Google Mobile Ads Manual

Contents

Introduction

This PDF manual is designed for you to use as a reference to the different Google Mobile Ads functions, and as such does *not* contain tutorials on how to set up the API in your games. If you wish information on setting up, general use, etc… then please see the following YoYo Games Knowledge Base articles:

- iOS and Android: Google Consent SDK
- iOS and Android: Google Mobile Ads Setup

This manual *only* covers Google Mobile Ads, as the rest of the Google Play Services are covered in the manual, and in the following YoYo Games Helpdesk documentation:

- Android: Google Play Services - Logging In
- Android: Google Play Services - Leaderboards
- Android: Google Play Services - Achievements

The Asynchronous Social Event

When using the Google Mobile Ads extension in your projects, you will be calling different functions that will trigger "callbacks" from the Google API. What this means is that certain functions will be run but won't actually give a result until sometime in the future, which could be the next step, or it could be a few seconds later. This result, when it comes, is called the "callback" and is the Google API responding to something you've done. This callback is dealt with in the **Asynchronous Social Event**.

This event will always have a DS map in the GML variable `async_load`, and this map can be parsed to get the required information. Each function will generate different callbacks, but they will all have certain key/value pairs in common, which we'll list here:

- **"id"** – This key is the ID of the event and will be the GML constant `GoogleMobileAds_ASyncEvent`.

- **"type"** – This is the event type key and it will hold a string with the type of event that has been triggered. For example, if it's an event to tell you a banner ad has been loaded, then the string will be "`banner_loaded`".

The rest of the key/value pairs in the map will depend on the function that triggered the Async Event, and you should check the individual functions listed in the rest of this manual for exact details.

Extension Functions

The rest of this manual is taken up with the different functions and constants that are included as part of the Google Mobile Ads Extension:

GoogleMobileAds_Init

Description

This function initialises the Google Mobile Ads API and should be called at the start of your game. The function requires you to supply the ID for any interstitial ad blocks that you want to use (you can give an empty string "" if you aren't using interstitials), and the app ID of the game that will be showing the ads. Both these strings can be found from the Google console for the ads.

Syntax

```
GoogleMobileAds_Init(interstitialID, appID);
```

Argument	Description	Data Type
interstitialID	The interstitial ID	String
appID	The App ID	String

Returns

N/A

Example

```
ads_app_id = "ca-app-pub-4334965884269841~8824063236";
interstitial_id = "ca-app-pub-3940256099942544/8691691433";
GoogleMobileAds_Init(interstitial_id, ads_app_id);
GoogleMobileAds_LoadInterstitial();
```

GoogleMobileAds_UseTestAds

Description

This function tells the app to use test ads instead of “live” ads, essential for testing your ads work without generating potentially fraudulent click-through. The first argument is either *true* or *false* to enable or disable test ads, and the second argument is the Device ID as returned by the debug output when testing.

Syntax

```
GoogleMobileAds_UseTestAds(use_test_ads, deviceID);
```

Argument	Description	Data Type
use_test_ads	Whether to enable/disable test ads	Boolean
deviceID	The unique device ID	String

Returns

N/A

Example

```
GoogleMobileAds_UseTestAds(true, "D5BA1B623CD21BB5A03CA60623C96F74");
```

GoogleMobileAds_AddBanner

Description

This function will add a banner advertisement to your game. You supply the unique banner ID for the ad block to use, as well as the "type" of ad format to use, which should be one of the following constants:

Constant Name	Size in dp (WxH)	Description	Availability
`GoogleMobileAds_Banner`	320x50	Standard Banner	Phones and Tablets
`GoogleMobileAds_Skyscraper`	320x100	Large Banner	Phones and Tablets
`GoogleMobileAds_MRect`	300x250	IAB Medium Rectangle	Phones and Tablets
`GoogleMobileAds_Full_Banner`	468x60	IAB Full-Size Banner	Tablets
`GoogleMobileAds_Leaderboard`	728x90	IAB Leaderboard	Tablets
`GoogleMobileAds_Smart_Banner`	Screen width x *32|50|90*	Smart Banner	Phones and Tablets

