ironSource Extension for GameMaker Studio Documentation

Documentation **v2.5.0**, released with Extension **v2.5.0**
Valid unless newer revision of the documentation is specified on the Marketplace asset page.
All updated documentation versions bundled in extension package.
Please read this documentation before sending a support request.
Most, if not all common issues are already addressed here.

Table of Contents

I. Requirements

ironSource for GameMaker Studio requires either the Android or iOS Export Module(s), depending on your use case.

Below you can find a list of the software and tools that the extension has been confirmed as working without any issues:

- Game Maker Studio 2 runtime v2.2.5.378 with Android or iOS export module, under both the VM (non-YYC) and YYC targets
- For Android:
  - ironSource Android SDK 6.10.1
  - Android Studio-bundled OpenJDK
  - Android NDK (side by side) installation in Android Studio
  - Target SDK 28
  - Min. SDK 26
  - Compile SDK 28
  - Build Tools 28.0.0
  - Support Library 28.0.0
  - Target architectures tested: ARMv7, x86, ARM64, x86_64
- For iOS:
  - ironSource iOS SDK 6.8.1
  - macOS High Sierra (10.13.4)
  - XCode 11, New build system

The list above is for orientation purposes only; the project should work just fine with various other OS, and tools versions, as long as they are not too out-of-date.

II. Quick Start

Before you can get up and running, there are a few boxes you need to tick:

ironSource setup

- If you're starting completely fresh, head over to http://ironsrc.com and create an account

- Once you have set up your account, it's time to create an application on this page: https://platform.ironsrc.com/partners/applications

- From the same page specified above, you can grab your application's **APP KEY**. This is what we will use to initialize the extension inside GameMaker Studio

Android setup

- The ironSource extension for GameMaker Studio **already includes a directive to compile Google Play Services!** In case you are using either the official **GooglePlayServicesExtension** or **GoogleAnalyticsExt** please make sure to **delete** any conflicting code from "Inject to AndroidManifest.xml" and "Inject to Gradle dependencies" in any of the aforementioned extension's *Android* tab in its *Properties* window, in order to avoid compilation errors!

- Please check the **Gradle** and **AndroidManifest inject** settings in order to enable Google AdMob, as it requires manual entry of your app ID. After entering the app ID, please uncomment the corresponding sections.

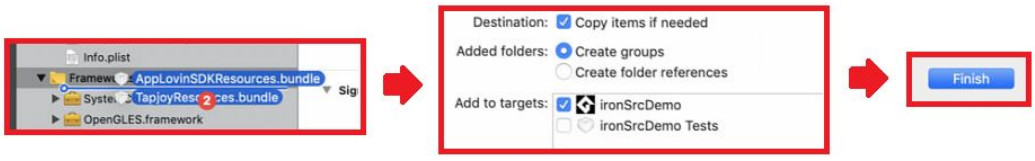
- In case you encounter a memory error while compiling, the issue can be fixed by going into *Preferences -> Android tab -> SDK tab -> Java Max Heap Size (GB)* and setting it to 4GB.

iOS setup

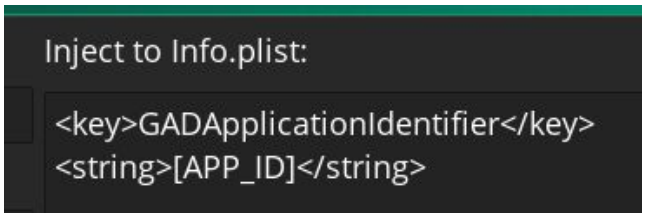
- The iOS 3rd party Ad network adapters are not bundled with the extension package downloaded from the YoYo Marketplace, due to the size limit. You can find the updated frameworks at the following link (current files version: **v2.5.0**):
https://mega.nz/file/i7BRmKiJ#vxNIYTN8GFT_ugaEWRObJ6FOIEAlSpcqjokB8tE82Sc

Extract the archive. You will end up with 2 zip files:

