

A game engine is a software framework designed for creating and developing games. Game developers or publishers use it to create games for consoles, mobile devices, and personal computers. Game engine technology is maturing and becoming more user-friendly. Game developers and publishers are now being serious about game experiences and are thus targeting towards multiple platforms including mobile devices and web browsers.

While there are many other game engines popular among mobile game developers and publishers, this whitepaper highlights solution on Ad SDK integration for the games developed using engines such as Unity, ANE, and Corona.

***Challenges that game publishers and developers face:***

- Game publishers build their game app using any of the previously listed game engines. In order to monetize their ad inventory, developers need to integrate mobile ad network and ad exchange SDK or plug-ins that are compatible to such game engines.
- Game engines have partnership only with leading mobile ad networks and ad exchanges leaving game publishers with very limited options to monetize.
- Game publishers prefer integrating with their existing partner ad network or ad exchange, but such partners may not offer Ad SDKs that are compatible to the game engines.

***Popular game engines:***

**Unity:** Unity is a flexible and powerful game development platform for creating multiplatform 3D and 2D games and interactive experiences. Unity offers a complete ecosystem for anyone who aims to build a business on creating high-end

content and connecting to their most loyal and enthusiastic game players. *Game publishers and developers need to have Native plug-ins included into their code to enable integration with partner Ad SDK.*

**Corona:** Corona's extensive API library enables everything from animation to networking with just a few lines of code. Whether you're building games or business apps, you can see changes instantly in the Corona Simulator and can iterate extremely quickly. *Game publishers and developers need to develop and integrate Native plug-in using library or service provider to enable integration with partner Ad SDK.*

**ANE:** Adobe AIR is by design cross-platform and device-independent, but AIR applications can still access the capabilities and APIs of native platforms through AIR native extensions. A native extension enables you to use platform-specific features, reuse existing native libraries, and achieve native-level speed for performance-sensitive code. *Game publishers and developers need to develop and integrate AIR Native Extension (ANE) file to allow integration with partner Ad SDK.*

# Solution Overview

***Technology as an enabler:***

Technological evolution in mobility defines the best practices for integrating mobile ad networks or ad exchanges supporting varied ad formats such as banner, interstitial, rich media, and video. This section details the solution approach for game developers and publishers on how they can update their SDK for ensuring monetization through ad integration and seamless integration of partner ad network with leading game engines such as Unity, Corona, and ANE. With some customization and workarounds, the same solution helps integration with generic OpenGL-based Android and iOS game engines as well.

**Unity:**

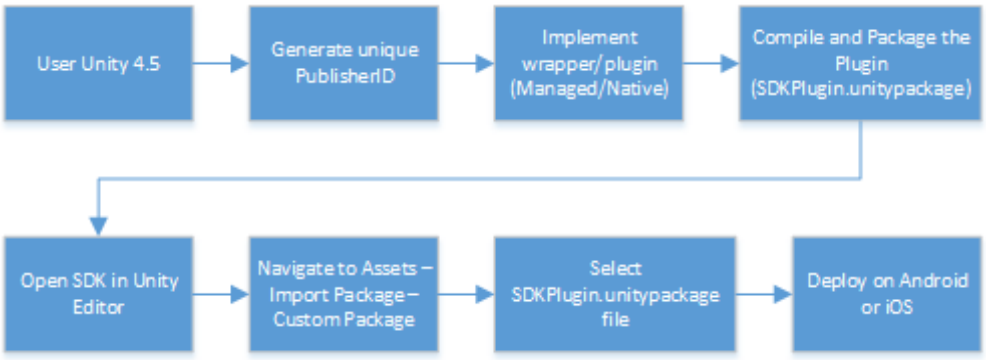
In Unity, game publishers and developers normally use scripts to create functionality, but they can also include code created outside Unity in the form of a plug-in.

There are two kinds of plug-ins available for use in Unity—Managed plug-ins and Native plug-ins.