This function will also trigger a **Social Asynchronous Event** when the add is loaded. The `async_load` DS map will have the following key/value pairs:

- "**id**" – This will be the GML constant GoogleMobileAds_ASyncEvent
- "**type**" – For an event triggered by a banner loading the value will be the string "banner_load"
- "**loaded**" – This will be either 1, for loaded, or 0, for not loaded. If it's 0 then it may be that there is an error, or no connection, or there is no ad to be served.
- "**width**" – The width of the banner that has been loaded (only valid if the "loaded" key equals 1).
- "**height**" – The height of the banner that has been loaded (only valid if the "loaded" key equals 1).

Cont…/

Cont…/

Syntax

```
GoogleMobileAds_AddBanner(bannerAdId, sizeType);
```

Argument	Description	Data Type
bannerAdId	The ID of the banner ad block	String
sizeType	The type of ad to use (see the list of constants given above)	Constant

Returns

N/A

Example

```
banner_id = "ca-app-pub-3940256099942544/6300978111";
GoogleMobileAds_AddBanner(banner_id, GoogleMobileAds_Banner);
```

GoogleMobileAds_AddBannerAt

Description

This function will add a banner the same as the standard function, only this time, apart from the banner ID and type, you also supply a position for it somewhere within the display. Note that negative positions for the ad are possible but may render it invisible to the user.

The available type constants are:

Constant Name	Size in dp (WxH)	Description	Availability
`GoogleMobileAds_Banner`	320x50	Standard Banner	Phones and Tablets
`GoogleMobileAds_Skyscraper`	320x100	Large Banner	Phones and Tablets
`GoogleMobileAds_MRect`	300x250	IAB Medium Rectangle	Phones and Tablets
`GoogleMobileAds_Full_Banner`	468x60	IAB Full-Size Banner	Tablets
`GoogleMobileAds_Leaderboard`	728x90	IAB Leaderboard	Tablets
`GoogleMobileAds_Smart_Banner`	Screen width x *32|50|90*	Smart Banner	Phones and Tablets

This function will also trigger a **Social Asynchronous Event** when the add is loaded. The `async_load` DS map will have the following key/value pairs:

- "**id**" – This will be the GML constant GoogleMobileAds_ASyncEvent
- "**type**" – For an event triggered by a banner loading the value will be the string "banner_load"
- "**loaded**" – This will be either 1, for loaded, or 0, for not loaded. If it's 0 then it may be that there is an error, or no connection, or there is no ad to be served.
- "**width**" – The width of the banner that has been loaded (only valid if the "loaded" key equals 1).
- "**height**" – The height of the banner that has been loaded (only valid if the "loaded" key equals 1).

Cont…/

Cont…/

Syntax

```
GoogleMobileAds_AddBannerAt(bannerAdId, sizeType, x, y);
```

Argument	Description	Data Type
bannerAdId	The ID of the banner ad block	String
sizeType	The type of ad to use (see the list of constants given above)	Constant
x	The X position in the display	Real
y	The Y position in the display	Real

Returns

N/A

Example

```
banner_id = "ca-app-pub-3940256099942544/6300978111";
GoogleMobileAds_AddBannerAt(banner_id, GoogleMobileAds_Banner, 0, 0);
```

GoogleMobileAds_HideBanner

Description

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 no longer receive input. You can show the banner again at any time using the `GoogleMobileAds_ShowBanner()` function.

Syntax

```
GoogleMobileAds_HideBanner();
```

Returns

N/A

Example

```
if room != rm_Menu
    {
    GoogleMobileAds_HideBanner();
    }
```

GoogleMobileAds_ShowBanner

Description

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

Syntax

```
GoogleMobileAds_ShowBanner();
```

Returns

N/A

Example

```
if room == rm_Menu
    {
    GoogleMobileAds_ShowBanner();
    }
```

GoogleMobileAds_RemoveBanner

Description

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 `GoogleMobileAds_AddBanner()` function first to add a new banner to the display.

