AdMob Extension

Contents

Extension’s Features

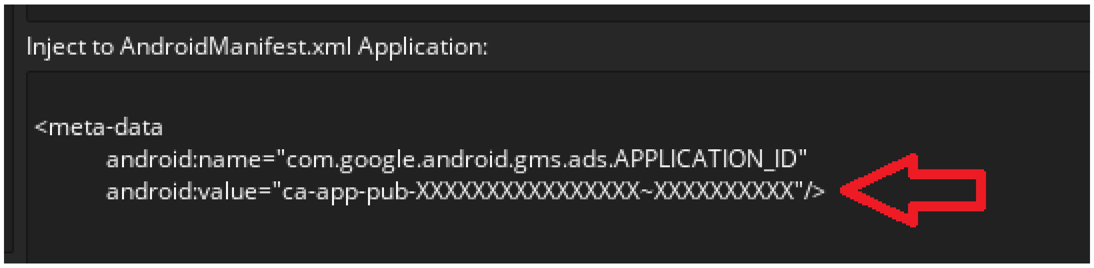
- Enable test mode (development)
- Request GDPR Consent
- Target ads to your specific audience
    - Under-age
    - Children
    - Max Rating system
- Use 3 distinct types of ads:
    - Banners (7 different banner types)
    - Interstitial
    - RewardedVideos
- Allows ad volume control (including mute toggle)

Setup

The AdMob extension is to be used alongside your google AdMob account (web page). All the required personal ad ids and consent messages should be handled through there.

1. For GDPR consent you should look to follow the options below:
   AdMob Console → **Privacy and Messaging** → **Go to funding choices** → (select your project) → **Create** (new message) → **EU Consent** → fill all the necessary details.

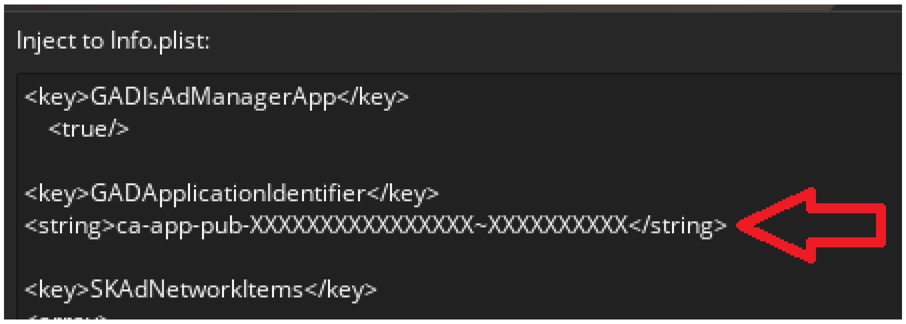
2. For setting up the AdMob extension with your new project go to:
   AdMob (extension options) → **Extra Platforms** (Android) → **Inject to AndroidManifest.xml Application** → Change to your own application ID:



```
Inject to AndroidManifest.xml Application:

<meta-data
        android:name="com.google.android.gms.ads.APPLICATION_ID"
        android:value="ca-app-pub-XXXXXXXXXXXXXXXX~XXXXXXXXXX"/>
```

   AdMob (extension options) → **Extra Platforms** (iOS) → **Inject to Info.plist** → Change to your own application ID:



```
Inject to Info.plist:

<key>GADIsAdManagerApp</key>
   <true/>

<key>GADApplicationIdentifier</key>
<string>ca-app-pub-XXXXXXXXXXXXXXXX~XXXXXXXXXX</string>

<key>SKAdNetworkItems</key>
```

3. To build and deploy to iOS it's required for the user to install CocoaPods (installation guide)

Quick Start Guide

This section aims to deliver an easy and simple set of examples that should help you migrate your project from a previous version of the AdMob extension into this new, updated version. The first thing to notice is that all previously named “**GoogleMobileAds_***” functions have been renamed to “**AdMob_***” providing a more compact naming convention which is consistent with the Google API function names outside the Game Maker Studio environment.

Initialization

The first thing to look for when creating an AdMob GMS2 project is initializing your API.

