GOOGLE PLAY BILLING (IAP) MANUAL

Contents

Introduction

This manual is designed for you to use as a reference to the different Google Play IAP (In App Purchase) functions for Android, 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 article:

- Android: Google Play Billing (IAPs)

We also recommend that before doing *anything* with this extension, you take a moment to look over the official Google Play Billing API documentation, as it will familiarise you with many of the terms and concepts required to use the extension correctly, and many of the functions in the extension are practically 1:1 mappings of the methods described there:

- Google Developer Docs: Billing Overview

NOTE: *The old runner "iap…()" functions no longer work as of GMS 2.2.4 and your code must be updated to use this extension.*

The Asynchronous IAP Event

When using the Google Play Billing extension in your projects, you will be calling different functions that will trigger “callbacks” from the Google Billing API. What this means is that certain functions will be run but won’t return 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 Billing API responding to something you’ve done. This callback is dealt with in the **Asynchronous IAP 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 one key in common which used to identify the type of callback that is being received:

- **“id”** – This is the event ID key and it will hold a CONSTANT with the ID of the event that has been triggered. For example, if it’s an event to tell you a product has been purchased, then the constant will be `gpb_receipt`. See the different functions for details about the constant returned for each.

The rest of the key/value pairs in the map will depend on the function that triggered the Async Event and the ID of the 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 contains a reference guide to all the functions used by the Google Billing Extension, along with any constants that they may use or return and examples of code that use them. Some of the examples are **Extended Examples** that also show code from callbacks in the Asynchronous IAP Event.

It is worth noting that in some cases the function description will mention the use of a private server to verify purchases. This is not required (although is still recommended) for regular product IAPs, however Google *highly recommend it for subscription IAPs*. Setting up subscriptions and the server to deal with them is outside of the scope of this documentation and, instead, we refer you to the Google docs here:

- Google Developer Docs: Add Subscription-Specific Features
- Google Developer Docs: Verify A Purchase On A Server

NOTE: *At various times the Google Billing documentation (and hence, this manual) talk about* ***"Product IDs"*** *and* ***"SKUs"****. These are two terms for the same thing and refer to the unique name you gave to your different IAP products on your Google Play Console.*

The general workflow for using this extension is as follows:

- At the start of the game, attempt to connect to the Play Store
- If connection fails, disable the possibility for purchases in your game UI (and set an alarm or something to test again at intervals)
- If connection succeeds, add the different products to the internal products list
- After adding the products but before permitting purchases, query existing purchases and if there are any consumable purchases or unacknowledged non-consumable/subscription purchases outstanding, then consume or acknowledge them.
- Permit the game to run as normal and let the user purchase/consume products as required, checking for connection to the store at all times to prevent erroneous purchases

Note that all products (except subscriptions) are considered **consumable**, so should you wish to have a non-consumable product – for example, a "no ads" product – then you simply do not call the `GPBilling_ConsumeProduct()` function on that item. However, non-consumable products still need to be **acknowledged** within 2 days of purchase, otherwise the purchase will be refunded. This would be done using the function `GPBilling_AcknowledgePurchase()`.

GPBilling_Init

Description

This function will initialise the Google Play Billing API and **is called automatically by the extension**. As such, you should *not* be writing it in your game code, as it is not required.

Syntax

```
GPBilling_Init();
```

Returns

N/A

Example

N/A

GPBilling_ConnectToStore

Description

This function will attempt to connect the Google Play Billing API with the Play store. When you call this function, it will return one the constants listed below to inform you of the status of the connection attempt. This function ***must be called before calling any other IAP functions*** and the return status should be **`gpb_no_error`**. This does not, however, mean that the Store is available, only that the connection *attempt* has been successful. Before you can successfully define, query or purchase any products, you must ensure that the connection is valid.

To check the availability of the Play Store, the function will also trigger one of two callbacks in the **Asynchronous IAP Event** (when the initial returned status is **`gpb_no_error`**). In this event, the `async_load` DS map will have the following constants returned for the "**id**" key:

Constant	Actual Value	Description
gpb_store_connect	2005	The API has connected to the Google Play store.
gpb_store_connect_failed	2006	The API has failed to connect to the Google Play store.

Syntax

```
GPBilling_ConnectToStore();
```

Returns

Constant

Constant	Actual Value	Description
gpb_error_unknown	-1	There was an unknown error preventing the Billing API from creating a connection request.
gpb_no_error	0	The Billing API has created a connection request correctly.

Cont…/

Extended Example

In this extended example, we first send an API request to connect to the store in some event. This would normally be done in the Create Event of a dedicated controller object that is one of the first things created in your game:

```
global.IAP_Enabled = false;
var _init = GPBilling_ConnectToStore();
if _init == gpb_error_unknown
    {
    show_debug_message(“ERROR - Billing API Has Not Connected!”);
    alarm[0] = room_speed * 10;
    }
```

Note that if the connection request has failed, then we can – for example - call an alarm, where we can call this same code again to test for store connection periodically.

Assuming the API has correctly requested a store connection, it will trigger an Asynchronous IAP Event where you can check to see if the API has successfully connected to the Google Play store or not:

```
var _eventId = async_load[? "id"];
switch (_eventId)
    {
    case gpb_store_connect:
        // Store has connected so here you would generally add the products
        global.IAP_Enabled = true;
        GPBilling_AddProduct(global.IAP_PurchaseID[0]);
        GPBilling_AddSubscription(global.IAP_PurchaseID[1]);
        // Etc…
        break;
    case gpb_store_connect_failed:
        // Store has failed to connect, so try again periodically
        alarm[0] = room_speed * 10;
        break;
    }
```

In the above example, if the store connection fails, you’ll see we call an alarm event, setting it to count down 10 seconds. In this event we can then try to initialise the store once more using the same code that we have in the Create Event, shown above.

GPBilling_IsStoreConnected

Description

This function will check to see if the Google Play Billing API currently has a connection to the Google Play store. The function will return `true` if it is and `false` if it is not. Note that if there is *no* connection, ***you should not permit any further Google Play Billing API function calls*** and you could also disable or hide any purchase options for the user in your game UI until connection has been re-established.

In general, you should always call this function and check connectivity before doing any interactions with the Billing API.

Syntax

```
GPBilling_IsStoreConnected();
```

Returns

Boolean

Example

```
if mouse_check_button_pressed(mb_left) &&
    {
    if instance_position(mouse_x, mouse_y, id)
        {
        if GPBilling_IsStoreConnected() && global.IAP_Enabled == true
            {
            GPBilling_PurchaseProduct(global.IAP_PurchaseID[0]);
            }
        else
            {
            global.IAP_Enabled == false;
            alarm[0] = room_speed * 10;
            }
        }
    }
```

GPBilling_AddProduct

Description

This function can be used to add a **consumable** product to the internal product list for purchase. You supply the Product ID (as a string, the same as the product ID on the Google Play Console for the game), and the function will return one of the constants listed below.