Syntax

```
GoogleMobileAds_RemoveBanner();
```

Returns

N/A

Example

```
if IAP_NoAds == true
    {
    GoogleMobileAds_RemoveBanner();
    }
```

GoogleMobileAds_MoveBanner

Description

This function can be used to move a banner that has been previously added. You supply the new X and Y position for the banner and it will be moved there, where the top left corner will be placed at the position given.

Syntax

```
GoogleMobileAds_MoveBanner(display_x, display_y);
```

Argument	Description	Data Type
display_x	The X position in the display to show the ad block at	Real
display_y	The Y position in the display to show the ad block at	Real

Returns

N/A

Example

```
var _x = display_get_width / 2;
var _w = GoogleMobileAds_BannerGetWidth() / 2;
if room == rm_Game
    {
    var _h = GoogleMobileAds_BannerGetHeight();
    GoogleMobileAds_MoveBanner(_x - _w, display_get_height() - _h);
    }
else
    {
    GoogleMobileAds_MoveBanner(_x - _w, 0);
    }
```

GoogleMobileAds_BannerGetWidth

Description

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

Syntax

```
GoogleMobileAds_BannerGetWidth();
```

Returns

Real

Example

```
var _x = display_get_width / 2;
var _w = GoogleMobileAds_BannerGetWidth() / 2;
if room == rm_Game
    {
    var _h = GoogleMobileAds_BannerGetHeight();
    GoogleMobileAds_MoveBanner(_x - _w, display_get_height() – _h);
    }
else
    {
    GoogleMobileAds_MoveBanner(_x - _w, 0);
    }
```

GoogleMobileAds_BannerGetHeight

Description

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

Syntax

```
GoogleMobileAds_BannerGetHeight();
```

Returns

Real

Example

```
var _x = display_get_width / 2;
var _w = GoogleMobileAds_BannerGetWidth() / 2;
if room == rm_Game
    {
    var _h = GoogleMobileAds_BannerGetHeight();
    GoogleMobileAds_MoveBanner(_x - _w, display_get_height() – _h);
    }
else
    {
    GoogleMobileAds_MoveBanner(_x - _w, 0);
    }
```

GoogleMobileAds_LoadInterstitial

Description

This function should be called when you want to load an interstitial ad. Caling 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 showing. 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 `GoogleMobileAds_InterstitialStatus()`.

This function will trigger a **Social Asynchronous Event** on load, which will have the following key/value pairs stored in the `async_load` DS map:

- "**id**" - The ID of the event being fired, in all cases for any AdMob extension callback it will be the GML constant `GoogleMobileAds_ASyncEvent`.
- "**type**" - This is the event type, and for interstitials and it will be either "interstitial_loaded" or "interstitial_closed".
- "**loaded**" - This will be either 1, for loaded, or 0, for not loaded. If it's 0 then it may be that there is an error, or no connection, or there is no ad to be served.

Syntax

```
GoogleMobileAds_LoadInterstitial();
```

Returns

N/A

Example

```
ads_app_id = "ca-app-pub-4334965884269841~8824063236";
interstitial_id = "ca-app-pub-3940256099942544/8691691433";
GoogleMobileAds_Init(interstitial_id, ads_app_id);
GoogleMobileAds_LoadInterstitial();
```

GoogleMobileAds_InterstitialStatus

Description

This function will return a status string which you can use to display the status of your interstitial ads. The possible return values are:

- "**Not Ready**" - No interstitial has been loaded
- "**Ready**" - The interstitial has been loaded and is ready to show

Syntax

```
GoogleMobileAds_InterstitialStatus();
```

Returns

String

Example

```
if GoogleMobileAds_InterstitialStatus() == "Ready"
    {
    GoogleMobileAds_ShowInterstitial();
    }
```

GoogleMobileAds_ShowInterstitial

Description

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

This function will trigger a **Social Asynchronous Event** when the add is closed, which will have the following key/value pairs stored in the `async_load` DS map:

- "**id**" - The ID of the event being fired, in all cases for any AdMob extension callback it will be the GML constant `GoogleMobileAds_ASyncEvent`.
- "**type**" - This is the event type, and for interstitials and it will be either "interstitial_loaded" or "interstitial_closed".