- "Copy to iOSSourceFromMac.zip"
  - Extract the files, and copy the **contents** of the archive into the iOSSourceFromMac folder of the extension. You should end up with a bunch of *.framework.zip* files.
- "Copy to XCode Frameworks.zip"
  - **Copy this archive <u>to your Mac</u>** after compiling your GM project.
  - Extract the files, then copy them onto your generated XCode project inside the Frameworks project directory, as shown in the image below. The reason you are required to do this is because of a **GM bug** that does not allow *.bundle files to go through the IDE nor the asset compiler, even though the IDE clearly states this is (and should be, really) supported. The bug has been reported and should get fixed in the following hundreds of weeks.

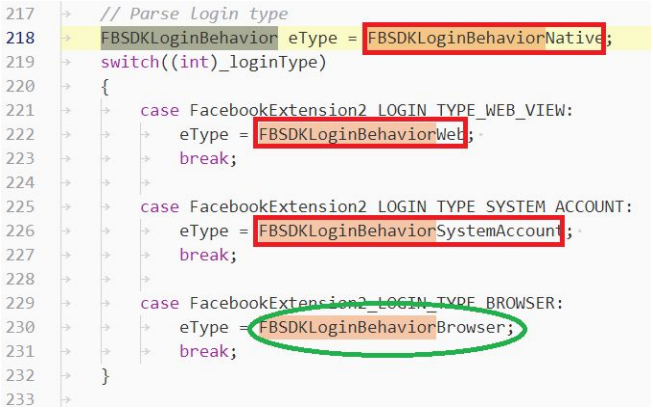


- **If you are using Google AdMob**, you need to **manually configure your AdMob App ID**. In order to do so, navigate to the Extension's iOS properties and change the Info.plist value accordingly. If you skip this step, your project will compile, but it will **instantly crash!**



- **If you are using the Facebook Extension, your project will not build in conjunction with this extension without a few modifications! This is because the Facebook Extension is (*surprise, surprise!*) terribly outdated. In order to make it work, please follow the steps below:**

1. Go to iOSSourceFromMac inside the FacebookExtension2 directory, and **delete** everything that is not "Bolts.framework" - that should be the only file that remains in the folder. The ironSource extension **already** includes up-to-date versions of the Facebook required frameworks which you just deleted, in order to mitigate this issue
2. Go to the iOSSource directory of FacebookExtension2, and open up **FacebookExtension2.mm**
    a. On line 715, **replace** *FBSDKLoginManagerRequestTokenHandler* **with** *LoginManagerLoginResultBlock*
    b. Around line 218, **replace all instances of** *FBSDKLoginBehavior[SOMETHING]* **with** *FBSDKLoginBehaviorBrowser*. Refer to the image on the next page for help.



3. Open up **FacebookExtension2.h**
    a. On line 82, **replace** *FBSDKLoginManagerRequestTokenHandler* **with** *LoginManagerLoginResultBlock*

These fixes should make FacebookExtension2 compile together with the ironSource extension just fine. This has nothing to do with the ironSource extension itself - if anything, the only “problem” with our wrapper extension is that we *actually updated the SDKs*. We are also not able to provide you directly with the patched FacebookExtension2, because of legal mumbo jumbo ¯_(ツ)_/¯

The only thing the community can do in order to avoid issues like these is leave feedback regarding the issues on the Facebook Extension page.

- If you use any other extension that has a LSApplicationQueriesSchemes plist key, read this!

Similar to Android's *Manifest* files, XCode uses *plist* files ("property lists"). You can find the setting for injecting custom keys in the plist file inside the *Extension package properties -> iOS tab*. If you use multiple extensions that inject the same keys in the plist file, **build errors will occur due to duplicate keys**. A notable example is the official Facebook extension, which uses the same "LSApplicationQueriesSchemes" dictionary that the ironSource adapter uses. If you do use the Facebook extension, please go to *Facebook Extension package properties -> iOS tab -> Inject to Info.plist* and **delete** everything.

