IwGameAds SDK Programming

Developed and maintained by Mat Hopwood of Pocketeers Limited (http://www.pocketeers.co.uk).
For support please visit http://www.drmop.com

Table of Contents

Making Money from Ads

On certain platforms, trying to get users to part with “the cost of less than a cup of coffee” for all your hard work is seemingly impossible. A popular alternative to paid apps are free ad supported apps, where the app is given away for free and revenue is generated by users clicking on ads displayed within the app. Before we get into the coding side of things there are a few important things you need to know about ads and the ad industry in general.

An ad's potential to make you money is measured in CPC (cost per click) or eCPM (cost per 1000 impressions). I prefer to deal in CPC as its a much clearer measuring stick to determine the value of particular ads and ad providers. Unfortunately the industry hides behind eCPM because it is vague and easy to mask, however its all we have to measure ad performance so we will speak in terms of eCPM.

eCPM is calculated based on how much you have or will have earned based on an average 1000 ads shown to users. Values can vary from as low as $0.10 to as high as $5.00. So as you can see it is going to take a LOT of impressions to make any decent money, but it is better than your app sat on some app store making nothing because users don't even know its there. Free apps tend to get much more exposure and users are much more likely to download them because they are free.

The next term you need to become familiar with is CTR (click through rate). This is the ratio of ads served to your app and the number of people that have actually clicked on your ads. So for example, lets say you get 10,000 ads served to your app in one day and 500 users click on ads, this will make you CTR (500 * 100) / 10,000 = 5%. Click through rates can typically range from 0.1% to 10.0%, with the average being around 1%. At the moment ads running using IwGame's animating ads system are getting the following CTR's:

- Android – 3.2%
- iPhone – 8%
- iPad – 3%
- Samsung Bada - 4%

As you can see by adding some style to your ads you can vastly exceed the standard 1% CTR rate that most static ads obtain.

The last term that you should need to become familiar with is fill rate. This is the ratio of ad requests (termed requests) made from your app to ads actually sent back (termed impressions) to the user from their ad network. This varies based on the device running your app, the country and language of the user and the ad providers size and popularity. To calculate fill rate you divide the number of impressions by the number of requests. So for example, lets say you get 10,000 ad requests in one day but only 5,000 impressions then you fill rate is (5000 * 100) / 10,000 = 50%. Average fill rates vary massively across platform, provider and country. iPhone and Android platforms tend to have very high fill rates (typically 90%+) because they are popular and everyone wants to advertise their wares on these platforms. Platforms such as Samsung Bada and Blackberry Playbook experience much lower fill rates. Bada fill rates (can be as low as 20%,) simply because they aren't quite as popular.

One way to combat low fill rates is to use multiple ad providers. If one ad provider does not return an ad for your platform then request an ad from the next provider. (usually referred to as ad mediation) You may need to integrate 4-5 ad providers to ensure a 100% fill rate and maximise your apps earning potential on some platforms.

Introduction to IwGameAds

IwGameAds is a cross platform C++ unified ad API that allows you to make requests for ads from a number of ad providers. It also goes one step further and provides a view that can display retrieved banner / full screen ads and register clicks. IwGameAds provides the following functionality:

- Asynchronously request text, image and html based ads from a variety of ad providers
- Automatic extraction of ad data, such as image url, click url and text
- Provision for targetted ads by age, gender, location etc..
- Integrated ads view that can display animating ads to the user as well as detect clicks and launch the external URL / app that deals with the click destination
- Cache and display multiple ads from multiple providers
- Ad mediation using the priority based IwGameAdsMediator class

IwGameAds currently supports the following ad providers:

- Inner-active
- AdFonic
- Vserv – Also provides support for InMobi, BuzzCity, JumpTap, ZestAdz / Komli Mobile and Inner-active
- Mojiva
- Millennial Media – Also provides support for AdMob, Amobee, JumpTap and Mojiva
- AdModa

More will be added as they are confirmed as working.