For subscription products, you should be using the function `GPBilling_AddSubscription()`.

Note, there is no difference between a consumable and a non-consumable product as far as the API is concerned. So, for non-consumable IAPs – like a “no ads” IAP, for example – you simply don’t call the `GPBilling_ConsumeProduct()` function on it. However, non-consumables should still be acknowledged using the `GPBilling_AcknowledgePurchase()` function when purchased.

Syntax

```
GPBilling_AddProduct(product_id);
```

Argument	Description	Data Type
product_id	The product ID (SKU) of the IAP product being added.	String

Returns

Constant

Constant	Actual Value	Description
gpb_error_unknown	-1	This error indicates that the product being added is not unique (ie: the product ID has already been added to the internal product list).
gpb_no_error	0	The product was successfully added to the internal product list.

Cont…/

Example

The following code is being called from the Asynchronous IAP Event when it has been triggered by the function `GPBilling_ConnectToStore()`.

```
var _eventId = async_load[? "id"];
switch (_eventId)
    {
    case gpb_store_connect:
        GPBilling_AddProduct(global.IAP_PurchaseID[0]);
        GPBilling_AddSubscription(global.IAP_PurchaseID[1]);
        GPBilling_QueryProducts();
        break;
    }
```

GPBilling_AddSubscription

Description

This function can be used to add a **subscription** product to the internal product list for purchase. You supply the Product ID (as a string, the same as the product ID on the Google Play console for the game), and the function will return one of the constants listed below.

For consumable products, you should be using the function GPBilling_AddProduct().

Syntax

```
GPBilling_AddSubscription(product_id);
```

Argument	Description	Data Type
product_id	The product ID (SKU) of the IAP product being added.	String

Returns

Constant

Constant	Actual Value	Description
gpb_error_unknown	-1	This error indicates that the product being added is not unique (i.e.: the product ID has already been added to the internal product list).
gpb_no_error	0	The product was successfully added to the internal product list.

Cont…/

Example

The following code is being called from the Asynchronous IAP Event when it has been triggered by the function GPBilling_ConnectToStore().

```
var _eventId = async_load[? “id”];
switch (_eventId)
    {
    case gpb_store_connect:
        GPBilling_AddProduct(global.IAP_PurchaseID[0]);
        GPBilling_AddSubscription(global.IAP_PurchaseID[1]);
        GPBilling_QueryProducts();
        break;
    }
```

GPBilling_QueryProducts

Description

This function can be used to query the state of any **consumable** products (for subscriptions, please use the function `GPBilling_QuerySubscriptions()`). This function will generate an Asynchronous IAP Event where the `async_load` DS map "**id**" key holds the constant **`gpb_product_data_response`**, as well as the key "**json_response**". This key contains a JSON object string, which – when decoded using `json_decode()` – will contain DS map. This map will have the key "**success**" - which will be `true` if the query has been successfully processed, and `false` otherwise – as well as the key "**skuDetails**" (only if "success" is `true`). The "skuDetails" key will, in turn, hold a DS list ID where each entry into the list contains a DS map ID with the details for each of the activated IAP products.

The DS map for each individual product will contain the following keys:

- "**skuDetailsToken**" – This is a unique token created by Google for the details request.
- "**productId**" – The product ID (SKU, a string) as listed on the Google Play console for the game.
- "**type**" – The IAP type for the product. Will be one of the following constants:

Constant	Actual Value	Description
gpb_purchase_skutype_inapp	"inapp"	This constant indicates that the product is a consumable purchase.
gpb_purchase_skutype_subs	"subs"	This constant indicates that the product is a subscription purchase.

- "**price**" – Returns *formatted* price of the item (a string), including its currency sign. The price does not include tax.
- "**price_amount_micros**" – Returns the price in *micro-units* (an integer), where 1,000,000 micro-units equals one unit of the currency. For example, if the price is "€7.99", then the price in micros is "7990000". This value represents the localised and rounded price for a particular currency.

Cont…/

- "**price_currency_code**" – Returns the ISO 4217 currency code for the price and original price (a string). For example, if the price is specified in British pounds sterling, then the code returned would be "GBP".

- "**title**" – Returns the title of the product (a string) as defined in the Google Play console.

- "**description**" – Returns the product description (a string) as defined in the Google Play console.

NOTE: *You should NOT call this function at the same time as the equivalent subscription query, as this may cause the Google API to error. Instead, call one function, and then in the Asynchronous IAP Event callback, call the other function if you need to.*

Syntax

```
GPBilling_QueryProducts();
```

Returns

N/A

Extended Example

The following code is being called from the Asynchronous IAP Event when it has been triggered by the function `GPBilling_ConnectToStore()`.

```
var _eventId = async_load[? "id"];
switch (_eventId)
    {
    case gpb_store_connect:
        GPBilling_AddProduct(global.IAP_PurchaseID[0]);
        GPBilling_AddProduct(global.IAP_PurchaseID[1]);
        GPBilling_QueryProducts();
        break;
    }
```

Cont…/

The query products function will then trigger another call to the Asynchronous IAP Event, which can be parsed by adding another case to the switch, this time checking the async_load key “**id**” for the constant **gpb_product_data_response**. Something like this:

```
case gpb_product_data_response:
    var _json = async_load[? “response_json”];
    var _map = json_decode(_json);
    if _map[? “success”] == true
        {
        var _plist = _map[? “skuDetails”];
        for (var i = 0; i < ds_list_size(_plist); ++i;)
            {
            var _pmap = _plist[| i];
            var _num = 0;
            while(_pmap[? “productId”] != global.IAP_PurchaseID[_num])
                {
                ++_num;
                }
            global.IAP_ProductData[_num, 0] = _pmap[? “price”];
            global.IAP_ProductData[_num, 1] = _pmap[? “title”];
            global.IAP_ProductData[_num, 2] = _pmap[? “description”];
            }
        GPBilling_QuerySubscriptions();
        }
    ds_map_destroy(_map);
    break;
```

Note that after parsing the returned product data, we then call the equivalent query function for subscriptions, and then when that triggers another asynchronous callback we’d call the function GPBilling_QueryPurchases() to check for any purchases that haven’t been consumed.

GPBilling_QuerySubscriptions

Description

This function can be used to query the state of any **subscription** products (for consumables, please use the function GPBilling_QueryProducts()). This function will generate an Asynchronous IAP Event where the `async_load` DS map "**id**" key holds the constant **`gpb_subscription_data_response`**, as well as the key "**json_response**". This key contains a JSON object string, which – when decoded using `json_decode()` – will contain DS map. This map will have the key "**success**" - which will be `true` if the query has been successfully processed, and `false` otherwise – as well as the key "**skuDetails**" (only if "success" is `true`). The "skuDetails" key will, in turn, hold a DS list ID where each entry into the list contains a DS map ID with the details for each of the activated IAP products.