Then, go to *ironSource Extension package properties -> iOS tab -> Inject to Info.plist* and locate the following block of code:

```
<!-- START Schemes for ironSource WITHOUT Facebook -->
<key>LSApplicationQueriesSchemes</key>
<array>
	<string>fb</string>
	<string>instagram</string>
	<string>tumblr</string>
	<string>twitter</string>
</array>
<!-- END App Queries Schemes -->
```

Replace that snippet of code with the following:

```
<!-- START Schemes for ironSource + Facebook -->
<key>LSApplicationQueriesSchemes</key>
<array>
	<string>fb</string>
	<string>instagram</string>
	<string>tumblr</string>
	<string>twitter</string>
	<string>fbapi</string>
```

```
        <string>fb-messenger-api</string>
        <string>fbauth2</string>
        <string>fbshareextension</string>
</array>
<!-- END App Queries Schemes -->
```

The code shown here in purple represents the values that the Facebook Extension itself had in its LSApplicationQueriesSchemes section. This type or merging should be done for **any** extension that already has the same key in its Info.plist file, and is a limitation of GameMaker itself, as it does not merge any keys when injecting to the plist file (it just copy-pastes everything you give it, which is why duplicates in extensions will result in duplicates inside the plist file itself, and thus errors).

- Unlike on Android, iOS ads will **not** pause the game! You need to handle this inside the game logic by listening for the "**adOpened**", "**adClosed**", "**rewardOpened**", "**rewardClosed**", "**offerwallOpened**", "**offerwallClosed**" events. If you do not handle those cases, not only will the game continue to run as if nothing is displaying over it, but some ad SDKs cause odd behaviour, where the game *still* receives inputs when you tap over ad content, even though it is displaying over the game.

- Lastly, if using the Google Play Services, there are some simple fixes to be done in order to ensure you don't get any errors:

- Right-click on *GooglePlayServicesExtension -> Open in Explorer*. Go to the *GooglePlayServicesExtension* folder, and delete both the *iOSSource* and *iOSSourceFromMac* folders.
- Right-click on *GooglePlayServicesExtension -> Properties -> iOS tab ->* delete everything inside System Frameworks and 3rd Party Frameworks.
    - Inside the same window, under the *General* tab, uncheck both *iOS* and *iOS(YYC)* in the targets list

- Expand *GooglePlayServicesExtension* by clicking on the + sign to the left, and double-click on *GoogleMobileAds.ext*. At the bottom list (“Copies To”), uncheck both *iOS* and *iOS(YYC)*

Whew, **done**! We are now ready to start using ironSource in our GameMaker project!

The sample project

These steps are here to help devs accommodate their existing projects in order to be able to use this extension. An already-set-up example is provided inside the extension itself, which you can use in order to test out the features.

In order to try out the sample project, simply **Import all assets** from the extension package into an **empty** project. Then:

- If targeting iOS, download the required frameworks from the link specified in the **iOS setup** section
- Set your App ID inside **obj_ironSrc**’s **Create** event

III. Extension Documentation

The extension is made to be as easy to use as possible whilst still being as packed with features as it can get. Inside you will find support for **all** of the ironSource callbacks, along with some helper functions/fixes for GM to make our life easier.

Although ironSource provides many callbacks for events regarding various ad formats, you do not *really* need to use all of them. However, for the advanced user's convenience, everything that ironSource exposes to native Java and Objective-C, the extension forwards to GameMaker.

The minimal approach

We'll now go through all essential functions for working with all kinds of ads, and their specific callbacks. Everything documented in here is also presented in the sample project, so be sure to check it out!