In order to use IwGame you will need a copy of the the Marmalade SDK which you can download from http://www.madewithmarmalade.com

IwGame is currently maintained at http://www.drmop.com/index.php/iwgameads-sdk/

Usage Rights and Warranties

IwGameAds and associated classes and components are provided “as is” and “without” any form of warranty. Yours, your employers, your companies, company employees, your clients use of IwGameAds is completely at your own risk. We are not obligated to provide any kind of support in any shape or form.

You are free to use IwGameAds in your projects in part or in whole as long as the header comments remain in-tact.

You may not claim the IwGameAds SDK or its documentation as your own work or package it up and include it in any kind of Middleware product without express prior written notice from an executive of Pocketeers Limited with the correct authority to do grant authorisation. The only exception to this rule is provided to the makers of the Marmalade SDK.

Our aim is to promote IwGameAds as a viable cross platform ad SDK built on top of the Marmalade SDK, so any publicity will serve to not only help increase public awareness of IwGameAds in the development community but also increase awareness of the Marmalade SDK, who made this all possible.

If you would like to let us know that you are using IwGameAds in your products then please get in touch with us at admin@pocketeers.co.uk. We also appreciate any comments or feedback.

Installing the IwGameAds SDK

Firstly, download the latest version of the IwGameAds SDK from
http://www.drmop.com/index.php/iwgameads-sdk/

If you haven't already installed the Marmalade SDK then grab a copy from
http://www.madewithmarmalade.com

IwGameAds is currently provided as a Marmalade library project and test bed project giving examples of how it should be used.

The IwGameAds zip archive that you downloaded contains the following important folders:

```
IwGame
      Docs - Contains documentation
      h - Contains IwGame engine headers
      Source - The full source to IwGame
      TestBed - A test bed app that shows IwGame usage and integration
```

To open up the test bed app go to IwGame\TestBed and open the IwGameAds_Test.mkb Marmalade project file, this will open up the test bed project in Visual Studio / XCode.

If you take a look at the solution explorer you will see a IwGameAds filter / folder. If you drill down you will be able to see the source files and the Marmalade project file that make up the SDK.

Now lets take a look at the IwGameAds_Test .mkb project file to find out how to integrate IwGameAds into your project.

From the MKB we can see that we only need to add two commands to integrate IwGameAds into our project:

```
options
{
      module_path="../IwGameAds"
}

subprojects
{
      IwGameAds
}
```

These commands will tell Marmalade where to locate the project.

Once you have updated your project MKB with the above changes, run your MKB file again to regenerate your project. You can of course place the IwGameAds engine anywhere you like, just remember to change the module path.

Note that if you already have something in module_path then you can quite easily tag IwGameAds into the path like this:

```
module_path=”module_path="$MARMALADE_ROOT/modules/third_party;../IwGameAds"
```

IwGameAds Programming

Basic Integration

CIwGameAds and CIwGameAdsView are both singletons that provide easy application wide access from anywhere in your game. CIwGameAds is not dependent upon CIwGameAdsView so can roll your own ad view system if you want to.

If you are going to use the provided ads view then the process is setting up and updating IwGameAds is very simple as the view will take care of initialising and updating the ads system:

```
// Create ad view
CIwGameAdsView::Create();

// Initialise with Application ID (you get this from your ad provider)
IW_GAME_ADS_VIEW->Init("Your App ID");

// Set ad request interval in seconds
IW_GAME_ADS_VIEW->setNewAdInterval(30);

// Set ad provider (if you want ads to be automaticall collected at an interval)
IW_GAME_ADS_VIEW->setAdProvider(CIwGameAds::InnerActive);

// Set total number of ads visible in the ads view
IW_GAME_ADS_VIEW->setNumAdsVisible(1);
```

If you not want ads to be requested automatically at a set interval then you should not use setNewAdInterval(). Calling setAdInterval(0) will disable auto collection of ads.