The DS map for each individual product will contain the following keys:

- "**skuDetailsToken**" – This is a unique token created by Google for the details request.
- "**productId**" – The subscription product ID (SKU, a string) as listed on the Google Play console for the game.
- "**type**" – The IAP type for the product. Will be one of the following constants:

Constant	Actual Value	Description
gpb_purchase_skutype_inapp	"inapp"	This constant indicates that the product is a consumable purchase.
gpb_purchase_skutype_subs	"subs"	This constant indicates that the product is a subscription purchase.

- "**price**" – Returns *formatted* price of the subscription (a string), including its currency sign. The price does not include tax.
- "**price_amount_micros**" – Returns the price in *micro-units* (an integer), where 1,000,000 micro-units equals one unit of the currency. For example, if the price is "€7.99", then the price in micros is "7990000". This value represents the localized, rounded price for a particular currency.

Cont…/

- "**price_currency_code**" – Returns the ISO 4217 currency code for the price and original price (a string). For example, if the price is specified in British pounds sterling, then the code returned would be "GBP".

- "**title**" – Returns the title of the subscription product (a string) as defined in the Google Play console.

- "**description**" – Returns the subscription product description (a string) as defined in the Google Play console.

NOTE: *You should NOT call this function at the same time as the equivalent products query, as this may cause the Google API to error. Instead, call one function, and then in the Asynchronous IAP Event callback, call the other function if you need to.*

Syntax

```
GPBilling_QuerySubscriptions();
```

Returns

N/A

Extended Example

The following code is being called from the Asynchronous IAP Event when it has been triggered by the function `GPBilling_ConnectToStore()`.

```
var _eventId = async_load[? "id"];
switch (_eventId)
    {
    case gpb_store_connect:
        GPBilling_AddSubscription(global.IAP_PurchaseID[0]);
        GPBilling_AddSubscription(global.IAP_PurchaseID[1]);
        GPBilling_QuerySubscriptions();
        break;
    }
```

Cont…/

The query subscriptions function will then trigger another call to the Asynchronous IAP Event, which can be parsed by adding another case to the switch, this time checking the async_load key "**id**" for the constant **gpb_subscription_data_response**, something like this:

```
case gpb_subscription_data_response:
    var _json = async_load[? "response_json"];
    var _map = json_decode(_json);
    if _map[? "success"] == true
        {
        var _plist = _map[? "skuDetails"];
        for (var i = 0; i < ds_list_size(_plist); ++i;)
            {
            var _pmap = _plist[| i];
            var _num = 0;
            while(_pmap[? "productId"] != global.IAP_PurchaseID[_num])
                {
                ++_num;
                }
            global.IAP_PurchaseData[_num, 0] = _pmap[? "price"];
            global.IAP_PurchaseData[_num, 1] = _pmap[? "title"];
            global.IAP_PurchaseData[_num, 2] = _pmap[? "description"];
            }
        GPBilling_QueryPurchases(gpb_purchase_skutype_inapp);
        }
    ds_map_destroy(_map);
    break;
```

Note that after parsing the returned subscription data, we then call the function GPBilling_QueryPurchases() to check for any purchases that haven't been consumed. If you haven't already queried consumable products, then you should probably do that first, then in the corresponding Asynchronous IAP Event callback, you'd check the purchase status.

GPBilling_QueryPurchases

Description

This function is used for querying the purchase state of the different products available for your game. This function should always be called before permitting any in-app purchases, preferable near the startup of the game itself. The function takes one of the following constants as the "type" argument:

Constant	Actual Value	Description
gpb_purchase_skutype_inapp	"inapp"	This constant indicates that you are querying the purchase state of consumable products.
gpb_purchase_skutype_subs	"subs"	This constant indicates that you are querying the purchase state of subscription products.

Unlike some of the other Billing functions, this one does not generate a callback event, but will instead immediately return a JSON string which can be decoded using the `json_decode()` function. The initial JSON top-level DS map will have the following keys:

- "**success**" – This will be either `true` or `false` depending on whether the purchase query succeeded or not.

If "success" is `false`, then there will be an additional key:

- "**responseCode**" – This is an integer value that corresponds to one of the Google Play Store response codes listed here. Note that if a purchase has been cancelled, you'll get "success: false" and "responseCode:1", for "USER_CANCELLED".

If "success" is `true`, then the additional key will be:

- "**purchases**" – This is a DS list ID, where each entry in the list corresponds to a DS map for an individual purchase.

When the "purchases" key exists, this can then be looped through (as shown in the extended example below) to get the individual DS maps with the product and purchase information. Each purchase map will contain the following keys:

- "**orderId**" - Returns a unique order identifier for the transaction (a string). This identifier corresponds to the Google payments order ID.

- "**packageName**" - Returns the application package from which the purchase originated (a string).

Cont…/

- "**productId**" - Returns the product ID (SKU, a string).
- "**purchaseTime**" - Returns the time the product was purchased (an integer). This is in milliseconds since the epoch (Jan 1, 1970).
- "**purchaseState**" - Returns the state of purchase (an integer). Possible values are:
  - 0 – Un-Specified State
  - 1 – Purchased
  - 2 – Pending
- "**purchaseToken**" - Returns a token that uniquely identifies a purchase for a given item and user pair (a string). This should be used for any server verification.
- "**autoRenewing**" - Indicates whether the subscription renews automatically (boolean, will always be false for non-subscription purchases).
- "**acknowledged**" - The acknowledgement state of the in-app product. Possible (integer) values are:
  - 0 - Yet to be acknowledged
  - 1 – Acknowledged

Purchases that have been made but not consumed will have a "purchaseState" of 1 for purchased, while purchases that are in progress but not yet resolved will have a state of 2 for pending.

Keep in mind that any NON-consumable purchases will also have the purchased state (1), as the Google Billing API makes no distinction between consumable and non-consumable and it's up to you to decide when and if a purchase is consumed. However, **all purchases must be *acknowledged* within 2 days of purchase, even if they are not being consumed**. This is done automatically when a consumable is used, however for non-consumables this must be done using the function `GPBilling_AcknowledgePurchase()`. If you do not acknowledge a purchase within 2 days, it will be refunded.

