Adze

> Read the code in the Examples Folder.

Introduction

Advertisements are the life-blood for many games. It is difficult to attract players if they have to pay upfront. There are many different formats with the three most common being:

- Banner - displays above or below game-play
- Interstitial - displays between game segments
- Reward - Player chooses to see an ad for gain

With an extensive range of networks to choose from:

- AdMob - Google-backed and the most popular
- Chartboost - specifically for games
- Appodeal - Aggregator of many of the others
- AdColony - Yet another contender for best network
- UnityAds - Simplicity to use, aimed at gamers
- and many more

Now you could use an aggregator and manually select the network. May networks provide aggregation. There are some problems with this:

- Dependency - If the aggregator is offline or shut down you will not have any advertisements served.
- Size - An aggregator loads libraries for all the networks it supports. It is a problem for Android with the 64k limit on entry points.
- Versioning - You have to rely on the version of a network API supplied by the aggregator.

- Complexity - The aggregator system to set up priorities on networks can be hard to program and painful to maintain.
- Reliability - Networks cannot always serve an advertisement. It is useful to have more than one source to cover this situation.
- Platforms - Different providers support different platforms, with Android and iOS being the most popular.

Adze provides a decoupled interface to the systems of your choice. If you have more than one source, you can choose different strategies for using them.

Adze also provides a rewarded video prefab that once reskinned can ease the process for you.

Installation

You will need to install the ***Askowl-Adze*** package using ***Askowl-Packager***. It is all you will need for development. The external interfaces are stubbed out and only show a message if used.

Install the API packages when you are ready to test on active devices. The stub message will give you a URL. Alternatively, you can use ***Askowl-Packager*** to fetch the ones you want.

Implementation

Servers and Distributors

All components are Custom Assets. Instances of Custom Assets reside in a ***Resources*** somewhere in your project.

1. Working in the Unity editor project view
2. Create a ***Resources*** directory if needed
3. Select this directory to open it
4. Open the right-click context menu
5. Select *Create // Adze // nameOfNetwork*
    1. Select the newly created asset
    2. Fill in the application keys for each platform you are supporting
    3. Select the type of advert (interstitial or reward)
    4. Select the priority (or leave all at 1) to set the order of network processing
6. Repeat for each network that may be needed
7. Select *Create // Adze // Distributor
    1. Choose round robin option.
    2. Drag in the networks you want active

To advertise in your code, fetch a reference to a *Distributor* and call show giving it an advertising type and an optional location, and network specific (unfortunately).

```
private void Start() {
  unityDistributor = AdzeDistributor.Asset(name: "UnityDistributor");
}

public IEnumerator ShowAd() {
  yield return unityDistributor.Show(Mode.Reward);
}
```

Rewards Prefab

A resource created using the custom asset AdzeReward will use the reward prefab to ask your player if they want to watch a video for a reward.

1. Drag the reward prefab from the Askowl-Adze/Prefabs folder into your scene. Retain the name *Reward*.
2. Modify the look and feel to match your game.
3. Create a custom Distributor asset in the Unity editor Create/Adze menu. It must be in a folder called *Resources*.
    1. Create and add AdzeServer custom assets for each network you want to use.
4. Create a custom asset in the Unity editor Create / Adze menu. It must be in a folder called *Resources*.
    1. Fill in some prompts and "thanks text" and button names.
    2. Drag the `AdzeDistributor` resource into the matching field.
5. When you want to offer a reward:
    1. Fetch a reference to the reward in `Start()` ... `ref = AdzeReward.Asset(assetName: "AdzeReward");` where `assetName` is optional if it is called *AdzeReward*.
    2. Display the results in a *Coroutine* `yield return ref.Show(location: location);`
    3. Check to see if the player accepted and watched the video `if (ref.AdWatched) ...`.

Entertainment

Put two text files in a directory called *quotes.txt* and *facts.txt*. Quotes are occasionally displayed instead of an ad. Set the frequency in the Rewards custom asset. Leave the file empty to not show quotes at all.