[Old Version]

```
// On the previous version you would need to provide application id and
// an interstitial id to the API initialization function.
GoogleMobileAds_Init(interstitialId, applicationId);

// Optionally you could set your device to test mode to use test ads instead
// of ‘live’ ones. You would provide the enable flag and the device id.
GoogleMobileAds_UseTestAds(enable, deviceId); // <----- this is OPTIONAL
```

[New Version]

```
// If you are using Test Ads the ‘AdMob_SetTestDeviceId’ function needs to be
// called BEFORE initialization (no arguments are required).
AdMob_SetTestDeviceId(); // <----- this is OPTIONAL (development only)


// In the new version you just need to call the initialization method without
// any arguments, as the extension now gets the application id from the
// “extension options” automatically (Setup, 2.), and the ad ids are setup on
// a per ad-type basis.
AdMob_Initialize();
```

Consent (GDPR)

One of the first concepts requiring your attention when setting up a project using AdMob is that, depending on the user's geographic location, you might need to ask for permission to use their personal information to personalize the ads that are shown.

[Old Version]

In previous versions you could ask the user for their consent to collect data for personalized ads using one of the following setups:

```
// You could forcefully show the consent form providing a privacy URL and
// options for personalized ads, non-personalized ads and ad-free version of
// your game.
GoogleMobileAds_ConsentFormShow(privacyPolicy, personalized, nonPersonalized,
adFree);
```

OR

```
// You could show the consent form ONLY if the user hasn’t yet answered it.
GoogleMobileAds_ConsentUpdate(publisherId, privacyPolicy, personalized,
nonPersonalized, adFree);
```

[New Version]

The new version has added functionality that requires you to:

1. check the consent **status**;
2. **load** the consent form before showing it to the user;
3. **show** the form (this replaces the old `’GoogleMobileAds_ConsentFormShow’`)

The first steps towards requesting user consent are the following:

```
// First of all we need to set the Consent Mode, which can be used for
// debugging purposes to run tests as a  user in a different geographic area;
// more info: Consent Mode. This function will generate an ASYNC SOCIAL EVENT.
AdMob_Consent_RequestInfoUpdate(AdMob_Consent_Mode_PRODUCTION);
```

After setting the consent mode we need to add some code to the ASYNC SOCIAL EVENT, to check for the events generated by the ‘AdMob_Consent_RequestInfoUpdate’ function call. Here is what each event allows us to do:

1. **“AdMob_Consent_OnRequestInfoUpdated”** → we check status & load form;
2. **“AdMob_Consent_OnLoaded”** → we show the consent form;
3. **“AdMob_Consent_OnShown”** → we get the user’s consent type;

Given below is the fully documented template code that you can use in your ASYNC SOCIAL EVENT:

```
// Early exit if there is no 'type' key defined
if (!ds_map_exists(async_load, "type")) exit;

// All the events triggered by the AdMob extension have a “type” key
// containing a string that starts with “AdMob_”.
switch(async_load[? "type"])
{
      // AdMob_Consent_RequestInfoUpdate() succeeded
      case "AdMob_Consent_OnRequestInfoUpdated":

            // Now we need to get the consent Status, this will tell us if we
            // are required to ask the user for GDPR consent.
            if (AdMob_Consent_GetStatus() == AdMob_Consent_Status_REQUIRED)

                  // Since we are REQUIRED, we now need to load the consent
                  // form before we can show it. For this we use the function
                  // below (more info: AdMob_Consent_Load). This function call
                  // will also generate an ASYNC SOCIAL EVENT.
                  AdMob_Consent_Load();
            break;



      // AdMob_Consent_RequestInfoUpdate() failed
      case "AdMob_Consent_OnRequestInfoUpdateFailed":

            // This means there was a problem while setting the consent
            // mode. Here we can add some code to deal with it.
            break;

      // AdMob_Consent_Load() succeeded
      case "AdMob_Consent_OnLoaded":

            // We have successfully loaded the consent form and we are now
            // ready to show it to the user (more info: AdMob_Consent_Show)
            AdMob_Consent_Show();
            break;

      // AdMob_Consent_Load() failed
      case "AdMob_Consent_OnLoadFailed":

            // This means there was a problem while loading the consent
            // form. Here we can add some code to deal with it.
            break;

      // AdMob_Consent_Show() succeeded and the user already answered it
      case "Consent_onShown":

            // At this point we now have the consent information from the
            // user. We can use both the GetStatus and GetType functions to
            // get the obtained information (more info:
            // AdMob_Consent_GetStatus and AdMob_Consent_GetType)
            global.ConsentStatus = AdMob_Consent_GetStatus();
            global.ConsentType = AdMob_Consent_GetType();
            break;
}
```

Targeting

This new version of the AbMob extension allows the developer to target ads to a specific audience. This could already be done in previous versions but the new version now has more features specifically for targeting.

