Config.lua Parameters

This page explains how the config.lua file works, what parameters are available and what they do.

What is config.lua ?

The file config.lua is used to store all game configuration data. The primary purpose is to make the startup process of your iOS app data-driven. Most pre-existing parameters are for the configuration of Cocos2D global settings and described below. But you can also use it to supply your code with your own startup parameters.

The syntax of the file is as follows:

```
local config =
{
 KKStartupConfig =
 {
  -- Kobold2D startup parameters go here
 },

 MyGlobalSettings =
 {
  -- your parameters go here, example:
  DebugMode = YES
 }
}

return config
```

To make use of your own configuration space you have to add properties to your class with the same name as in the Lua file, but beginning with a lowercase character. In the above example, you would create this property:

```
@property (nonatomic) bool debugMode;
```

To inject the values from config.lua into the properties in your class, you need just one line of code:

```
[KKConfig injectPropertiesFromKeyPath:@"MyGlobalSettings" target:self];
```

This will check the current class for corresponding properties and will assign the value from config.lua to that property. In the above case it has a setting DebugMode so it looks for a @property named debugMode. If found, the value of DebugMode is assigned to the debugMode property. It is your responsibility to ensure that the property type (bool in this case) matches the type used in config.lua.

KKStartupConfig Parameters Explained

Game Setup

FirstSceneClassName

Possible Values
"ClassName"

Example:
FirstSceneClassName = "GameScene";

"ClassName" is a string that defines the name of an Objective-C class which is the first scene that should be loaded. This class must be derived from a CCScene class.

If an Objective-C class with this name can not be found, Kobold2D looks for a lua file in the format ClassName.lua (beware: filenames are case-sensitive on iOS!). If found, it will load this script and create a KKScriptScene derived class which will become the first scene.

If neither is found or the FirstSceneClassName is an empty string, it is up to you to write code that calls the CCDirector runWithScene method. You normally do this in the AppDelegate class, by providing the following function and code:

```
-(void) initializationComplete
{
 [[CCDirector sharedDirector] runWithScene:[MyScene node]];
}
```

DirectorType & DirectorTypeFallback

Possible Values
DirectorType.NSTimer DirectorType.MainLoop DirectorType.ThreadMainLoop DirectorType.DisplayLink

DirectorType determines how Cocos2D's Director implements the game loop.
DirectorTypeFallback allows you to specify to which Director type Cocos2D should fall back to in case the primary Director type is unavailable. This is only needed if you specify the DisplayLink director because it requires iOS 3.1 or higher and may not be available on all devices.

The default setting is the DisplayLink director, which couples the updates with the screen refresh rate using a CADisplayLink object.

If you want to use UIKit views in your game, it is highly recommended to use the ThreadMainLoop director, which runs Cocos2D on a seperate thread except for the rendering part. This will make UIKit views more responsive while Cocos2D is running.

The MainLoop director uses a classic "while (isRunning)" endless loop which is non-cooperative and should be avoided since it may not interact well with iOS multitasking. The NSTimer is a safe fallback but has since become largely irrelevant, since almost all devices have iOS 3.1 installed and can run the DisplayLink director. But it is still used as the safe fallback option.

MaxFrameRate

Possible Values

1 to 60

How many times per second the screen content should be updated (framerate). This only determines the maximum possible frames per second (fps) which are drawn. If your game can not keep up computing and drawing its contents in the time between two frames, your framerate will drop. To achieve 60 frames per second, your update loop and drawing must be completed within 16,6 milliseconds (0,0166 seconds).

All iOS devices are locked at a maximum of 60 frames per second. Setting MaxFrameRate above 60 is not recommended.

If your game's framerate fluctuates, it will feel smoother for the user if you reduce the MaxFrameRate. For example if the fluctuation is in the range of 40 to 60 fps, you might want to experiment setting the MaxFrameRate in the range of 40 to 45 in order to achieve a slower but more constant framerate.

DisplayFPS

Possible Values
YES NO

Set to YES to display a framerate counter in the lower left corner of the screen. By the default, the framerate counter is only visible in Debug and Release builds, but will be disabled automatically in Ad Hoc and App Store distribution builds.

DisplayFPSInAdHocBuilds

Possible Values
YES NO

In some cases, you want the users of your Ad Hoc builds to also see the framerate. Set DisplayFPSInAdHocBuilds to YES if you want to show the framerate in Ad Hoc builds as well. DisplayFPS must also be set to YES. And the framerate counter will still be disabled in App Store distribution builds.

EnableUserInteraction

Possible Values
YES NO

This must be enabled to receive any events from the user. This includes touch events as well as keyboard input events. Typically you'll want this to be enabled.