Syntax

```
GoogleMobileAds_ShowInterstitial();
```

Returns

N/A

Example

```
if GoogleMobileAds_InterstitialStatus() == “Ready”
    {
    GoogleMobileAds_ShowInterstitial();
    }
```

GoogleMobileAds_LoadRewardedVideo

Description

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 a rewarded video ad, nothing will be shown. Note that you can check whether a rewarded video is loaded or not using the function `GoogleMobileAds_RewardedVideoStatus()`.

This function will trigger a **Social Asynchronous Event** on load, which will have the following key/value pairs stored in the `async_load` DS map:

- "**id**" - The ID of the event being fired, in all cases for any AdMob extension callback it will be the GML constant `GoogleMobileAds_ASyncEvent`.

- "**type**" - This is the event type, and for rewarded videos and it will be either "rewardedvideo_adloaded" or " rewardedvideo_loadfailed".

- "**errorcode**" – If the event "type" is "rewardedvideo_loadfailed" then the DS map will also have this key, which will contain a string with the error code for the issue (see here for more information on error codes).

Syntax

```
GoogleMobileAds_LoadRewardedVideo(reward_id);
```

Argument	Description	Data Type
Reward_id	The unique ID of the reward video ad block	String

Returns

N/A

Example

```
ads_app_id = "ca-app-pub-4334965884269841~8824063236";
rewarded_id = "ca-app-pub-3940256099942544/5224354917";
GoogleMobileAds_Init(interstitial_id, ads_app_id);
GoogleMobileAds_LoadRewardedVideo(rewarded_id);
```

GoogleMobileAds_RewardedVideoStatus

Description

This function will return a status string which you can use to check the status of your rewarded video ads. The possible return values are:

- "**Not Ready**" - No rewarded video has been loaded
- "**Ready**" - The rewarded video has been loaded and is ready to show

Syntax

```
GoogleMobileAds_RewardedVideoStatus();
```

Returns

String

Example

```
if GoogleMobileAds_RewardedVideoStatus() == "Ready"
    {
    GoogleMobileAds_ShowRewardedVideo ();
    }
```

GoogleMobileAds_ShowRewardedVideo

Description

This function will show the rewarded video ad, if one is available and loaded. You can check whether an ad is available using the function `GoogleMobileAds_RewardedVideoStatus()`. Note that while a rewarded video ad is being shown, your app will be put into the background and be effectively “paused”.

Calling this function will show the rewarded video which will then trigger a sequence of **Social Asynchronous** Events, which will have the following key/value pairs stored in the `async_load` DS map:

- "**id**" - The ID of the event being fired, in all cases for any AdMob extension callback it will be the GML constant `GoogleMobileAds_ASyncEvent`.

- "**type**" - This is the event type, and for rewarded videos it will be either:

    - "*rewardedvideo_adopened*" - The ad has been opened (the `async_load` map has no further entries)

    - "*rewardedvideo_videostarted*" - The ad video has started playing (the `async_load` map has no further entries)

    - "*rewardedvideo_watched*" - The ad has been watched and the reward obtained. In this case the `async_load` map will contain two further keys:

        - "*currency*" - The name of the currency being awarded (a string). This can be stored and shown to the user, or it can be used to compare against a variable to make sure that the correct currency is being added (if you have multiple rewarded currency types)

        - "*amount*" - The amount of currency being awarded (a real number).

      Note that this event will only be triggered if the video has been watched to the end.