[Old Version]

In previous versions the developer could mark a user as underage ads using the following function:

```
//  The function below classifies users as under age and only non-personalised
// ads can be shown, regardless of any other setting, and the consent form
// will not be shown.
GoogleMobileAds_ConsentSetUserUnderAge(true);
```

[New Version]

In the new version of the extension the developer is presented with more features that will help target the right kind of ads to the right kind of audience:

```
// The function below allows the developer to enable or disable ads for
// under-age users (much like the old version). Note that it is up to the
// developer to identify the age of the consumer.
AdMob_Targeting_UnderAge(true);

// The function below allows the developer to enable or disable ads for
// children. Note that it is up to the developer to identify the age of the
// consumer.
AdMob_Targeting_COPPA(true);

// The new version of the AdMob extension allows for even further control
// over the filtering of the ads being displayed to the user. The function
// below allows the developer to set a maximum content rating of the ads to be
// displayed (info: AdMob_Targeting_MaxAdContentRating and Content Rating.
AdMob_Targeting_MaxAdContentRating(AdMob_ContentRating_GENERAL);
```

Banner Ads

There have been a lot of API changes since the previous version, that said let's jump in on a quick set up for adding banner ads to your application.

[Function Mappings]

Even though not all old functions map perfectly into the new API the following list should get you started with some of the changes (check the section below for more details):

- **AdMob_Banner_Init(...)** initializes the banner ads with a unique id.
- **AdMob_Banner_Create(...)** replaces GoogleMobileAds_AddBanner(...)
  - triggers the Social event "AdMob_Banner_OnLoaded" if creation succeeds
  - triggers the Social event "AdMob_Banner_OnLoadFailed" if creation failed
- **AdMob_Banner_Move(...)** replaces GoogleMobileAds_MoveBanner(...)
- **AdMob_Banner_Remove()** replaces GoogleMobileAds_RemoveBanner()
- **AdMob_Banner_Show()** replaces GoogleMobileAds_ShowBanner()
- **AdMob_Banner_Hide()** replaces GoogleMobileAds_HideBanner()

[New Version]

The first step we should take towards adding banners is to use a manager object with the following CREATE EVENT (remember that you need to initialize the API first - Initialization)

```
// After API initialization we can proceed to initialize the banner options.
// This function requires a unique ad block id string that can be obtained
// from the google developer console (more info: AdMob_Banner_Init).
// This function will generate an ASYNC SOCIAL EVENT.
var banner_id = "ca-app-pub-3940256099942544/6300978111"
AdMob_Banner_Init(banner_id);
```

The second step requires us to make sure the API was initialized, so for this example we will use the ASYNC SOCIAL EVENT to check for the events generated by the 'AdMob_Initialize' function call.

```
// Early exit if there is no 'type' key defined
if (!ds_map_exists(async_load, "type")) exit;

// All the events triggered by the AdMob extension have a "type" key
// containing a string that starts with "AdMob_".
switch(async_load[? "type"])
{
      // AdMob_Initialize() finished initializing the API
      case "AdMob_OnInitialized":

            // Now that we are sure that the API was initialized we can create
            // our new banner on the device display.

            // The function below will create a new banner of a given type,
            // allowing the developer to also select the position of the
            // banner (this can be either at the top or the bottom of the
            // screen (more info: AdMob_Banner_Create and Banner_Type).
            var banner_type = AdMob_Banner_ADAPTIVE;
            var bottom = true;
            AdMob_Banner_Create(banner_type, bottom)
            break;

      // AdMob_Banner_Create() succeeded
      case "AdMob_Banner_OnLoaded":

            // At this point we should now have a banner on the screen.
            break;

      // AdMob_Banner_Create() failed
      case "AdMob_Banner_OnLoadFailed":

            // At this point there was a problem while creating the banner.
            // Here we can add some code to deal with it.

            // NOTE: Don't try to create a banner here because it can lead to
            // an infinite loop if the banner creation fails constantly.
            break;
}
```

Interstitial Ads

Interstitial ads are ads that will fill up the entire screen and need to be dismissed in order for the application to continue execution. Initial setup for interstitials is very similar to Banner ads (more info: Banner Ads). Let's look at some code to quickly set up interstitial ads for your application.