And updating the view in your main loop would look like this:

```
// Update the ads view
IW_GAME_ADS_VIEW->Update(1.0f);
// draw the ads view
IW_GAME_ADS_VIEW->Draw();
```

Finally cleaning up the ads view is done like this:

```
// Clean up the ads view
IW_GAME_ADS_VIEW->Release();
CIwGameAdsView::Destroy();
```

As you can see its a very simple 3 stage integration. If however you have decided to roll your own ads view then you will need to set up CIwGameAds yourself:
To initialise the ads system you should implement the following:

```
// Create ad view
CIwGameAds::Create();
IW_GAME_ADS->Init();
// Set the Application ID (you get this from your ad provider)
IW_GAME_ADS->setApplicationID(id);
```

You need to update the ads system each game loop using:

```
// Update the ads system
IW_GAME_ADS->Update();
```

And finally clean up the ad system when done with it using:

```
IW_GAME_ADS->Release();
CIwGameAds::Destroy();
```

Different Ad Types

CIwGameAdsView currently supports the rendering of image based banner ads only. It will not render text or html based ads. However you tell the ads system which types of ad you are willing to accept. You can tell the ad system to return these types of ads using:

```
void                setTextAds(bool text_ads)
void                setHtmlAds(bool html_ads)
```

Note that not all ad providers may offer this functionality so the system uses it more as a hint.

Requesting Ads

Requesting an ad using CIwGameAdsView is very simple as the following piece of code will show:

```
// Request an Inner-active ad
IW_GAME_ADS_VIEW->RequestNewAd(CIwGameAds::InnerActive);
```

Once the ad is available the ads view will display it to the user.

If you are rolling your own ad view then requesting ads looks very much the same.

```
// Request an Inner-active ad
IW_GAME_ADS->RequestAd(CIwGameAds::InnerActive);
```

The slightly more complicated process is detecting when the ad has arrived. The code below shows how to detect the arrival of a new ad and how to get at the ad information:

```
// Check to see if new ad has arrived
if (IW_GAME_ADS->isAdAvailable())
{
	CIwGameAds::eIwGameAdsError error = IW_GAME_ADS->getError();
	if (IW_GAME_ADS->getError() <= MinError)
	{
		// No error - You can now acess the ads information
		CIwGameAd& ad = IW_GAME_ADS->getAd();
	}
	else
	{
		// Error occuured - error contains an error
	}

	// Allow the next ad request
	IW_GAME_ADS->setAdAvailable(false);
}
```

Your own implementation will also need to take care of displaying ads and responding to clicks.

Working with CIwGameAdsView

CIwGameAdsView is a convenient ad view that will retrieve ads, display them using animations and monitor and respond to ad clicks. We've already explained how to create an update an ads view, but we haven't yet covered some of the cooler features of the view. Here is a list of features:

- Can display multiple ads
- Can display ads anywhere on screen
- Can rotate and scale ads
- Can change the colour and opacity of ads
- Can hide and show ads
- Can animate ads

When you create the CIwGameAdsView you can set how many ads you would like the view to display using:

```
    void                setNumAdsVisible(int count)
```

You can also set the position, scale, rotation, colour and visibility of each separate ad as well add animators using the following methods:

```
    void                setVisible(int index, bool visible)
    void                setPosition(int index, int x, int y)
    void                setScale(int index, float scale)
    void                setAngle(int index, float angle)
    void                setColour(int index, int r, int g, int b, int a)
    void                addAnimator(int index, CIwGameAdsViewAnimator* animator)
```

Using the above methods you can provide a whole range of animations to your ads

Targeted Ads

The CiwGameAds class provides methods for setting additional targeting information that can be sent in an ad request to ad providers, this can help to return ads targeted at a specific audience and increase CTR. The following target information can be supplied:

- User-Agent – You can provide your own custom user-agent
- IP Address – You can supply your own custom IP address
- Gender – See eIwGameAdsGender
- Location – Format is determined by the ad provider
- UserGPSLocation – Format is determined by the ad provider
- Category - Single word description of the application
- UserMobileNumber - Users mobile number - MSISDN format, with international prefix
- UserKeywords - Comma separated list of keywords relevant to this user's specific session
- ExtraInfo – A list of key/value pairs separated by an ampersand, e.g. colour=red&height=70&wast=32

Ad Animators

To test out our animating ads idea we created a basic class called CIwGameAdsViewAnimator. This class provides an easy way to quickly get ads animating. This class is provided mainly as an example showing how to animate ads yourself.