    - "*rewardedvideo_adclosed*" - The reward video has been closed. Note that this will trigger whenever the ad overlay is closed, **regardless of whether the video has been watched to the end**.

Cont../

Cont../

It is important to note that these async events will **not be triggered while the rewarded video is being shown**, as the app is put into the background. Instead they will be triggered consecutively when the app comes back to the main thread, so you can parse them and see what happened after calling the show function.

Syntax

```
GoogleMobileAds_ShowRewardedVideo();
```

Returns

N/A

Example

```
if GoogleMobileAds_RewardedVideoStatus() == “Ready”
    {
    GoogleMobileAds_ShowRewardedVideo ();
    }
```

GoogleMobileAds_ConsentUpdate

Description

This function can be used to update the Consent SDK cached data. You supply the publisher ID (as found on the AdMob dashboard), as well as a URL to the privacy policy that you want users to be linked to. You can then set flags to show buttons for opting into personalised ads, non-personalised ads, and a button to show the player an upsell option. Note that this function will check any cached data first, and only show the consent form if the “status” map entry is “UNKNOWN”.

The functions shown above will trigger a Social Asynchronous Event, which will populate the built-in async_load DS map with the following key/value pairs:

- "**type**" - This is used to identify the type of event being called, and for the consent SDK it will be "consent_status".

- "**id**" - This will be the built-in constant GoogleMobileAds_ASyncEvent, and is used to identify the event kind that has been triggered.

- "**status**" - This represents the user's choice when it comes to personalised ads. If "non_personalized" has been selected, the user should not be shown any personalised ad content. Note that this key will not exist if the "error" key exists. The possible return values are:

    - "UNKNOWN"
    - "NON_PRESONALIZED"
    - "PERSONALIZED"

    Note that if the status is "UNKNOWN", then the SDK will attempt to show the consent form to the user. Also note that if the user selects non-personalised ads, then they will be shown an additional forum to which they must agree.

- "**ad_free**" - This will either be 0 or 1. If set to 1, the user prefers an ad-free version of the app, in which case you can perform a url_open on the store page for the paid version of the app, or show some other kind of up-sell interface within the game.

- "**error**" - This reports any errors encountered while updating/querying the players consent, or while showing the consent form.

You can then parse this map data in the Async event and react accordingly. Note that if you choose to show a button to attract users to a paid version of the game, consent will not have been given to show ads. So, you will still have to show the consent form again if the user keeps using the app and you want to use ads.

Cont…/

Cont../

IMPORTANT! This function does NOT update the actual cached values, and you should always call GoogleMobileAds_ConsentSetAllowPersonalizedAds() in the async event with the result of the user input (unless "ad_free" is true, in which case you would be directing the user to an upsell screen and will still have to get their consent for showing ads).

Syntax

```
GoogleMobileAds_ConsentUpdate(publisherIds, privacyPolicyURL,
personalisedAds, noPersonalisedAds, adFree);
```

Argument	Description	Data Type
publisherIds	The publisher ID that you got from the Google AdMob dashboard	String
privacyPolicyURL	A URL that links to the Privacy Policy	String
personalisedAds	When set to *true*, the user will be presented with the option to see personalised ads.	Boolean
noPersonalisedAds	When set to *true*, the user will be presented with the option to see non-personalised ads.	Boolean
adFree	When set to *true*, the user will be presented with the option to see an ads-free version of your game.	Boolean

Returns

N/A

Example

```
ads_app_id = "ca-app-pub-4337965866269847~8824553239";
ads_consent_id = "pub-1564346280128643";
ads_device_id = "57BFBCB8E9A1AAA0CDEA08E925D11960";
ads_privacy_url = "https://www.yoyogames.com/legal/privacy";
GoogleMobileAds_Init("", ads_app_id);
GoogleMobileAds_ConsentDebugAddDevice(ads_device_id);
GoogleMobileAds_ConsentDebugSetDeviceInEEA(true);
GoogleMobileAds_ConsentUpdate(ads_consent_id, ads_privacy_url, true, true,
true);
```

GoogleMobileAds_ConsentFormShow

Description

This function can be used to show the Consent Form. You supply the publisher ID (as found on the AdMob dashboard), as well as a URL to the privacy policy that you want users to be linked to. You can then set flags to show buttons for opting into personalised ads, non-personalised ads, and a button to show the player an upsell option. Note that unlike the Update function, this will NOT check the cached values.