[Function Mappings]

Even though not all old functions map perfectly into the new API, the following list should get you started with some of the changes (check the section below for more details):

- **AdMob_Interstitial_Init(...)** initializes the interstitial ads with a unique id.
- **AdMob_Interstitial_Load()** replaces GoogleMobileAds_LoadInterstitial()
  - triggers the Social event "AdMob_Interstitial_OnLoaded" if load succeeds
  - triggers the Social event "AdMob_Interstitial_OnLoadFailed" if load fails
- **AdMob_Interstitial_IsLoaded()** replaces GoogleMobileAds_InterstitialStatus()
- **AdMob_Interstitial_Show()** replaces GoogleMobileAds_ShowInterstitial(...)
  - triggers the Social event "AdMob_Interstitial_OnFullyShown" if show succeeds
  - triggers the Social event "AdMob_Interstitial_OnShowFailed" if show fails
  - triggers the Social event "AdMob_Interstitial_OnDimissed" upon dismissing

[New Version]

The first step we should take towards adding interstitials is to use a manager object with the following CREATE EVENT (remember that you need to initialize the API first - Initialization)

```
// After API initialization we can proceed to initialize the interstitial
// options. This requires a unique ad block id string that can be obtained
// from the google developer console. (more info: AdMob_Interstitial_Init).
// This function will generate a SOCIAL ASYNC EVENT.
var interstitial_id = "ca-app-pub-3940256099942544/1033173712"
AdMob_Interstitial_Init(interstitial_id);
```

The second step requires us to make sure the API was initialized so for this example we will use the ASYNC SOCIAL EVENT, to check for the events generated by the 'AdMob_Initialize' function call.

```
// Early exit if there is no 'type' key defined
if (!ds_map_exists(async_load, "type")) exit;

// All the events triggered by the AdMob extension have a "type" key
// containing a string that starts with "AdMob_".
switch(async_load[? "type"])
{
```

```
// AdMob_Initialize() finished initializing the API
case "AdMob_OnInitialized":

      // Now that we are sure that the API got initialized we can load
      // a new interstitial ad.(more info: AdMob_Interstitial_Load).
      // This function will generate an ASYNC SOCIAL EVENT.
      AdMob_Interstitial_Load();
      break;

// AdMob_Interstitial_Load() succeeded
case "AdMob_Interstitial_OnLoaded":

      // At this point we should now have the interstitial ad loaded and
      // and we can check that using the ´AdMob_Interstitial_IsLoaded´
      // function. We are now ready to show the interstitial ad to the
      // user (more info: AdMob_Interstitial_Show). This function will
      // generate an ASYNC SOCIAL EVENT.
      AdMob_Interstitial_Show();
      break;

// AdMob_Interstitial_Load() failed
case "AdMob_Interstitial_OnLoadFailed":

      // At this point there was a problem while loading the
      // interstitial ad. Here we can add some code to deal with it.

      // NOTE: Don't try to reload the interstitial ad here because
      // it can lead to an infinite loop.
      break;

// AdMob_Interstitial_Show() succeeded
case "AdMob_Interstitial_OnFullyShown":

      // At this point the interstitial ad is on screen and the user is
      // looking at it. Note that at this point in time your game is
      // paused and will remain paused until the interstitial gets
      // dismissed.
      break;

// AdMob_Interstitial_Show() failed
case "AdMob_Interstitial_OnShowFailed":

      // At this point the interstitial ad failed to get shown to the
      // user. You can add code to deal with the problem here.

      // NOTE: Don't try to reload/show the interstitial ad here
      // because it can lead to an infinite loop.
      break;

// Interstitial got dismissed
case "AdMob_Interstitial_OnDismissed":

      // At this point the interstitial ad got dismissed by the user and
```

```
                // the game logic is running again.
                break;

}
```

Rewarded Video Ads

Rewarded video ads are ads that will fill up the entire screen and play a video, at the end of which the user can be rewarded in some way, and similar to interstitial ads they also need to be dismissed in order for the application to continue execution. Rewarded Video ads setup is very similar to other previous ads (more info: Banner Ads and Interstitial Ads). Let's look at some code to quickly set up rewarded video ads on your application.