EnableMultiTouch

Possible Values
YES NO

Set to YES if you want to receive and handle multi touch events. If you only need to receive single touches, set this to NO.

Render Settings

DefaultTexturePixelFormat

Possible Values

Possible Values
TexturePixelFormat.RGBA8888 TexturePixelFormat.RGBA4444 TexturePixelFormat.RGB565 TexturePixelFormat.RGB5A1

Sets the default pixel format for new textures.

Format	Meaning
RGBA8888	32-Bits Color Depth, 1 Alpha Channel (Bits: 8 red, 8 green, 8 blue, 8 alpha)
RGBA4444	16-Bits Color Depth, 1 Alpha Channel (Bits: 4 red, 4 green, 4 blue, 4 alpha)
RGB565	16-Bits Color Depth, no Alpha Channel (Bits: 5 red, 6 green, 5 blue)
RGB5A1	16-Bits Color Depth, on/off Alpha Channel (Bits: 5 red, 5 green, 5 blue, 1 alpha)

Using RGBA4444 for textures is a quick way to effectively half the memory usage of your textures while improving rendering performance. If you don't use gradients in your images you may not even see a big loss in image quality.

GLViewColorFormat

Possible Values
GLViewColorFormat.RGBA8888 GLViewColorFormat.RGB565

Determines the color depth of the render surface.

RGBA8888 is a 32 bit color surface, RGB565 has 16 bit color depth. More colors, better image quality but slower rendering. The Color Format also affects how much memory is reserved for the render buffer. The calculation is as follows:

```
ScreenWidth * ScreenHeight * ColorDepth (in Bytes) * 2 (Double Buffering) / 1024 = Memory Used (in KB)
```

Color Format	Device Resolution	Memory Usage
RGB565	480x320	600 KB
RGB565	960x640	1200 KB
RGB565	1024x768	3072 KB
RGBA8888	480x320	1200 KB
RGBA8888	960x640	4800 KB
RGBA8888	1024x768	6144 KB

GLViewDepthFormat

Possible Values
GLViewDepthFormat.DepthNone GLViewDepthFormat.Depth16Bit GLViewDepthFormat.Depth24Bit

How many bits to use for the Z-Buffer, or none at all. You should use DepthNone unless specifically instructed to do so. You will only have to use a depth buffer in one of these cases:

- you are using 3D models (eg. via Cocos3D or your own GL drawing code)
- you are using 2D Projection, usually used for Isometric Tilemaps
- you want to use the vertexZ property instead of zOrder for sorting the draw order of sprites

Enabling the Z Buffer via Depth Format consumes additional memory. The table was created according to this calculation:

```
ScreenWidth * ScreenHeight * DepthFormat (in Bytes) * 2 (Double Buffering) / 1024 = Memory Used (in KB)
```

Depth Format	Device Resolution	Memory Usage
Depth16Bit	480x320	300 KB
Depth16Bit	960x640	600 KB
Depth16Bit	1024x768	1536 KB
Depth24Bit	480x320	450 KB
Depth24Bit	960x640	1800 KB
Depth24Bit	1024x768	2304 KB

GLViewPreserveBackBuffer

Possible Values
YES NO

Determines if the contents of the render surface should be cleared each frame. According to Apple:
"Setting the value to YES is recommended only when you need the content to remain unchanged, as using it can result in both reduced performance and additional memory usage. The default value is NO."

GLViewMultiSampling

Possible Values
YES NO

Enables or disables multisampling, also called anti-aliasing. It's a technique to smooth edges and improves image quality. But in some circumstances it may also be perceived as an undesirable blur effect.

It is recommended to set this to NO since multisampling can affect performance, especially on 1st and 2nd generation devices. Multisampling will also double the memory used by the render buffer (see the GLViewColorFormat table).

If you enable multisampling, you must also set the GLViewNumberOfSamples property.

GLViewNumberOfSamples

Possible Values
0 1 2 3 4

How "strong" the multisampling effect should be. Using 0 with multisampling enabled will effectively disable multisampling but still create the multisampling renderbuffer. The higher the value, the more surrounding pixels will be considered for multisampling, but higher values may also adversely affect performance.

Enable2DProjection

Possible Values
YES NO

Enable 2D Projection only if you want to render content with depth buffering. You will only need to enable 2D Projection if you want full control over the Z ordering (layering) of all sprites using the CCNode vertexZ property. Some games need to do this to correctly overlap sprites based on their Y position or in relation to where the player character is.