Syntax

```
GPBilling_QueryPurchases(type);
```

Argument	**Description**	**Data Type**
type	The type of product to be queried.	Constant (see the description above)

Cont…/

Returns

String (JSON)

Extended Example

For this example, we would first want to connect to the store, then add products and then query the product status, before checking the purchase state of each product. So, for example, we'd have a Create Event like this:

```
global.IAP_Enabled = false;
var _init = GPBilling_ConnectToStore();
if _init == gpb_error_unknown
    {
    show_debug_message(“ERROR - Billing API Has Not Connected!”);
    alarm[0] = room_speed * 10;
    }
```

Assuming the API has correctly requested a store connection, it will trigger an Asynchronous IAP Event where you can check to see if the API has successfully connected to the Google Play store or not, and then add each of the products that you want to be available to the user, and then query the product details:

```
var _eventId = async_load[? "id"];
switch (_eventId)
    {
    case gpb_store_connect:
        global.IAP_Enabled = true;
        GooglePlayBilling_AddProduct(global.IAP_PurchaseID[0]);
        GooglePlayBilling_AddProduct(global.IAP_PurchaseID[1]);
        GPBilling_QueryProducts();
        break;
    }
```

The query products function will then trigger another Asynchronous IAP event, and we can add another `case` to our `switch` statement where we can check the state of any purchases from those that we've added, and consume or acknowledge any purchased product as required:

Cont…/

```
        case gpb_product_data_response:
            if async_load[? "success"] == true
                {
                var _json = GPBilling_QueryPurchases();
                var _jsonmap = json_decode(_json);
                if _jsonmap[? "success"] == true
                    {
                    var _list = _jsonmap[? "purchases"];
                    var _sz = ds_list_size(_list);
                    for (var i = 0; i < _sz; ++i;)
                        {
                        var _map = _list[| i];
                        if _map[? "purchaseState"] == 1
                            {
                            var _pid = _map[? "productId"];
                            var _token = map[? "purchaseToken"];
                            var _add = false;
                            if _pid == global.IAP_PurchaseID[0]
                                {
                                GPBilling_ConsumeProduct(_token);
                                _add = true;
                                }
                            else if _pid == global.IAP_PurchaseID[1]
                                {
                                if _map[? "acknowledged"] == 0
                                    {
                                    GPBilling_AcknowledgePurchase(_token);
                                    _add = true;
                                    }
                                }
                            if _add
                                {
                                ds_list_add(global.CurrentTokens, _token);
                                ds_list_add(global.CurrentProduct, _pid);
                                }
                            }
                        }
                    }
                ds_map_destroy(_jsonmap);
                }
            break;
```

Note that we store the purchase tokens and product IDs of those products we consume or acknowledge in global ds lists. This is done so that we can track the purchases correctly when the consumed or acknowledged response comes back (see the functions `GPBilling_AcknowledgePurchase()` and `GPBilling_ConsumeProduct()` for more details).

GPBilling_PurchaseProduct

Description

This function will send a purchase request to the Billing API and attempt to purchase the product with the given ID. The ID value should be a string and is the product identifier name on the Google Play console, for example "buy_100_gold". The function will return one of the constants listed below to indicate the initial status of the purchase request, and then an Asynchronous IAP Event will be triggered with the callback.

The Async IAP Event callback will return the `async_load` DS map, which will contain an "**id**" key with the constant **`gpb_iap_receipt`** for a purchase request, as well as the key "**response_json**", which will contain a JSON formatted string with the purchase data. This JSON can then be decoded using the function `json_decode()` and parsed to retrieve the different elements of the purchase data.

The decoded JSON will be a DS map with two keys: "**success**" – which will be `true` or `false` depending on whether the purchase request was successful – and "**purchases**" (this will not exist if the "success" key returns `false`). The value held in the "purchases" key will be a DS list ID, where each list entry corresponds to an individual purchase DS map, so you should iterate through the list and parse the map data from each entry.

The DS map for each individual purchase will contain the following keys:

- "**orderId**" - Returns a unique order identifier for the transaction (a string). This identifier corresponds to the Google payments order ID.

- "**packageName**" - Returns the application package from which the purchase originated (a string).

- "**productId**" - Returns the product ID (SKU, a string).

- "**purchaseTime**" - Returns the time the product was purchased (an integer). This is in milliseconds since the epoch (Jan 1, 1970).

- "**purchaseState**" - Returns the state of purchase (an integer). Possible values are:
  - 0 – Un-Specified State
  - 1 – Purchased
  - 2 – Pending

- "**purchaseToken**" - Returns a token that uniquely identifies a purchase for a given item and user pair (a string). This should be used for any server verification.

Cont…/

- "**autoRenewing**" - Indicates whether the subscription renews automatically (boolean, will always be false for non-subscription purchases).

- "**acknowledged**" - The acknowledgement state of the in-app product. Possible values are:
  - 0 - Yet to be acknowledged
  - 1 – Acknowledged

Keep in mind that any NON-consumable purchases will also have the purchased state (1), as the Google Billing API makes no distinction between consumable and non-consumable and it's up to you to decide when and if a purchase is consumed. However, **all purchases must be *acknowledged* within 2 days of purchase, even if they are not being consumed**. This is done automatically when a consumable is used, however for non-consumables this must be done using the function `GPBilling_AcknowledgePurchase()`. If you do not acknowledge a purchase within 2 days, it will be refunded.

Syntax

```
GPBilling_PurchaseProduct(product_id);
```

Argument	Description	Data Type
product_id	The ID of the product as shown on the Google Play console.	String

Returns

Constant

Constant	Error Code	Description
gpb_error_not_initialised	1	The Billing API has not been initialised before calling this function.
gpb_error_no_skus	2	There are no SKUs in the product list *nor* the subscription list.
gpb_error_selected_sku_list_empty	3	You have tried to purchase a product when there is no product in the list (although there may be subscriptions in the list)
gpb_no_error	0	The Billing API has been initialised correctly.

Cont…/

Extended Example

The following code would be used in (for example, but not limited to) a mouse pressed event to purchase a product:

```
if GPBilling_StoreIsConnected()
    {
    var _chk = GPBilling_PurchaseProduct(global.IAP_PurchaseID[0]);
    if _chk != gpb_no_error
        {
        // Purchase unavailable, add failsafe code if required
        }
    }
```

You would then have something like the following code in the **IAP Asynchronous Event** to deal with the purchase callback (note that the following code shows a simple purchase verification scheme, however you should ideally verify the purchase with an http call to your server, supplying the returned token string, and then consume the purchase when you receive verification in the Asynchronous HTTP Event):

```
var _eventId = async_load[? "id"];
switch (eventId)
    {
    case gpb_iap_receipt:
        var _json = async_load[? "response_json"];
        var _map = json_decode(response_json);
        if _map[? “success”] == true
            {
            if ds_map_exists(_map, "purchases")
                {
                var _plist = ds_map_find_value(_map, "purchases");
                for (var i = 0; i < ds_list_size(purchases);  ++i;)
                    {
                    var _pmap = _plist[| i];
                    var _ptoken = _pmap[? "purchaseToken"];
                    var _sig = GPBilling_Purchase_GetSignature(_ptoken);
                    var _pjson = GPBilling_Purchase_GetOriginalJson(_ptoken);
                    if GPBilling_Purchase_VerifySignature(_pjson, _sig)
                        {
                        GPBilling_ConsumeProduct(_ptoken);
                        ds_list_add(global.CurrentTokens, _ptoken);
                        ds_list_add(global.CurrentProducts, _pmap[? "productId"]);
                        }
                    }
                }
            }
        ds_map_destroy(_map);
        break;
    }
```

Note that we store the purchase tokens and product IDs of those products we consume or acknowledge in global ds lists. This is done so that we can track the purchases correctly when the consumed or acknowledged response comes back (see the functions `GPBilling_AcknowledgePurchase()` and `GPBilling_ConsumeProduct()` for more details).