**Managed plug-ins** are managed .NET assemblies created with tools such as Visual Studio or MonoDevelop. They contain only .NET code, which means that they can't access any features that are not supported by the .NET libraries. However, managed code is accessible to the standard .NET tools that Unity uses to compile scripts. There is, thus, a little difference in usage between managed plug-in code and Unity script code, except for the fact that the plug-ins are compiled outside Unity and so the source may not be available.

**Native plug-ins** are platform-specific native code libraries. They can access features such as OS calls and third-party code libraries that would otherwise not be available to Unity. However, these libraries are not accessible to Unity's

tools in the way that managed libraries are. For example, if you forget to add a managed plug-in file to the project, you will get standard compiler error messages. If you do the same with a native plug-in, you will only see an error report when you try to run the project.




- Prerequisite: Game publisher and developer will need to have Native plug-ins included into their code that are required to allow integration with partner Ad SDK for easy navigation on the website.
- Implementation Approach:
    - Use Unity 2018.3.3
    - Generate unique publisher ID or partner Ad SDK's ad unit ID
    - Implement a wrapper or plug-in (Managed plug-in or Native plug-in) compatible with Unity
    - Compile and package the plug-in in SDKPlugin.unitypackage file
- Integration Approach:
    - Open your SDK project in the Unity editor
    - Navigate to Assets -> Import Package ->Custom Package
    - Select the SDKPlugin.unitypackage file
    - Import all the files for the plug-ins by selecting Import. Make sure to check for any conflicts with files
        - To deploy on Android: Android: OS 4.1 or later
        - To deploy on iOS: XCode 9.0 or higher

**Unity Ads SDK**

It provides monetization framework for games and analytics related to advertising and in-app purchases.

- Prerequisite:
    - Latest Unity Ads SDK should be downloaded
    - Unity Developer (UDN) account should be registered on the Developer Dashboard
    - Developer Dashboard to be used to configure Placements for monetization content.

Unity recommends developers to read their best practices guide to understand their monetization strategy better.

- Implementation:

- Integrate Unity Ads using the Monetization API
- Ad integration options
    - Reward players for watching ads
    - Include banner ads
    - Include Augmented Reality (AR) ads
- Integrate with IAP (In-app purchases) Promo if your app uses in-app purchasing
- Unity provides the option to convert Ads and promo placements into personalized placements to take advantage of Unity's revenue optimization features

**CORONA:**

*Plug-ins allow you to extend the functionality of Corona.*

Development is done in Lua, a lightning-fast and easy-to-learn scripting language. Corona SDK allows you to publish on multiple devices such as
iPhone and iPad, Android phones and tablets, Amazon Fire, Mac desktop, Windows Desktop, and even connected TVs such as Apple TV, Fire TV, and Android TV from a single project.

Corona plug-ins leverage Lua's module system. In Lua's module system, plug-ins are lazily loaded by a corresponding call to the Lua function require(). Typically, these plug-ins are just shared native libraries, which is supported on platforms such as Mac (.dylib), Windows (.dll), and Android (.so).

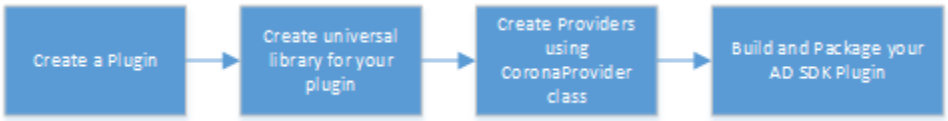
In some cases, plug-ins cannot be written as shared libraries:

- On iOS, shared libraries are not supported, so plug-ins must be static libraries (.a).
- On Android, it's often convenient to create pure Java plug-ins (.jar).

*Types of plug-ins:*

1. **Libraries**. Library plug-ins are just Lua libraries. These can be used to wrap native functionality. These libraries are namespaced with the prefix plug-in.
2. **Providers**. Provider plug-ins are designed to standardize functionality provided by third-party service providers. You would never explicitly load these plug- ins through require. Instead, a wrapper interface loads them implicitly because the wrapper depends on the provider for the underlying functionality. These wrapper interfaces can be libraries (for example, ads) or functions.