The functions shown above will trigger a Social Asynchronous Event, which will populate the built-in async_load DS map with the following key/value pairs:

- "**type**" - This is used to identify the type of event being called, and for the consent SDK it will be "consent_status".

- "**id**" - This will be the built-in constant GoogleMobileAds_ASyncEvent, and is used to identify the event kind that has been triggered.

- "**status**" - This represents the user's choice when it comes to personalised ads. If "non_personalized" has been selected, the user should not be shown any personalised ad content. Note that this key will not exist if the "error" key exists. The possible return values are:

    - "UNKNOWN"
    - "NON_PRESONALIZED"
    - "PERSONALIZED"

    Note that if the status is "UNKNOWN", then the SDK will attempt to show the consent form to the user. Also note that if the user selects non-personalised ads, then they will be shown an additional forum to which they must agree.

- "**ad_free**" - This will either be 0 or 1. If set to 1, the user prefers an ad-free version of the app, in which case you can perform a url_open on the store page for the paid version of the app, or show some other kind of up-sell interface within the game.

- "**error**" - This reports any errors encountered while updating/querying the players consent, or while showing the consent form.

You can then parse this map data in the Async event and react accordingly. Note that if you choose to show a button to attract users to a paid version of the game, consent will not have been given to show ads. So, you will still have to show the consent form again if the user keeps using the app and you want to use ads.

Cont…/

Cont../

IMPORTANT! This function does NOT update the actual cached values, and you should always call GoogleMobileAds_ConsentSetAllowPersonalizedAds() in the async event with the result of the user input (unless "ad_free" is true, in which case you would be directing the user to an upsell screen and will still have to get their consent for showing ads).

Syntax

```
GoogleMobileAds_ConsentFormShow(privacyPolicyURL, personalisedAds, noPersonalisedAds, adFree);
```

Argument	Description	Data Type
privacyPolicyURL	A URL that links to the Privacy Policy	String
personalisedAds	When set to *true*, the user will be presented with the option to see personalised ads.	Boolean
noPersonalisedAds	When set to *true*, the user will be presented with the option to see non-personalised ads.	Boolean
adFree	When set to *true*, the user will be presented with the option to see an ads-free version of your game.	Boolean

Returns

N/A

Example

```
ads_app_id = "ca-app-pub-4337965866269847~8824553239";
ads_device_id = "57BFBCB8E9A1AAA0CDEA08E925D11960";
ads_privacy_url = "https://www.yoyogames.com/legal/privacy";
GoogleMobileAds_Init("", ads_app_id);
GoogleMobileAds_ConsentDebugAddDevice(ads_device_id);
GoogleMobileAds_ConsentDebugSetDeviceInEEA(true);
GoogleMobileAds_ConsentFormShow(ads_privacy_url, true, true, true);
```

GoogleMobileAds_ConsentSetUserUnderAge

Description

This function can be used to set the cached user underage flag. If this is set to true, then the user is classed as under age and non-personalised ads will only be shown, regardless of any other setting, and the Consent form will not be shown when calling the Update function. Also note that it is up to you, the developer, to check the age of the user in some way.

Syntax

```
GoogleMobileAds_ConsentSetUserUnderAge(isUnderAge);
```

Argument	Description	Data Type
isUnderAge	Set to true to flag the user as under age, or false otherwise	Boolean

Returns

N/A

Example

```
if global.Age < 13
       {
       GoogleMobileAds_ConsentSetUserUnderAge(true);
       }
else GoogleMobileAds_ConsentSetUserUnderAge(false);
```

GoogleMobileAds_ConsentIsUserUnderAge

Description

This function can be used to check the cached user underage flag. If this returns true, then the user is classed as under age and non-personalised ads will only be shown, regardless of any other setting, and that the Consent form will not be shown when calling the Update function.