GPBilling_PurchaseSubscription

Description

This function will send a subscription purchase request to the Billing API and attempt to subscribe to the product with the given ID. The ID value should be a string and is the subscription identifier name on the Google Play console, for example "improved_version". The function will return one of the constants listed below to indicate the initial status of the subscription request, and then an Asynchronous IAP Event will be triggered with the callback.

The Async IAP Event callback will return the `async_load` DS map, which will contain an "**id**" key with the constant **`gpb_iap_receipt`** for a purchase request, as well as the key "**response_json**", which will contain a JSON formatted string with the purchase data. This JSON can then be decoded using the function `json_decode()` and parsed to retrieve the different elements of the purchase data.

The decoded JSON will be a DS map with two keys: "**success**" – which will be `true` or `false` depending on whether the purchase request was successful – and "**purchases**" (this will not exist if the "success" key returns `false`). The value held in the "purchases" key will be a DS list ID, where each list entry corresponds to an individual purchase DS map, so you should iterate through the list and parse the map data from each entry.

The DS map for each individual purchase will contain the following keys:

- "**orderId**" - Returns a unique order identifier for the transaction (a string). This identifier corresponds to the Google payments order ID.
- "**packageName**" - Returns the application package from which the purchase originated (a string).
- "**productId**" - Returns the subscription ID (SKU, a string).
- "**purchaseTime**" - Returns the time the subscription was purchased (an integer). This is in milliseconds since the epoch (Jan 1, 1970).
- "**purchaseState**" - Returns the state of purchase (an integer). Possible values are:
    - 0 – Un-Specified State
    - 1 – Purchased
    - 2 – Pending
- "**purchaseToken**" - Returns a token that uniquely identifies a subscription purchase for a given item and user pair (a string). This should be used for any server verification.

Cont…/

- "**autoRenewing**" - Indicates whether the subscription renews automatically (boolean, will always be false for non-subscription purchases).

- "**acknowledged**" - The acknowledgement state of the subscription. Possible values are:
  - 0 - Yet to be acknowledged
  - 1 – Acknowledged

Keep in mind that **all subscription purchases must be *acknowledged* within 2 days of purchase**, using the function `GPBilling_AcknowledgePurchase()`. If you do not acknowledge a purchase within 2 days, it will be refunded. Also note that after the initial purchase, the subscription will be renewed automatically by Google, and all you have to do is query the subscription state each time the game starts to unlock or block any additional content as required.

IMPORTANT! *Setting up and using subscriptions requires an external server to be able to communicate with your app and with the Google Play servers for verification and other purposes. This is outside of the scope of this documentation and instead we refer you to the following documents:*

- Google Developer Docs: Add Subscription-Specific Features
- Google Developer Docs: Verify A Purchase On A Server

Syntax

```
GPBilling_PurchaseSubscription(product_id);
```

Argument	Description	Data Type
product_id	The ID of the product as shown on the Google Play console.	String

Cont…/

Returns

Constant

Constant	Error Code	Description
gpb_error_not_initialised	1	The Billing API has not been initialised before calling this function.
gpb_error_no_skus	2	There are no SKUs in the product list or subscription list.
gpb_error_selected_sku_list_empty	3	You have tried to purchase a subscription when there is no subscription in the list (although there may be products in the list)
gpb_no_error	0	The Billing API has been initialised correctly.

Extended Example

The following code would be used in (for example, but not limited to) a mouse pressed event to purchase a product:

```
if GPBilling_StoreIsConnected()
    {
    var _chk = GPBilling_PurchaseSubscription(global.IAP_PurchaseID[0]);
    if _chk != gpb_no_error
        {
        // Purchase unavailable, add failsafe code if required
        }
    }
```

You would then have something like the following code in the **IAP Asynchronous Event** to deal with the purchase callback (note that the following code shows a simple purchase verification scheme, however you should ideally verify the purchase with an http call to your server, supplying the returned token string, and then consume the purchase only when you receive verification in the Asynchronous HTTP Event):

Cont…/

Contents

```
var _eventId = async_load[? "id"];
switch (eventId)
    {
    case gpb_iap_receipt:
        var _json = async_load[? "response_json"];
        var _map = json_decode(response_json);
        if _map[? “success”] == true
            {
            if ds_map_exists(_map, "purchases")
                {
                var _plist = ds_map_find_value(_map, "purchases");
                for (var i = 0; i < ds_list_size(purchases);  ++i;)
                    {
                    var _pmap = _plist[| i];
                    var _ptoken = _pmap[? "purchaseToken"];
                    var _sig = GPBilling_Purchase_GetSignature(_ptoken);
                    ds_list_add(global.CurrentTokens, _ptoken);
                    ds_list_addglobal.CurrentProduct, _pmap[? “productId”]);
                    // SERVER VERIFICATION CODE HERE
                    // Here you would send the token and signature to
                    // your servers for verification with the Google
                    // Play API and then return the result, which would
                    // then be received in an Asynchronous HTTP event.
                    // Once received, the subscription can then
                    // be acknowledged, and content/bonuses etc…
                    // can be unlocked in your game
                    }
                }
            }
        ds_map_destroy(_map);
        break;
    }
```

Note that we store the purchase tokens and product IDs of those products we consume or acknowledge in global ds lists. This is done so that we can track the purchases correctly when the consumed or acknowledged response comes back (see the functions `GPBilling_AcknowledgePurchase()` and `GPBilling_ConsumeProduct()` for more details).

GPBilling_AcknowledgePurchase

Description

This function will acknowledge a purchase or subscription. When you receive notification that a purchase has been made, it needs to be acknowledged with the Google servers within 2 days otherwise it is refunded. This is done automatically for consumable purchases when you call the function `GPBilling_ConsumeProduct()`, but for non-consumable and subscriptions, you must call this function to let Google know the purchase has been received correctly.

On calling the function initially, it will return one of the constants listed below to inform you of the request status, and if this is **`gpb_no_error`** then an Asynchronous IAP Event will be triggered where the `async_load` DS map will have the key "**id**" which will correspond to the extension constant **`gpb_acknowledge_purchase_response`**. Additionally, the map will have the key "**response_json**" which will be a JSON string that can be converted into a DS map using the `json_decode()` function.