- Prerequisite: Game publisher or developer will need to develop and integrate Native Plug-in using library or provider, which allows integration with partner Ad SDK. Developing a library plug-in is preferred here.




- Implementation Approach:
    - Create a plug-in:  When you create a plug-in, you can use the App project template to simplify the plug-in creation and testing project. After all, the plug-in itself is not executable and needs to run inside an actual application.
    - This project has separate sub-folders for the platform-specific project files for each target platform: iOS and Android
    - There's also a Corona folder that sits alongside these folders that contain the classical Corona project, for example, main.lua. You can modify these files to test the Lua APIs that are offered by your plug-in
    - For iOS plug-in development, you need to modify the Plugin.xcodeproj
    - Create universal library for your plug-in
    - Build plug-ins on Lua's module system, so they can come in three different flavors – 1. Pure Lua code, 2. Native code (recommended to build wrapper for partner Ad SDK) and 3. Hybrid of Lua and Native code
    - Providers can also be created in Lua using the CoronaProvider class. You can create your own provider types in Lua by subclassing CoronaProvider.
    - Build and package your Ad SDK plug-in: Plug-ins should be submitted through a bitbucket repo that will be shared with you. Prepare the directory structure, create metadata.json, .jar (for Android) and metadata.lua (for iOS) and submit the plug-in

- Integration Approach:
    - SDK: Plug-ins are hosted on Corona's server. You can incorporate plug-ins by making an appropriate change in build. settings
    - Enterprise:  Native plug-ins can easily be added to your iOS or Android project. On iOS, plug-ins will be in the form of .a (static library) files that needs to be linked into your app executable. On Android, plug-ins can come in the form of .so (shared library) or .jar files. These files should be in your projects libs directory to ensure that the ant script sees them.
    - Library Plug-ins: Library plug-ins are simply Lua modules where the module name is prefixed with plug-in. The prefix helps avoid namespace collisions with Corona libraries. These libraries should be loaded through Lua's require function.

- Provider plug-ins: Provider plug-ins are also Lua modules, but you do not load them directly. Instead the wrapper library (or function) allows you to specify the provider and then loads them for you. In order for the provider to be loaded, it needs to follow a specific namespace convention. These libraries are prefixed with CoronaProvider followed by the name of the wrapper
- For game publishers or developers, it's recommended to use Native Library Plug- in development and integration for AD SDK.

Corona supports multiple plugins for in-app advertising and monetization; and multiple ad providers with a variety of ad units, which app developers can choose as per their monetization goals and design or UI requirements such as banners, interstitials (static and video), rewarded videos, and surveys.

| Plugin | Unit Types | Unique Features |
|---|---|---|
| Adcolony | video interstitial; rewarded video | |
| AdMob | banner; static interstitial; video interstitial | child-safe |
| AppLovin | static interstitial; video interstitial; rewarded video | |
| Appnext | static interstitial; video interstitial; rewarded video; other | |
| Appodeal | banner; static interstitial; video interstitial; rewarded video | ad meditation |
| Chartboost | static interstitial; video interstitial; rewarded video; other | |
| Facebook Audience Network | banner; static interstitial | |
| InMobi | banner; static interstitial | |
| KIDOZ | banner; static interstitial; other | child-safe |
| MediaBrix | static interstitial; video interstitial; rewarded video; other | |
| PeanutLabs | offer wall; poll/survey | |
| Persona.ly | ewarded video; offer wall; poll/survey | |
| Pollfish | poll/survey | |
| RevMob | banner; static interstitial; video interstitial; rewarded video | |
| SuperAwesome | banner; static interstitial; video interstitial | child-safe |
| Supersonic | static interstitial; rewarded video; offer wall | |
| TrialPay | static interstitial; video interstitial; rewarded video; offer wall | |
| Unity Ads | video interstitial; rewarded video | |
| Vungle | video interstitial; rewarded video | |

*Source: Corona Developer guide*

**ANE:**

Native extensions for Adobe AIR are code libraries that contain native code wrapped with an ActionScript API. Examples of native extensions include making a mobile device vibrate, integrating ad-networks, and in-app purchasing systems into your games.