GDPR consent

Users living in the E.U. now have to give consent to process their data in case they want to see more tailored ads. In the case of those users, the programmer of the application must tell ironSource about their explicit consent before initialization. This is done with the following function:

```
ironSrc_setConsent(consent flag);
```

Tells ironSource whether it can serve tailored ads

`consent flag` - either true or false

This function should be called before initialization of the ironSource extension. It tells ironSource whether or not it can process the user's GAID/IDFA (Advertiser IDs) in order to show them more relevant ads. If the user is outside of the EU, just call this function with the flag set to true. For

EU citizens, you must show a dialogue where you ask the user whether or not they wish to share their info with the Ad providers in order to get ads better targeted to them. You then call this function with the appropriate value (true, or false). You should also store the user's answer, so that on subsequent runs, you just call the function with the last answer stored, instead of asking the user over and over on every run of the app.
It is also recommended that you implement a reliable way of detecting whether or not the user is from the EU (the sample project includes a rudimentary check script involving the country code reported by the operating system).
For more information about mediating through ironSource vs the EU GDPR, check out
https://developers.ironsrc.com/ironsource-mobile/general/making-sure-youre-compliant-post-gdpr/#step-2

Initialization

Once you've got your appKey and called the GDPR-compliance function, it's time to initialize so that we can start working with ads. To do that, just call:

```
ironSrc_init(appKey);
```
Initializes ironSource

Banner Ads

```
ironSrc_bannerDefaultPosition(pos);
```
Sets the position where a banner will be created.
Argument should be BANNER_BOTTOM or BANNER_TOP.
Call this **before** creating a new banner!

In case you need to create a banner in the other screen position, first **remove** the current banner, then call this function to set up the position, then create a new banner.

`ironSrc_getBannerDefaultPosition();`
Returns BANNER_BOTTOM or BANNER_TOP, as set previously. Default value is BANNER_BOTTOM.

`ironSrc_createBanner();`
Creates a banner ad view and loads the ad content

`ironSrc_hideBanner();`
Hides a banner ad view, but does not remove it
This function is very useful in situations where you need to not display the banner for a few screens, but then have it back up really fast
In order to unhide the banner, just call `ironSrc_createBanner();` again
That will not create a new banner, instead, it will bring back the previously hidden one (provided that a banner had been created and hidden)

`ironSrc_removeBanner();`
Destroys the banner ad view
Calling `ironSrc_createBanner();` after this will create a completely new banner ad

`ironSrc_bannerWidth();`
Gets the banner width (if one exists), in **screen pixels**

`ironSrc_bannerHeight();`
Gets the banner height (if one exists), in **screen pixels**

`ironSrc_createBannerWithOffset(offset);`
Same as the default banner creation function, but it makes the banner "float" from the bottom of the screen at exactly `offset` pixels.
This function is **iOS ONLY** and is useful for devices such as the iPhone X, where you may want the banner to not be exactly stuck to the bottom bezel.
Please note that for this to work properly, you need to **not** already have a banner on-screen, or destroy the previous banner!

Full Screen Ads

`ironSrc_isInterstitialReady();`
Returns whether or not an Interstitial Ad can be displayed

`ironSrc_isRewardedVideoAvailable();`
Returns whether or not an Rewarded Video can be displayed

`ironSrc_isOfferwallAvailable();`
Returns whether or not an Offer Wall can be displayed

`ironSrc_loadInterstitial();`
The extension autoloads ad content. However, if Interstitials are not loading, you may explicitly ask for one by using this function

`ironSrc_showInterstitial();`
Displays an Interstitial Ad

`ironSrc_showRewardedVideo();`
Displays a Rewarded Video

```
ironSrc_showOfferwall();
```
Displays an Offer Wall

Debugging

```
ironSrc_setDebug(enable debugging?);
```
Configures the extension to enable or disable detailed reporting of mediated SDKs information in the application logs. Useful for tracking down issues with SDK configurations.

Callbacks / Events

We will now cover the *essential* events/callbacks that the ironSource extension calls inside the a GameMaker application. These events should be everything that is needed to build the ad-related logic into your game. Later on in the document, we will also cover the extra events that ironSource provides for certain situations.