The decoded JSON will be a DS map which will have a key "**responseCode**", which can be checked before proceeding to deal with the acknowledgement response. This key will have one of the following integer values:

- -3 – The request has reached the maximum timeout before Google Play responds
- -2 – Requested feature is not supported by Play Store on the current device.
- -1 – The Play Store service is not connected currently
- 0 – Success
- 1 – User has cancelled the action
- 2 – Network connection is down
- 4 – Requested product is not available for purchase
- 6 – Fatal error during the API action
- 7 – Failure to purchase since item is already owned
- 8 – Failure to consume since item is not owned

Syntax

```
GPBilling_AcknowledgePurchase(purchase_token);
```

Argument	Description	Data Type
purchase_token	The unique string token for the purchase being acknowledged.	String

Cont…/

Contents

Returns

Constant

Constant	Actual Value	Description
gpb_error_unknown	-1	There was an unknown error preventing the Billing API from creating an acknowledgment request.
gpb_no_error	0	The Billing API has created an acknowledgment request correctly.

Example

This example shows how you'd deal with the callback in the Asynchronous IAP event for an acknowledged product (for an example of when the GPBilling_AcknowledgePurchase() function should be called, see the Extended Example for the function GPBilling_PurchaseSubscription()):

```
var _eventId = async_load[? "id"];
switch (eventId)
    {
    case gpb_acknowledge_purchase_response:
        var _map = json_decode(async_load[? "response_json"]);
        var _num = -1;
        if map[? "responseCode"] == 0
            {
            var _sz = ds_list_size(global.CurrentProducts);
            for (var i = 0; i < _sz; ++i;)
                {
                if global.CurrentProducts[| i] == global.IAP_ProductID[0]
                    {
                    global.NoAds = true;
                    _num = i;
                    break;
                    }
                // Add further checks for other products here…
                }
            if _num > -1
                {
                ds_list_delete(global.CurrentProducts, _num);
                ds_list_delete(global.CurrentTokens, _num);
                }
            }
        else
            {
            // Parse the other response codes here
            // and react appropriately
            }
        ds_map_destroy(_map);
        break;
    }
```

GPBilling_ConsumeProduct

Description

This function will consume a purchase. When you receive notification that a purchase has been made, it needs to be consumed or acknowledged with the Google servers within 2 days otherwise it is refunded. Consumable purchases are acknowledged automatically when you call this function, but for non-consumable and subscription purchases, see the function `GPBilling_AcknowledgePurchase()`.

On calling the function initially, it will return one of the constants listed below to inform you of the request status, and if this is **`gpb_no_error`** then an Asynchronous IAP Event will be triggered where the `async_load` DS map will have the key "**id**" which will correspond to the extension constant **`gpb_product_consume_response`**. Additionally, the map will have the key "**response_json**" which will be a JSON string that can be converted into a DS map using the `json_decode()` function.

The decoded JSON will be a DS map which will have *either* the key "**responseCode**" (if there is an error) or the key "**purchaseToken**", which will be the purchase token string of the consumed product.

If you have the "responseCode" key, then it can be checked for one of the following integer values:

- -3 – The request has reached the maximum timeout before Google Play responds
- -2 – Requested feature is not supported by Play Store on the current device.
- -1 – The Play Store service is not connected currently
- 0 – Success
- 1 – User has cancelled the action
- 2 – Network connection is down
- 4 – Requested product is not available for purchase
- 6 – Fatal error during the API action
- 7 – Failure to purchase since item is already owned
- 8 – Failure to consume since item is not owned

Syntax

```
GPBilling_ConsumeProduct(purchase_token);
```

Argument	Description	Data Type
purchase_token	The unique string token for the purchase being acknowledged.	String

Cont…/

Contents

Returns

Constant

Constant	Actual Value	Description
gpb_error_unknown	-1	There was an unknown error preventing the Billing API from creating a consume request.
gpb_no_error	0	The Billing API has created a consume request correctly.

Example

This example shows how you'd deal with the callback in the Asynchronous IAP event for a consumed product (for an example of when the GPBilling_ConsumeProduct() function should be called, see the Extended Example for the function GPBilling_PurchaseProduct()):

```
var _eventId = async_load[? "id"];
switch (eventId)
    {
    case gpb_product_consume_response:
        var _map = json_decode(async_load[? "response_json"]);
        var _num = -1;
        if ds_map_exists(_map, "purchaseToken")
            {
            for (var i = 0; i < ds_list_size(global.CurrentTokens); ++i;)
                {
                if _map[? "purchaseToken"] == global.CurrentTokens[| i]
                    {
                    if global.CurrentProducts[| i] == global.IAP_ProductID[0]
                        {
                        global.Gold += 500;
                        _num = i;
                        break;
                        }
                    // Check any other products here…
                    }
                }
            if _num > -1
                {
                ds_list_delete(global.CurrentProducts, _num);
                ds_list_delete(global.CurrentTokens, _num);
                }
            }
        else
            {
            // Parse the error response codes here
            // and react appropriately
            }
        ds_map_destroy(_map);
        break;
    }
```

GPBilling_Sku_GetDescription

Description

This function will return the descriptive text as defined on the Google Play Developer Console for the given SKU (product ID). You supply the product ID as a string, and the function will return a string with the description. Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

Syntax

```
GPBilling_Sku_GetDescription(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

String

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _price = GPBilling_Sku_GetPrice(global.IAP_PurchaseID[0]);
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, _price);
```

Contents

GPBilling_Sku_GetFreeTrialPeriod

Description

This function will return the Trial Period as defined on the Google Play Developer Console for the given SKU (product ID). You supply the product ID as a string, and the function will return a string in ISO 8601 format, for example "P7D" which would equate to seven days. Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

> **IMPORTANT!** *This function is only valid for subscriptions which have a trial period configured.*

Syntax

```
GPBilling_Sku_GetFreeTrialPeriod(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

N/A

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _period = GPBilling_Sku_GetFreeTrialPeriod(global.IAP_PurchaseID[0]);
var _days = string_digits(_period);
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, "Trial Period: " + _days + " days");
```

GPBilling_Sku_GetIconUrl

Description

This function will return the URL for the icon of the given SKU (product ID) as created on the Google Play Developer Console. You supply the product ID as a string, and the function will return a string with the URL. You can then use the sprite_add() function to retrieve this image (which will trigger an Asynchronous Image Loaded Event) and then display it in your game.

Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

Syntax

```
GPBilling_Sku_GetIconUrl(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

String

Extended Example

In this example, we first call the function to get the URL of the icon for a product (probably in an Asynchronous IAP event, after querying the product details):

```
var _url = GPBilling_Sku_GetIconUrl(global.IAP_PurchaseID[0]);
iap_sprite = sprite_add(_url, 0, false, false, 0, 0);
```

This will trigger an Asynchronous Image Loaded event where you can then store the returned image for drawing:

```
if ds_map_find_value(async_load, "id") == iap_sprite
    {
    if ds_map_find_value(async_load, "status") >= 0
        {
        sprite_index = iap_sprite;
        }
    }
```

GPBilling_Sku_GetIntroductoryPrice

Description

This function will return the introductory price as defined on the Google Play Developer Console for the given SKU (product ID). You supply the product ID as a string, and the function will return a formatted string with the price that includes the currency sign, for example "€3.99". Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

> **IMPORTANT!** *This function is only valid for subscriptions which have an introductory period configured.*

Syntax

```
GPBilling_Sku_GetIntroductoryPrice(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

String

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _price = GPBilling_Sku_GetIntroductoryPrice(global.IAP_PurchaseID[0]);
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, _price);
```

GPBilling_Sku_GetIntroductoryPriceAmountMicros

Description

This function will return the introductory price as defined on the Google Play Developer Console for the given SKU (product ID) in *micros*, where 1,000,000 micros equals one unit of the currency. You supply the product ID as a string, and the function will return an integer value for the price in micros, for example 7990000 (which would be 7.99 in currency). Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

> **IMPORTANT!** *This function is only valid for subscriptions which have an introductory period configured.*

Syntax

```
GPBilling_Sku_GetIntroductoryPriceAmountMicros(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

Integer

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _m = GPBilling_Sku_GetIntroductoryPriceAmountMicros(global.IAP_PurchaseID[0]);
var _c = GPBilling_Sku_GetPriceCurrencyCode(global.IAP_PurchaseID[0]);
var _val = string(_m / 1000000);
var _symbol = “”;
switch (_c)
    {
    case “GBP”: _symbol = “£”; break;
    case “JPY”: _symbol = “¥”; break;
    case “EUR”: _symbol = “€”; break;
    }
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, _symbol + _val);
```

GPBilling_Sku_GetIntroductoryPriceCycles

Description

This function will return the introductory price cycles as defined on the Google Play Developer Console for the given SKU (product ID). You supply the product ID as a string, and the function will return a string of the value for the number of billing cycles that the user will pay the introductory price for, for example “3”. Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

> **IMPORTANT!** *This function is only valid for subscriptions which have an introductory period configured.*

Syntax

```
GPBilling_Sku_GetIntroductoryPriceCycles(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

String

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _price = GPBilling_Sku_GetIntroductoryPrice(global.IAP_PurchaseID[0]);
var _cycles = GPBilling_Sku_GetIntroductoryPriceCycles(global.IAP_PurchaseID[0]);
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, _price)
draw_text(x, y + 80, “for “ + _cycles + “ months”);
```

GPBilling_Sku_GetIntroductoryPricePeriod

Description

This function will return the introductory price period as defined on the Google Play Developer Console for the given SKU (product ID). You supply the product ID as a string, and the function will return a string in ISO 8601 format, for example "P7D" would equate to seven days. Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

> **IMPORTANT!** *This function is only valid for subscriptions which have an introductory period configured.*

Syntax

```
GPBilling_Sku_GetIntroductoryPricePeriod(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

String

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _period = GPBilling_Sku_GetIntroductoryPricePeriod(global.IAP_PurchaseID[0]);
var _days = string_digits(_period);
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, "Offer Lasts For " + _days + " days!");
```

Contents

GPBilling_Sku_GetOriginalJson

Description

This function will return the original JSON corresponding to the given SKU (product ID), containing all the details about the product. You supply the product ID as a string, and the function will return a JSON string that can be decoded into a DS map using the `json_decode()` function. The map contents will correspond to the details listed in the Google Billing documentation (see here for more information). Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

Syntax

```
GPBilling_Sku_GetOriginalJson(sku);
```

Argument	**Description**	**Data Type**
sku	The unique SKU ID of the product.	String

Returns

String

Example

The following code can be called after querying a product to retrieve all the information about it without calling the individual SKU functions:

```
var _json = GPBilling_Sku_GetOriginalJson(global.IAP_PurchaseID[0]);
var _map = json_decode(_json);
global.IAP_PurchaseData[0, 0] = _map[? “price”];
global.IAP_PurchaseData[0, 1] = _map[? “title”];
global.IAP_PurchaseData[0, 2] = _map[? “decription”];
ds_map_destroy(_map);
```

GPBilling_Sku_GetOriginalPrice

Description

This function will return the original price as defined on the Google Play Developer Console for the given SKU (product ID), where the original price is the price of the item before any applicable sales have been applied. You supply the product ID as a string, and the function will return a formatted string with the price that includes the currency sign, for example "€3.99". Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

Syntax

```
GPBilling_Sku_GetOriginalPrice(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

String

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _price = GPBilling_Sku_GetOriginalPrice(global.IAP_PurchaseID[0]);
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, _price);
```

GPBilling_Sku_GetOriginalPriceAmountMicros

Description

This function will return the original price as defined on the Google Play Developer Console for the given SKU (product ID) in *micros*, where 1,000,000 micros equals one unit of the currency. The original price is defined as the price of the item before any applicable sales have been applied, and the value represents the localized, rounded price for a particular currency. You supply the product ID as a string, and the function will return an integer value for the price in micros, for example 7990000 (which would be 7.99 in currency). Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

Syntax

```
GPBilling_Sku_GetOriginalPriceAmountMicros(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

Integer

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _m = GPBilling_Sku_GetOriginalPriceAmountMicros(global.IAP_PurchaseID[0]);
var _c = GPBilling_Sku_GetPriceCurrencyCode(global.IAP_PurchaseID[0]);
var _val = string(_m / 1000000);
var _symbol = “”;
switch (_c)
    {
    case “GBP”: _symbol = “£”; break;
    case “JPY”: _symbol = “¥”; break;
    case “EUR”: _symbol = “€”; break;
    }
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, _symbol + _val);
```

GPBilling_Sku_GetPrice

Description

This function will return the current price as defined on the Google Play Developer Console for the given SKU (product ID). You supply the product ID as a string, and the function will return a formatted string with the price that includes the currency sign, for example “€3.99”. Note that this function requires you to have called GPBilling_QueryProducts() or GPBilling_QuerySubscriptions() *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

Syntax

```
GPBilling_Sku_GetPrice(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

String

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _price = GPBilling_Sku_GetPrice(global.IAP_PurchaseID[0]);
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, _price);
```

GPBilling_Sku_GetPriceAmountMicros

Description

This function will return the current price as defined on the Google Play Developer Console for the given SKU (product ID) in *micros*, where 1,000,000 micros equals one unit of the currency, and the value represents the localized, rounded price for a particular currency. You supply the product ID as a string, and the function will return an integer value for the price in micros, for example 7990000 (which would be 7.99 in currency). Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

Syntax

```
GPBilling_Sku_GetPriceAmountMicros(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