Syntax

```
GoogleMobileAds_ConsentIsUserUnderAge();
```

Returns

Boolean

Example

```
global.age_flag = GoogleMobileAds_ConsentIsUserUnderAge();
```

GoogleMobileAds_ConsentUserInEEA

Description

This function can be used to check the cached user location flag. If this is set to true then the user will be considered to be within the EEA and the consent form will be required, while if this is set to false, then the consent SDK can be bypassed.

Syntax

```
GoogleMobileAds_ConsentUserInEEA();
```

Returns

Boolean

Example

```
if global.EEA_Check = GoogleMobileAds_ConsentUserInEEA();
```

GoogleMobileAds_ConsentGetAllowPersonalizedAds

Description

This function can be used to check whether the Consent SDK cached setting for personalised ads is enabled or not. The function will return true if the user has enabled personalised ads, or false otherwise.

Syntax

```
GoogleMobileAds_ConsentGetAllowPersonalAds();
```

Returns

Boolean

Example

```
global.ads_personalised = GoogleMobileAds_GetAllowPersonalizedAds();
```

GoogleMobileAds_SetAllowPersonalizedAds

Description

This function can be used to set the cached personalised ads flag, which in turn will be used by Google to decide what ads to serve. The function takes a single argument which when set to true will enable personalised ads, and when set to false will disable them.

Syntax

```
GoogleMobileAds_SetAllowPersonalizedAds(allowPersonalized);
```

Argument	Description	Data Type
allowPersonalized	Set to true to permit personalised ads, or false not to.	Boolean

Returns

N/A

Example

```
if async_load[? "id"] == GoogleMobileAds_ASyncEvent
{
if async_load[? "type"] == "consent_status"
    {
    if !ds_map_exists(async_load, "error")
        {
        if async_load[? "status"] == "PERSONALIZED"
            {
            GoogleMobileAds_ConsentSetAllowPersonalizedAds(true);
            }
        else if async_load[? "status"] == "NON_PERSONALIZED"
            {
            GoogleMobileAds_ConsentSetAllowPersonalizedAds(true);
            }
        }
    }
}
```

GoogleMobileAds_ConsentDebugSetDeviceInEEA

Description

This function can be used to set the cached device location flag. If this is set to true then the device will be considered to be within the EEA and the consent form will be required, while if this is set to false, then the consent SDK will be bypassed. Note that this is a test/debug function only, and should only be used along with a test device to ensure that you games behaviour is correct regardless of whether the user is in the EEA or not.

Syntax

```
GoogleMobileAds_ConsentDebugSetDeviceInEEA(isInEEA);
```

Argument	Description	Data Type
isInEEA	Set to true to flag the device as being in the EEA, or false otherwise	Boolean

Returns

N/A

Example

```
if global.debug
{
GoogleMobileAds_ConsentDebugSetDeviceInEEA(true);
GoogleMobileAds_ConsentDebugAddDevice(ads_device_id);
}
```

GoogleMobileAds_ConsentDebugAddDevice

Description

This function can be used to add a test device to the Consent SDK. You supply the device ID string (which is shown in the Output Window when you runt he game with Ads enabled), and the function will add the connected device as a test device to the consent SDK, permitting you to set the location flag for testing and debugging (see the function GoogleMobileAds_ConsentDebugSetDeviceInEEA()).

Syntax

```
GoogleMobileAds_ConsentDebugAddDevice(id);
```

Argument	Description	Data Type
Id	The device id, as shown in the output window.	String

Returns

N/A

Example

```
if global.debug
{
GoogleMobileAds_ConsentDebugSetDeviceInEEA (true);
GoogleMobileAds_ConsentDebugAddDevice(ads_device_id);
}
```