[Function Mappings]

Even though not all old functions map perfectly into the new API the following list should get you started with some of the changes (check the section below for more details):

- **AdMob_RewardedVideo_Init(...)** initializes the interstitial ads with a unique id.
- **AdMob_RewardedVideo_Load()** replaces GoogleMobileAds_LoadRewardedVideo()
    - triggers the event "AdMob_RewardedVideo_OnLoaded" if load succeeds
    - triggers the event "AdMob_RewardedVideo_OnLoadFailed" if load fails
- **AdMob_RewardedVideo_IsLoaded()** replaces GoogleMobileAds_RewardedVideoStatus()
- **AdMob_RewardedVideo_Show()** replaces GoogleMobileAds_ShowRewardedVideo(...)
    - triggers the event "AdMob_RewardedVideo_OnFullyShown" if show succeeds
    - triggers the event "AdMob_RewardedVideo_OnShowFailed" if show fails
    - triggers the event "AdMob_RewardedVideo_OnDimissed" upon dismissing
    - triggers the event "AdMob_RewardedVideo_OnReward" when reward conditions are met.

[New Version]

The first step we should take towards adding rewarded videos is to use a manager object with the following CREATE EVENT (remember that you need to initialize the API first - Initialization)

```
// After API initialization we can proceed to initialize the rewarded video
// options. This requires an unique ad block id string that can be obtained
// from the google developer console. (more info: AdMob_RewardedVideo_Init).
// This function will generate an ASYNC SOCIAL EVENT.
var rewarded_id = "ca-app-pub-3940256099942544/1033173712"
AdMob_RewardedVideo_Init(rewarded_id);
```

The second step requires us to make sure the API was initialized. For this example we will use the ASYNC SOCIAL EVENT, to check for the events generated by the 'AdMob_Initialize' function call.

```
// Early exit if there is no 'type' key defined
if (!ds_map_exists(async_load, "type")) exit;

// All the events triggered by the AdMob extension have a "type" key
// containing a string that starts with “AdMob_”.
switch(async_load[? "type"])
{
      // AdMob_Initialize() finished initializing the API
      case "AdMob_OnInitialized":

            // Now that we are sure that the API got initialized we can load
            // a new rewarded video ad.(more info: AdMob_RewardedVideo_Load).
            // This function will generate an ASYNC SOCIAL EVENT.
            AdMob_RewardedVideo_Load();
            break;

      // AdMob_RewardedVideo_Load() succeeded
      case "AdMob_RewardedVideo_OnLoaded":

            // At this point we should now have the rewarded ad loaded and
            // and we can check that using the ´AdMob_RewardedVideo_IsLoaded´
            // function. We are now ready to show the rewarded video ad to the
            // user (more info: AdMob_RewardedVideo_Show). This function will
            // generate an ASYNC SOCIAL EVENT.
            AdMob_RewardedVideo_Show();
            break;

      // AdMob_RewardedVideo_Load() failed
      case "AdMob_RewardedVideo_OnLoadFailed":

            // At this point there was a problem while loading the
            // interstitial ad. Here we can add some code to deal with it.

            // NOTE: Don’t try to reload the interstitial ad here because
            // it can lead to an infinite loop.
            break;

      // AdMob_RewardedVideo_Show() succeeded
      case "AdMob_RewardedVideo_OnFullyShown":

            // At this point the rewarded video ad is playing and the user is
            // looking at it. Note that at this point in time your game is
            // paused and will remain paused until the rewarded video ad gets
            // dismissed.
            break;

      // AdMob_RewardedVideo_Show() failed
      case "AdMob_RewardedVideo_OnShowFailed":

            // At this point the rewarded video ad failed to get shown to the
            // user. You can add code to deal with the problem here.

            // NOTE: Don’t try to reload/show the rewarded video here
            // because it can lead to an infinite loop.
            break;

      // RewardedVideo got dismissed
```

```
        case "AdMob_RewardedVideo_OnDismissed":

              // At this point the rewarded video ad got dismissed by the user
              // and the game logic is running again.
              break;

        // RewardedVideo triggered the reward event
        case "AdMob_RewardedVideo_OnReward":

              // At this point the user watched enough of the rewarded video and
              // can be rewarded for it. Here we can add the reward code.
              show_debug_message("You got 1000 points").
              break;

  }
```

Functions

Some of the provided functions generate **Social Async** callbacks. In these cases the extension populates the **async_load** map with a "**type**" key that will contain the specific identifier of a given callback. Note that all AdMob extension related callbacks start with “*AdMob_*”.