The ad view animator assumes that the ad is going to be in one of 4 states:

- AnimIn – The ad is coming into view
- AnimOut – The ads is going out of view
- AnimStay – The ad is at rest but visible to the user
- AnimDone – The ad is finished

The ad view will allow you to add animations for the in, out and stay phases.

Hers an example that shows a typical sweep in over one second, stay on screen for seven seconds and then sweep back out over one second.

```
// Create and attach an animator that fades the ad in over 1 second, pauses for 7
// seconds and then fades the ad back out
CIwGameAdsViewAnimator* anim = new CIwGameAdsViewAnimator();
anim->Init();
anim->setAdViewDataIndex(0);
anim->setCanvasSize(width, height);
anim->setInAnim(CIwGameAdsViewAnimator::AnimFadeIn, 1000);
anim->setOutAnim(CIwGameAdsViewAnimator::AnimFadeOut, 1000);
anim->setStayDuration(7000);
IW_GAME_ADS_VIEW->addAnimator(0, anim);
```

You can add multiple animations to the same ad as shown in the following example:

```
// Create and attach an animator that sweeps the ad in from the right the over 1.2
// seconds, pauses for 7 seconds and then sweeps back out
anim = new CIwGameAdsViewAnimator();
anim->Init();
anim->setAdViewDataIndex(0);
anim->setCanvasSize(width, height);
anim->setRestingPosition(0, -height / 8);
anim->setInAnim(CIwGameAdsViewAnimator::AnimRightSweepIn, 1200);
anim->setOutAnim(CIwGameAdsViewAnimator::AnimRightSweepOut, 1200);
anim->setStayDuration(7000);
IW_GAME_ADS_VIEW->addAnimator(0, anim);

// Create and attach an animator that scales the ad in over 1.5 seconds, pauses for 7
// seconds and then scales back out
anim = new CIwGameAdsViewAnimator();
anim->Init();
anim->setAdViewDataIndex(0);
anim->setCanvasSize(width, height);
anim->setInAnim(CIwGameAdsViewAnimator::AnimScaleIn, 1500);
```

```
	anim->setOutAnim(CIwGameAdsViewAnimator::AnimScaleOut, 1500);
	anim->setStayDuration(7000);
	IW_GAME_ADS_VIEW->addAnimator(0, anim);

	// Create and attach an animator that rotates the ad in over 1 second, pauses for 7
	// seconds and then rotates back out
	anim = new CIwGameAdsViewAnimator();
	anim->Init();
	anim->setAdViewDataIndex(0);
	anim->setCanvasSize(width, height);
	anim->setInAnim(CIwGameAdsViewAnimator::AnimSpinIn, 1000);
	anim->setOutAnim(CIwGameAdsViewAnimator::AnimSpinOut, 1000);
	anim->setStayDuration(7000);
	IW_GAME_ADS_VIEW->addAnimator(0, anim);
```

By combining animators you can produce some interesting effects.

Ad Mediation

Integrating a single ad provider into your product is not a great idea if you want to maximise your products earning potential. When ad providers give fill rate figures of over 90%, they leave out the fact that the fill rate they advertise is based upon their best performing platforms and countries. An ad provider will based eCPM on how many ads were delivering to your app and not how many ad requests were made from your app, which can be VERY misleading. Some platforms such as Bada have fill rates as low as 20%, whilst some countries such as China have even lower fill rates.

To get around this problem you need to use ad mediation. This is the process of going through a list of prioritised ad providers requesting an ad, if the ad provider does not provide an ad then request an ad from the next provider. Keep doing this until you get an ad that you can use.