Native extensions for Adobe AIR provide ActionScript APIs that provide you access to device-specific functionality programmed in native code. Native extension developers sometimes work with device manufacturers, and sometimes are third-party developers.

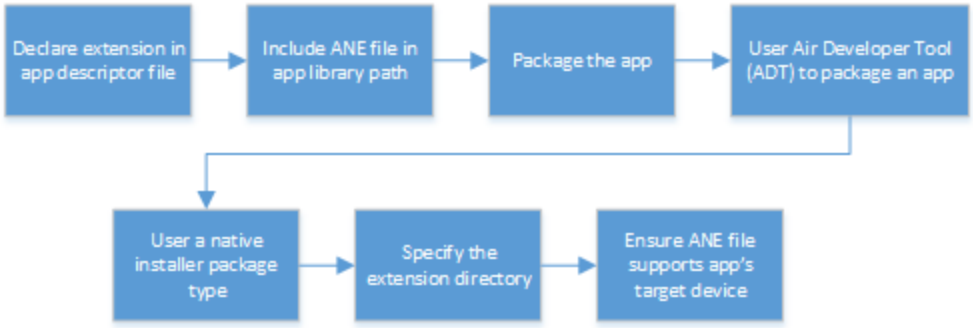
*AIR Native Extension (ANE) Files:*

Native extension developers package a native extension into an ANE file. An ANE file is an archive file that contains the necessary libraries and resources for the native extension.

As an AIR application developer, you use the ANE file as follows:

- Include the ANE file in the application's library path in the same way you include a SWC file in the library path. This action allows the application to reference the extension's ActionScript classes. Note: When compiling your application, be sure to use dynamic linking for the ANE. If using Flash Builder, specify External on the ActionScript Builder Path Properties panel; if using the command line, specify -external-library-path.
- Package the ANE file with the AIR application

- Prerequisite: Game publisher and developer will need to develop and integrate AIR Native Extension (ANE) file to allow integration with partner Ad SDK.




- Implementation Approach:
  - Declare the extension in your application descriptor file
  - Include the ANE file in your application's library path
  - Package the application

    - Include the ANE file in your application's library path: To compile an application that uses a native extension, include the ANE file in your library path – using the ANE file with Flash Builder or with Flash Professional.
    - Packaging an application that uses native extensions: Use AIR Developer Tool (ADT) to package an application that uses native extensions. You cannot package the application using Flash Professional CS5.5 or Flash Builder 4.5.1
- Integration Approach:
    - Use a native installer package type
    - Specify the extension directory
    - Make sure that the ANE file supports the application's target device

***Business-level benefits:***

- Flexibility to integrate any partner ad network or ad exchange
- A plug-in or wrapper SDK compatible to the game engine of your choice
- Scalable solution to include newer Interactive Advertising Bureau (IAB) mobile ad formats
- Compatibility to VAST, Video Player Ad- Serving Interface Definition (VPAID), and Video Multiple Ad Playlist (VMAP) among others as per IAB standards

# Conclusion

With the growing penetration of mobile gaming, game publishers and developers intend to monetize their inventory by integrating their preferred partner mobile Ad Network or Ad Exchange. Such game publishers either use leading game engines mentioned earlier or OpenGL-based Android and iOS game engines to build their games and apps. Developers find it challenging to integrate with partner mobile Ad SDK, which doesn't have readily available wrapper or plug-in or API compatible to such game engine frameworks used in the game development. This whitepaper serves the easy reference with implementation details to seamlessly integrate with preferred mobile Ad SDK. For the specific use case and game engines described in this whitepaper, game publishers and developers are recommended to implement and integrate:

- Native plug-in for Unity
- Native plug-ins using Library or Provider for Corona
- Native Extensions for ANE