`AdMob_Initialize()`

Description: This function initialises the Google AdMob API and should be called at the start of your game.

Returns: N/A

Triggers: Social Async Event

type: "AdMob_OnInitialized"

Replaces: GoogleMobileAds_Init(interstitialId, applicationId)

`AdMob_SetTestDeviceId()`

Description: This function tells the app to use test ads instead of “live” ads, essential for testing whether your ads work without generating potentially fraudulent click-throughs. If you need to use test mode, call this function BEFORE calling ‘AdMob_Initialize‘.

Returns: N/A

Replaces: GoogleMobileAds_ConsentDebugAddDevice(id)

Consent Functions

`AdMob_Consent_RequestInfoUpdate(testing)`

Allows to set the mode of the consent request being used. This function allows you to debug different regions and EEA and NON-EEA and should be passed in as a 'AdMob_Consent_Mode_*' constant. This function should be called before 'AdMob_Consent_GetStatus' and 'AdMob_Consent_GetType' in order to get the correct output from both functions.

Params:

{Consent Mode} **testing**

Returns: N/A

Triggers: Social Async Event

<if succeeds>

type: "AdMob_Consent_OnRequestInfoUpdated"

<if fails>

type: "AdMob_Consent_OnRequestInfoUpdateFailed"

`AdMob_Consent_GetStatus()`

Returns a constant representing the current status of the GDPR consent.

Returns: *{Consent Status}* The current status of the GDPR consent for this application.

Notes: If the status equals 'AdMob_Consent_Status_REQUIRED' then we need to ask the user for GDPR consent.

Replaces: GoogleMobileAds_ConsentUserInEEA() and adds extra functionality.

`AdMob_Consent_GetType()`

Returns the answer given by the user to a previous GDPR consent request.

Returns: *{Consent Type}* The constant representing the answer given by the user to the GDPR consent.

Notes: For EEA users the return value will always be 'AdMob_Consent_Type_UNKNOWN'.

`AdMob_Consent_IsFormAvailable()`

Checks whether or not the GDPR consent form is available on this device.

Returns: *{boolean}*

`AdMob_Consent_Load()`

Loads the consent form into memory so it can be displayed to the user. If you do not call this function before trying to show the GDPR consent, nothing will be shown.

Returns: N/A

Triggers: Social Async Event

<if succeeds>

type: "AdMob_Consent_OnLoaded"

<if fails>

type: "AdMob_Consent_OnLoadFailed"

`AdMob_Consent_Show()`

Shows the consent form to the user. If you do not call the 'AdMob_Consent_Load' function before trying to show the GDPR consent, nothing will be shown.

Returns: N/A

Triggers: Social Async Event

type: "AdMob_Consent_OnShown"

Replaces: GoogleMobileAds_ConsentFormShow(privacyPolicyURL, personalisedAds, noPersonalisedAds, adFree);

`AdMob_Consent_Reset()`

This function resets the consent status flag.

Returns: N/A

Targeting Functions

`AdMob_Targeting_COPPA(enable)`
Toggles on/off ads for children.

Params:

{bool} **enable**: true → on; false → off;

Default: false

Returns: N/A

`AdMob_Targeting_UnderAge(enable)`
Toggles on/off ads for under aged users.

Params:

{bool} **enable**: true → on; false → off;

Default: false

Returns: N/A

`AdMob_Targeting_MaxAdContentRating(maxRating)`
Allows for setting the maximum content rating of the ads to be displayed.

Params:

{Content Rating} **maxRating**

Default: AdMob_ContentRating_GENERAL

Returns: N/A

Banner Functions

`AdMob_Banner_Init(banner_id)`

This is a required initialization if you want to use banner ads. Allows to set up the unique string id for banner ads.

Params:

{string} **banner_id**: The unique id of the banner ad block.