Facts are displayed while ads are loading. It will help keep player attention on slow networks.

Application Keys

Each AdzeServer resource can have an application key for each platform it supports. In this way, the same project can be used even when compiling for different target platforms.

Priority

Priority sets the order of networks.

- It is the order displayed in *Round Robin* mode.
- The highest priority will always be used when available when *Round Robin* is not selected.

Usage Balance

Each network will display *Usage Balance* times before going to the next on the list in *Round Robin* mode.

Round Robbin

In *Round Robin* mode each network is given an opportunity to display an ad. Use *Usage Balance* to control the proportion of the pie each option will consume.

When not in *Round Robin* mode the network with the highest priority will always be selected. Only when it cannot serve an ad is the next on the list used.

Adding a new advertising Network

At last count, there are over 40 networks, not including special and regional ones. Here is a partial list:

> AdColony, Adcash, Admob, Adsterra, Airpush, Applovin, Appnext, AppsUnion, Avazu, Billy, Mobile, Black6Adv, CPAlead, Chartboost, Clickdealer, Epom, Fyber, GOWIDE, InMobi, Kimia, Leadbolt, Minimob, MobAir, Mobidea, MobileCore, Mobobeat, MoBrain, Mobusi, Mobvista, Mpire, Network, Msales, PassionTeck, Persona.ly, Propeller, Ads, RevMob, Smaato, Startapp, Tapjoy, Unity, Ads, Vungle and Yeahmobi

`Adze` only supports a few of them. If you want to add a new one, it is a relatively simple process. Inherit from `AdzeServer` and override some methods and encapsulated fields. Use ***AdzeUnity.cs*** and ***AdzeChartboost.cs*** as examples. The former is the simplest implementation while the latter provides a more detailed implementation without getting too complicated.

Available Fields

- `AdActionTaken` -
  1. *Interstitial* - Is set if the player clicks on the advertisement. It can be used to provide some subtle reward.
  2. *Rewarded* - Is set if the player does not abort the ad. If not set, do not give then the prize unless you are feeling particularly generous.
- `AppKey` - Set in the Unity Editor for the current platform.
- `Error` - Set this if an error occurred. Used to decided whether to show an ad from another network.
- `Location` - Set optionally by the call to show. Its use is network specific. It is used for analytics and sometimes to help target ads in campaigns.
  1. *Chartboost* - Location can be user-defined, or one of Default, Startup, Main Menu, Achievements, Level Start, Level Dismissed, Turn Dismissed, Game Over,

   Leaderboard, IAP Store, Item Store, Settings and Quit.
2. *Unity* - It is the placement ID. Use **video** for interstitial and **rewardedVideo** for rewarded ads. You can also create new ones from the Unity console and use those.

- `Mode` - Can be Mode.Interstitial or Mode.Reward.
- `name` - The network name is the same as the class. An example is *AdzeAdMob*.
- `Priority` - Set in the Unity Editor. Change this to affect the order networks are accessed.
- `UsageBalance` - When in `RoundRobin` mode, each network will be used `UsageBalance` times before Adze will move on to the next network in the list.

Available Methods

- `Log(action, result, ...)` - A message is logged to the analytics package of choice. It will include the action, result, name of the network, type of advertisement, location supplied and any other parameters supplied.
- `LogError(message)` - Logs an error message to the analytics package of choice including the information listed above.

Virtual Methods and Encapsulated Fields

- `void Initialise()` - Is called once when the custom asset loads - so once per game invocation. Preparations may include initialising the network interface and preparing callbacks.
- `void Destroy()` - Is called just before the game exits.
- `bool ShowNow()` - Typically tells the network to display an ad and then returns immediately. Return false if the ad is not ready.
- `bool Dismissed` - Is an encapsulated boolean that returns true when the ad is no longer on the screen, and the game is ready to continue. Override the setter if the network requires manual caching.