For example, a game like Final Fantasy VI would need to use 2D Projection so that characters can both stand in front of a house but also being occluded if walking around the back of a house:

> ⚠ Note that 2D Projection won't do this automatically for you. It just allows you to implement your custom sprite Z ordering algorithm via the vertexZ property.

EnableRetinaDisplaySupport

Possible Values
YES NO

If you want to provide HD graphics for Retina devices, you should enable Retina Display support. Cocos2D will then try to load images, particle effects and fonts with -hd suffix on Retina devices while keeping the same aspect ratio.

If you enable Retina Display support but some or all images do not have a HD version with the -hd suffix, those SD images will be displayed smaller than on a regular iPhone display. If you want to support Retina, you have to create Retina assets for all images, particle effects, fonts and so on.

> ⛔ **Don't scale!**
> You may be tempted to provide only SD or only HD images and scale them on Retina or non-Retina devices respectively. Don't do this!
>
> If you only provide HD images, the non-Retina devices will have to load images which are 4 times larger than they need to be. You'll quickly run out of memory, especially if you consider that non-Retina devices have far less memory (128 MB or 256 MB).
>
> On the other hand there is no point in supporting Retina display if all you do is to scale up all your SD images. You don't gain image quality, it's no different from not supporting Retina and leaving it up to the device to scale up your content.

Orientation & Autorotation

DeviceOrientation

Possible Values
DeviceOrientation.Portrait DeviceOrientation.PortraitUpsideDown DeviceOrientation.LandscapeLeft DeviceOrientation.LandscapeRight

In which device orientation the game should be started. If AutorotationType is set to None, this will specify the orientation your app supports.

EnableAutoRotation

Possible Values
YES NO

Set this to YES if you want to enable the autorotation feature. See the next parameters. Otherwise leave it at NO to only support the initial device orientation.

AutorotationType

Possible Values
Autorotation.None Autorotation.CCDirector Autorotation.UIViewController

If set to None, the screen content will never change orientation regardless of how the user is holding the device.
If set to CCDirector, the screen content will be rotated by transforming the GL View. This is a relatively fast operation.
If set to UIViewController, the RootViewController will perform the rotation. This is slower than rotating via CCDirector but will also properly rotate UIKit views.

If you enable autorotation, you should also specify to which orientations you want to allow. See the ShouldAutorotate* parameters.

ShouldAutorotateToLandscapeOrientations & ShouldAutorotateToPortraitOrientations

Possible Values
YES NO

These settings allow you to control which orientations your app supports. Especially for games it often makes little sense to allow users to rotate from landscape to portrait and vice versa. But it requires no additional effort to allow users to rotate the device from the LandscapeLeft to LandscapeRight orientation respectively from Portrait to PortraitUpsideDown (and vice versa).

Set ShouldAutorotateToLandscapeOrientations to YES if you want to support both landscape orientations.
Set ShouldAutorotateToPortraitOrientations to YES if you want to support both portrait orientations.

If you set both to YES, you will support all rotations and you have to consider how your game should behave if it is rotated from landscape to portrait and vice versa.

---

iAd Setup

EnableAdBanner

Possible Values
YES NO

Set this to YES if you want to show iAd advertisement banners in your App.

iAd & Autorotation
For iAd to support autorotation you should use the AutorotationType set to UIViewController. Otherwise ad banners will not rotate and might be shown upside down or displaced.

LoadOnlyPortraitBanners & LoadOnlyLandscapeBanners

Possible Values
YES NO

If your app only supports rotation to either portrait or landscape orientations, you should specify that for iAd banners as well. For example, if you only enable ShouldAutorotateToLandscapeOrientations but not Portrait, you should set LoadOnlyLandscapeBanners to YES and LoadOnlyPortraitBanners to NO.

This will prevent iAd from loading banners over the network which will never be shown. This saves bandwidth and loads ads faster.

PlaceBannerOnBottom

Possible Values
YES NO

By default, iAd banners are shown on the top of the screen. Set PlaceBannerOnBottom to YES if you want the iAd banners to be placed at the bottom of the screen.

For custom placement of iAd banners you will have to subclass the KKRootViewController class in your project.

Mac OS X exclusive settings

These settings are specific to Mac OS X builds and will be ignored in iOS builds.

AutoScale

Possible Values
YES NO

Set this to YES if you want the window content to be scaled to the window size.

AcceptsMouseMovedEvents

Possible Values
YES NO

Set this to YES if you want to receive MouseMoved events.

WindowFrame

Possible Values
CGRectMake(x, y, width, height)

Example:
WindowFrame = CGRectMake(400, 225, 800, 600);

This positions the Mac OS X window of Cocos2D to the screen location 400x225 (lower left corner of the window) and sets the window size to 800x600 pixels.