Returns: N/A

`AdMob_Banner_Create(banner_type, true)`

This function can be used to create a banner at the bottom or top of the display. You supply a banner type and a boolean that determines if the banner should be placed at the bottom or at the top of the display.

Params:

{Banner Type} **banner_type**: The type of the banner to be displayed.
{boolean} **bottom**: Whether the banner should be placed at the bottom of the display.

Returns: N/A

Triggers: Social Async Event

<if succeeds>

type: "AdMob_Banner_OnLoaded"

<if fails>

type: "AdMob_Banner_OnLoadFailed"

Replaces: GoogleMobileAds_AddBanner(bannerAdId, sizeType)

`AdMob_Banner_Move(bottom)`

This function can be used to move a banner that has been previously added. You supply a boolean that will determine if the banner should be placed at the bottom or at the top of the display.

Params:

{boolean} **bottom**: Whether the banner should be placed at the bottom of the display.

Returns: N/A

Notes: No positional (x, y) information can be set.

Replaces: GoogleMobileAds_MoveBanner(display_x, display_y)

`AdMob_Banner_Remove()`

This function will remove the currently active banner from the app. If you call this function then want to show ads again, you must call the 'AdMob_Banner_Create' function first to add a new banner to the display.

Returns: N/A

Replaces: GoogleMobileAds_RemoveBanner()

`AdMob_Banner_Show()`

This function can be used to show the currently active, but hidden, banner ad block. When called, the banner will be shown to the user again and will be able to receive input. You can hide the banner again at any time using the 'AdMob_Banner_Hide' function.

Returns: N/A

Replaces: GoogleMobileAds_ShowBanner()

`AdMob_Banner_Hide()`

This function can be used to hide the currently active banner ad block. When called, the banner will be removed from the user's view and will no longer receive input. You can show the banner again at any time using the 'AdMob_Banner_Show' function.

Returns: N/A

Replaces: GoogleMobileAds_HideBanner()

`AdMob_Banner_GetWidth()`

This function can be used to get the width of the currently loaded banner ad block. The value returned is in pixels.

Notes: This function returns the width in screen pixels, it's up to the developer to convert the value to the correct scale according to the render target being used.

Returns: *{real}* The width of the ad.

Replaces: GoogleMobileAds_BannerGetWidth()

`AdMob_Banner_GetHeight()`

This function can be used to get the height of the currently loaded banner ad block. The value returned is in pixels.

Notes: This function returns the height in screen pixels, it’s up to the developer to convert the value to the correct scale according to the render target being used.

Returns *{real}* The height of the ad.

Replaces GoogleMobileAds_BannerGetHeight()

Interstitial Functions

`AdMob_Interstitial_Init(interstitial_id)`

This is a required initialization if you want to use interstitial ads. Allows you to set up the unique string id for interstitial ads.

Params:

{string} **interstitial_id** The unique id of the interstitial ad block.

Returns: N/A

`AdMob_Interstitial_Load()`

This function should be called when you want to load an interstitial ad. Calling it will send a request to the ad server to provide an interstitial ad, which will then be loaded into the app for display. This function does not show the ad, just stores it in memory ready for being shown. If you do not call this function before trying to show an ad, nothing will be shown. Note that you can check whether an interstitial is loaded or not using the function 'AdMob_Interstitial_IsLoaded'.

Returns: N/A

Triggers: Social Async Event

<if succeeds>

type: "AdMob_Interstitial_OnLoaded"

<if fails>

type: "AdMob_Interstitial_OnLoadFailed"

Replaces: GoogleMobileAds_LoadInterstitial()

`AdMob_Interstitial_IsLoaded()`

This function will return whether or not the interstitial ad is loaded.

Returns: *{boolean}* 1 if ad is loaded; 0 if ad is not loaded

Replaces: GoogleMobileAds_InterstitialStatus()

`AdMob_Interstitial_Show()`

This function will show the interstitial ad, if one is available and loaded. You can check whether an ad is available using the function 'AdMob_Interstitial_IsLoaded'. Note that while an interstitial is being shown, your app will be put into the background and will effectively be “paused”.