The example project responds to all events created by the ironSource extension. As for most extensions, asynchronous events are handled in the not-so-appropriately-named "**Social**" event. An ironSource event is presented in the form of a **Social** event with a field named "**type"** which is always equal to "**ironSrc**". The project shows how you can use the async events in 2 ways:

- By directly reading from the **async_load** map inside of the **Social** event (implemented inside **obj_ironSrc** in the sample project). You can differentiate between all of the ironSource events by reading the "**id**" field, as shown in the project. Bear in mind to check whether or not the "**type**" field is set to "**ironSrc**" first
- By implementing the logic inside of **callback functions** (which in the sample project, are called by **obj_ironSrc**, which forwards the **async_load** data to function calls). The sample callbacks are located inside the *Scripts -> Callbacks* folder

You are free to use any of the two approaches, either directly managing events in the Social async event, or by relaying the data to callback scripts.

We will now go through the essential events and their callbacks, along with any extra data that they might expose:

ironSource Event ID ("id" field)	Corresponding Callback script (as implemented in the sample project)
bannerLoaded *Indicates that a Banner has been loaded.*	`ironSrc_bannerLoaded()`
bannerLoadFailed *Indicates that a Banner couldn't load.*	`ironSrc_bannerFailedToLoad()`
bannerClicked *Indicates that the user has tapped the Banner.*	`ironSrc_bannerClicked()`
adLoaded *Indicates an Interstitial ad has loaded.*	`ironSrc_adLoaded()`
adClicked *Indicates that the user has tapped the call-to-action button inside of an Interstitial.*	`ironSrc_adClicked()`

adLoadFailed *Indicates that an Interstitial ad couldn't load.*	`ironSrc_adFailedToLoad()`
adOpened *Indicates that an Interstitial ad has been opened on the user's screen.* This event is important for properly pausing the game when showing ads!	`ironSrc_adOpened()`
adClosed *Indicates that the currently opened Interstitial ad has been closed.* This event is important for unpausing the game when closing ads!	`ironSrc_adClosed()`
adShowFailed *An Interstitial ad couldn't be shown.*	`ironSrc_adFailedToShow()`
rewardAvailable *The game is ready to show a Rewarded ad.*	`ironSrc_rewardAvailable()`
rewardUnavailable *There is no Rewarded adready, so the game can't show one.*	`ironSrc_rewardUnavailable()`

rewardOpened *Indicates that a Rewarded ad has been opened on the user's screen.* This event is important for properly pausing the game when showing ads!	`ironSrc_rewardOpened()`
rewardClosed *Indicates that a Rewarded ad has been closed.* This event is important for unpausing the game when closing ads!	`ironSrc_rewardClosed()`

rewardReceived
The Rewarded Ad objective has been passed and the game should receive the reward.
 	
Additional parameters:
"**reward**" = Name of the reward
"**amount**" = Number of "reward" to receive in-game
 	
These fields are configurable via the ironSource control panel. This mechanism allows for "dynamic rewards", where you can configure what and how much of something your users get without needing to update the game each time you make a change.
	`ironSrc_rewardReceived(`
	`  reward name = string,`
	`  reward amount = real`
	`)`
	
	Additional parameters documentation presented on the left.

rewardShowFailed *A Rewarded ad couldn't be shown.*	`ironSrc_rewardFailedToShow()`

rewardClicked
The user has tapped the call-to-action button inside of the Rewarded ad.
 	