IwGame provides automated ad mediation using the CIwGameAdsMediator. To use the mediator you create a CIwGameAdsMediator, populate it with ad party's then attach it to the CIwGameAds object as shown below:

```
// Create ad mediator and attach it to the main ad object
CIwGameAdsMediator* ad_mediator = new CIwGameAdsMediator();
IW_GAME_ADS->setMediator(ad_mediator);

// Create Inner-active ad party and add to the mediator
CIwGameAdsParty* party = new CIwGameAdsParty();
party->ApplicationID = "Your inner-active App ID";
party->Provider = CIwGameAds::InnerActive;
ad_mediator->addAdParty(party);

// Create AdModa ad party and add to the mediator
party = new CIwGameAdsParty();
party->ApplicationID = "You AdModa App ID";
party->Provider = CIwGameAds::AdModa;
ad_mediator->addAdParty(party);

// Create AdFonic ad party and add to the mediator
party = new CIwGameAdsParty();
party->ApplicationID = "Your AdFonic App ID";
party->Provider = CIwGameAds::AdFonic;
ad_mediator->addAdParty(party);
```

As you can see, incredibly simple. The process of attaching the ad mediator will ensure that the ad object will attempt to collect ads from all ad parties should previous parties fail to collect an ad. You do not need to make any changes to the ad view, the system is completely automated.

Supported Ad Providers

The following ad providers are currently supported:

- Inner-active
- AdFonic
- Vserv – Also provides support for InMobi, BuzzCity, JumpTap, ZestAdz / Komli Mobile and Inner-active
- Mojiva
- Millennial Media – Also provides support for AdMob, Amobee, JumpTap and Mojiva
- AdModa

We have plans on adding support for the following ad providers:

- MobFox
- InMobi
- Madvertise

If you know of any additional ad providers that support a REST based API then please let us know at admin@pocketeers.co.uk

Ads Test Bed

An example test bed app has been included in the Testbed folder to show you how to easily integrate IwGameAds into your own products.

The test bed shows a basic game loop with ad initialisation, clean-up and per frame processing functions.

- AdTest_Init() - Initialises the ad system and any dependent sub systems (called before your main loop begins processing)
- AdTest_InitMediator() - Optional ad mediator initialisation (called after AdTest_Init()
- AdTest_Release() - Ad engine clean up (called before app exits)
- AdTest_Update() - Per frame ad engine processing (called every game frame update)
- AdTest_Show(bool show) – Shows / hides the ad view

Note that the ad animation system is disabled by default in the test bed, but you can enable this by uncommenting line 12

```
#define ANIMATE_ADS
```

A Few Additional Notes

If you check the IwGameAds.h header file you will see a number of constants defined in there:

```
#define       IW_GAME_ADS_TIMEOUT         10000
#define       IW_GAME_MAX_CACHED_ADS      4
#define       IW_GAME_MAX_AD_AGE          (5 * 60000)
```

IW_GAME_ADS_TIMEOUT - Represents the amount of time that IwGameAds (in millseconds) will wait before deciding if an ad request has timed out. If you set this value too low or too high then you may never collect an ad.
IW_GAME_MAX_CACHED_ADS - Represents the maximum number of ads that the system will cache
IW_GAME_MAX_AD_AGE – Represents the amount of time (in milliseconds) that an add will be cached, if you are using cached ads then you may need to adjust this value to work a wide range of ad providers

OpenGL Considerations

If you are using raw OpenGL functions in your rendering pipeline then you may experience a conflict between Iw2D and OpenGL. The problem is that Iw2D can change certain GL states that are not put back to their original setting when Iw2D has finished rendering. For example, Iw2D can change the texture matrix or the shading model state (amongst other states). To combat this issues you should ensure that you reset any features that you plan to use after displaying the ads view. A couple of examples include:

To switch back to smooth shading use:

```
glShadeModel(GL_SMOOTH);
```

To repair the texture matrix add the following before you render your GL content:

```
glMatrixMode(GL_TEXTURE);
glPushMatrix();
glLoadIdentity();
```

After your GL rendering is finished restore the GL texture matrix:

```
glMatrixMode(GL_TEXTURE);
glPopMatrix();
```