Returns: N/A

Triggers: Social Async Event

> **<if succeeds>**
>
> > type: "AdMob_Interstitial_OnFullyShown"
>
> **<if fails>**
>
> > type: "AdMob_Interstitial_OnShowFailed"
>
> Upon being **dismissed** another event is triggered:
>
> > type: "AdMob_Interstitial_OnDismissed"

Replaces: GoogleMobileAds_ShowInterstitial()

Rewarded Video Functions

`AdMob_RewardedVideo_Init(rewarded_id)`
This is a required initialization if you want to use rewarded video ads. Allows you to set up the unique string id for rewarded videos ads.

Params:

{string} **rewarded_id** The unique id of the rewarded video ad block.

Returns: N/A

`AdMob_RewardedVideo_Load()`
This function should be called when you want to load a rewarded video ad. Calling it will send a request to the ad server to provide a rewarded ad, which will then be loaded into the app for display. This function does not show the ad, just stores it in memory ready for showing. If you do not call this function before trying to show an ad, nothing will be shown. Note that you can check whether a rewarded video is loaded or not using the function 'AdMob_RewardedVideo_IsLoaded'.

Returns: N/A

Triggers: Social Async Event

<if succeeds>

type: "AdMob_RewardedVideo_OnLoaded"

<if fails>

type: "AdMob_RewardedVideo_OnLoadFailed"

Replaces: GoogleMobileAds_LoadRewardedVideo(reward_id);

`AdMob_RewardedVideo_IsLoaded()`
This function will return whether the rewarded video ad has been loaded or not.

Returns: *{boolean}* 1 if ad is loaded; 0 if ad is not loaded

Replaces: GoogleMobileAds_RewardedVideoStatus()

`AdMob_RewardedVideo_Show()`
This function will show the rewarded video ad, if one is available and loaded. You can check whether an ad has previously been loaded using the function

'AdMob_RewardedVideo_IsLoaded'. Note that while a rewarded video ad is being shown, your app will be put into the background and will effectively be “paused”.

Returns: N/A

Triggers: Social Async Event

<if succeeds>

type: "AdMob_RewardedVideo_OnFullyShown"

<if fails>

type: "AdMob_RewardedVideo_OnShowFailed"

Upon being **dismissed** another event is triggered:

type: "AdMob_RewardedVideo_OnDismissed"

<if reward time reached>

type: "AdMob_RewardedVideo_OnReward"

Replaces: GoogleMobileAds_ShowRewardedVideo()

Settings Functions

`AdMob_Settings_SetMuted(enable)`

This method provides control over muting the sound when playing rewarded video ads. This method will trigger a reload of the current Interstitial and RewardedVideo ads.

Params:

{boolean} **enable**: true → on; false → off;

Returns: N/A

`AdMob_Settings_SetVolume(amount)`

This method provides control over the sound's loudness when playing rewarded video ads. This method will trigger a reload of the current Interstitial and RewardedVideo ads.

Params:

{real[0-1]} **amount** The amount to set the volume to.

Returns: N/A

Constants

Banner Type

AdMob_Banner_NORMAL: 0
AdMob_Banner_LARGE: 1
AdMob_Banner_MEDIUM: 2
AdMob_Banner_FULL: 3
AdMob_Banner_LEADERBOARD: 4
AdMob_Banner_SMART: 5
AdMob_Banner_ADAPTIVE: 6

Content Rating

AdMob_ContentRating_GENERAL: 0
AdMob_ContentRating_PARENTAL_GUIDANCE: 1
AdMob_ContentRating_TEEN: 2
AdMob_ContentRating_MATURE_AUDIENCE: 3

Consent Status

AdMob_Consent_Status_UNKNOWN: 0
AdMob_Consent_Status_NOT_REQUIRED: 1
AdMob_Consent_Status_REQUIRED: 2
AdMob_Consent_Status_OBTAINED: 3

Consent Type

AdMob_Consent_Type_UNKNOWN: 0
AdMob_Consent_Type_NON_PERSONALIZED: 1
AdMob_Consent_Type_PERSONALIZED: 2

Consent Mode

AdMob_Consent_Mode_DEBUG_GEOGRAPHY_DISABLED: 0
AdMob_Consent_Mode_DEBUG_GEOGRAPHY_EEA: 1
AdMob_Consent_Mode_DEBUG_GEOGRAPHY_NOT_EEA: 2
AdMob_Consent_Mode_PRODUCTION: 3