Additional parameters:
"**reward**" = Name of the reward
"**amount**" = Number of "reward" to receive in-game
 	
*These fields work in the **same way** as for **rewardReceived**, **BUT** this event only triggers when the user actually **taps** the button at the end of the ad! Use this if you only want to reward the user when they actually press that button.*
	`ironSrc_rewardClicked(`
	`  reward name = string,`
	`  reward amount = real`
	`)`

offerwallAvailable *The game is ready to display an Offer Wall.*	`ironSrc_offerwallAvailable()`
offerwallUnavailable *Offer Walls are currently unavailable and the game cannot show one.*	`ironSrc_offerwallUnavailable()`
offerwallOpened *An Offer Wall ad has been opened.* This event is important for properly pausing the game when showing ads!	`ironSrc_offerwallOpened()`
offerwallClosed *The Offer Wall ad has been closed by the user.* This event is important for unpausing the game when closing ads!	`ironSrc_offerwallClosed()`

offerwallShowFailed *An Offer Wall couldn't be shown.*	`ironSrc_offerwallFailedToShow()`
offerwallCreditFailed *The Offer Wall failed to credit the user.*	`ironSrc_offerwallFailedToCredit()`

offerwallCredited
An Offer Wall offer has been completed and the game should now reward the user.
 	
Additional parameters:
"**credits**" = The amount of [your in-game reward] to reward
"**totalCredits**" = Total "credits" earned by the user from Offer Walls
"**isTotal**" = true or false. In some cases, ironSource can't tell how many credits you've got in THIS session. In those cases, you won't be able to increment the in-game currency; rather, ironSource sets both the credits and total credits fields to the same value, and also notifies us via a flag. When this occurs, instead of incrementing the currency, we just set the value to what ironSource tells us in any of the fields.
We only need to do this if the "isTotal" flag is set to TRUE.
	`ironSrc_offerwallCredited(`
	`  credits = real,`
	`  totalCredits = real,`
	`  isTotal = boolean`
	`)`
	
	Additional parameters documentation presented on the left.

As mentioned in the table, it is very important to take ...**Opened** and ...**Closed** events into accounts in order to implement a **pause** mechanic so that the game does not:

- Continue playing sounds / running gameplay mechanics under the ad
- Receive touch inputs **through** the ads themselves (as noted with some of the mediated ad networks on iOS)

In the following table, we will go through *optional* events that are not 100% necessary for implementing the game logic around ads, but are nice to have for specific situation that the developer may encounter.

Optional callbacks table:

ironSource Event ID ("id" field)	Corresponding Callback script (as implemented in the sample project)
bannerPresented *Indicates that a Banner ad has appeared on screen. This is called after bannerOpened, and after every refresh of the banner ad.*	`ironSrc_bannerPresented()`
ironSrc_bannerDismissed *Indicates that a banner has been dismissed (i.e. disappeared from the banner view). This does not necessarily mean that the banner ad view has been closed, just that the current banner ad has ended.*	`ironSrc_bannerDismissed()`
ironSrc_bannerLeftApp *Indicates that a banner has left the application. This event does not get called for all Banner ad adapters.*	`ironSrc_bannerLeftApp()`

adShown *Called when an Interstitial Ad is displayed on the screen. You can consider this to be a redundant event for the more reliable adOpened.*	`ironSrc_adShown()`
rewardStarted *Indicates that a Rewarded ad has begun playing back content on the user's device.*	`ironSrc_rewardStarted()`
rewardEnded *Called when a Rewarded ad has stopped playing back content.*	`ironSrc_rewardEnded()`

It is worth noting that the **...Started** and **...Ended** events/callbacks are **not** a reliable replacement for the **...Opened** and **...Closed** events/callbacks! These *optional* events are only provided inside the GameMaker extension for the advanced programmer's convenience. In order to provide as much functionality as possible, no assumptions have been made regarding what is and what isn't useful out of the ironSource SDK. As such, all possible callbacks have been implemented in this extension.

Important notice

GameMaker "*likes*" to sometimes effectively **lose** the **async_load** data **before** the event actually fires from an extension. This will effectively crash the game. In order to go around this issue, we check whether or not **async_load actually exists** before parsing it. Please check the **sample project** for an example (inside **obj_ironSrc**'s **Social** event).

Support

For any issues that might arise when using the extension or any errors in the documentation, do not hesitate to report it on the Marketplace page! We're very responsive to feedback and wish to provide our customers with the best experience as possible when it comes to using our assets.

Happy coding!