String

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _m = GPBilling_Sku_GetPriceAmountMicros(global.IAP_PurchaseID[0]);
var _c = GPBilling_Sku_GetPriceCurrencyCode(global.IAP_PurchaseID[0]);
var _val = string(_m / 1000000);
var _symbol = “”;
switch (_c)
    {
    case “GBP”: _symbol = “£”; break;
    case “JPY”: _symbol = “¥”; break;
    case “EUR”: _symbol = “€”; break;
    }
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, _symbol + _val);
```

GPBilling_Sku_GetPriceCurrencyCode

Description

This function will return the currency code for the given SKU (product ID). You supply the product ID as a string, and the function will return a string in ISO 4217 format, for example "EUR" would equate the Euro currency. Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

Syntax

```
GPBilling_Sku_GetPriceCurrencyCode(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

String

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _m = GPBilling_Sku_GetPriceAmountMicros(global.IAP_PurchaseID[0]);
var _c = GPBilling_Sku_GetPriceCurrencyCode(global.IAP_PurchaseID[0]);
var _val = string(_m / 1000000);
var _symbol = "";
switch (_c)
    {
    case "GBP": _symbol = "£"; break;
    case "JPY": _symbol = "¥"; break;
    case "EUR": _symbol = "€"; break;
    }
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, _symbol + _val);
```

GPBilling_Sku_GetSubscriptionPeriod

Description

This function will return the subscription renewal period as defined on the Google Play Developer Console for the given SKU (product ID). You supply the product ID as a string, and the function will return a string in ISO 8601 format, for example "P7D" would equate to seven days. Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

IMPORTANT! *This function is only valid for subscription IAPs.*

Syntax

```
GPBilling_Sku_GetSubscriptionPeriod(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

String

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _period = GPBilling_Sku_GetSubscriptionPeriod(global.IAP_PurchaseID[0]);
var _days = string_digits(_period);
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, "Renewal Period: " + _days + " days");
```

GPBilling_Sku_GetTitle

Description

This function will return the title of a given SKU (product ID) as defined on the Google Play Developer Console. You supply the product ID as a string, and the function will return a string with the product title. Note that this function requires you to have called `GPBilling_QueryProducts()` or `GPBilling_QuerySubscriptions()` *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

Syntax

```
GPBilling_Sku_GetTitle(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

String

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _price = GPBilling_Sku_GetPrice(global.IAP_PurchaseID[0]);
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, _price);
```

GPBilling_Sku_GetType

Description

This function will return the in-app purchase type of a given SKU (product ID) as defined on the Google Play Developer Console. You supply the product ID as a string, and the function will return a constant with the IAP product type (see below). Note that this function requires you to have called GPBilling_QueryProducts() or GPBilling_QuerySubscriptions() *first*, but once those have returned their respective Async callbacks, this function can be used anywhere in your game code to retrieve the required information.

Syntax

```
GPBilling_Sku_GetType(sku);
```

Argument	Description	Data Type
sku	The unique SKU ID of the product.	String

Returns

Constant

Constant	Actual Value	Description
gpb_purchase_skutype_inapp	“inapp”	The product is a consumable IAP
gpb_purchase_skutype_subs	“subs”	The product is a subscription IAP.

Example

The following code would be called in the Draw Event of an object used to display the information for a given IAP product:

```
var _name = GPBilling_Sku_GetTitle(global.IAP_PurchaseID[0]);
var _desc = GPBilling_Sku_GetDescription(global.IAP_PurchaseID[0]);
var _price = GPBilling_Sku_GetPrice(global.IAP_PurchaseID[0]);
draw_set_halign(fa_center);
draw_text(x, y + 32, _name);
draw_text(x, y + 48, _desc);
draw_text(x, y + 64, _price);
if GPBilling_Sku_GetType(global.IAP_PurchaseID[0]) == gpb_purchase_skutype_subs
    {
    draw_text(x, y + 80, “Subscription!”);
    }
```

GPBilling_Purchase_GetState

Description

This function can be used to check the current state of a product purchase. You supply the unique purchase token (as a string), and the function will return one of the constants listed below to indicate the current state of the purchase.

Syntax

```
GPBilling_Purchase_GetState(purchase_token);
```

Argument	Description	Data Type
purchase_token	The purchase token for the purchase to check.	String

Returns

Constant

Constant	Actual Value	Description
gpb_purchase_state_pending	3002	The purchase is still pending
gpb_purchase_state_purchased	3001	The purchase has been completed
gpb_purchase_state_unspecified	3000	The API can't retrieve the purchase status for some reason
gpb_error_unknown	-1	There is an unknown issue with the Billing API (possible connection issue)
gpb_error_not_initialised	1	The Billing API has not initialised correctly.

Example

```
if GPBilling_Purchase_GetState(global.CurrentToken) == gpb_purchase_state_purchased
    {
    sprite_index = spr_IconConsume;
    }
```

GPBilling_Purchase_GetSignature

Description

This function will return a string containing the signature of the purchase data that was signed with the private key of the developer. If the function fails, then an empty string "" will be returned.

Syntax

```
GPBilling_Purchase_GetSignature(purchase_token);
```

Argument	Description	Data Type
purchase_token	The purchase token for the purchase to check.	String

Returns

String

Example

```
for (var i = 0; i < ds_list_size(global.CurrentTokens); ++i;)
    {
    var _sig = GPBilling_Purchase_GetSignature(global.CurrentTokens[ i]);
    var _json = GPBilling_Purchase_GetOriginalJson(global.CurrentTokens[ i]);
    if GPBilling_Purchase_VerifySignature(_json, _sig)
        {
        GPBilling_ConsumeProduct(global.CurrentTokens[| i]);
        }
    }
```

GPBilling_Purchase_VerifySignature

Description

This function can be used to verify a purchase before consuming or acknowledging it. You supply a JSON string plus the unique signature for a purchase. You can retrieve these details using the `GPBilling_Purchase_GetSignature()` and `GPBilling_Purchase_GetOriginalJson()` functions, and the function will return `true` if the purchase can be verified, or `false` otherwise.

> **Warning!** *This form of verification **isn't truly secure** because it requires you to bundle purchase verification logic within your app. This logic becomes compromised if your app is reverse engineered. Instead we recommend that you create your own server to verify any product purchases.*

Syntax

```
GPBilling_Purchase_VerifySignature(original_json, signature);
```

Argument	Description	Data Type
original_json	The original JSON related to the purchase being verified	String
signature	The unique signature used to verify the purchase	String

Returns

Boolean

Example

```
for (var i = 0; i < ds_list_size(global.CurrentTokens); ++i;)
    {
    var _sig = GPBilling_Purchase_GetSignature(global.CurrentTokens[ i]);
    var _json = GPBilling_Purchase_GetOriginalJson(global.CurrentTokens[ i]);
    if GPBilling_Purchase_VerifySignature(_json, _sig)
        {
        GPBilling_ConsumeProduct(global.CurrentTokens[| i]);
        }
    }
```

GPBilling_Purchase_GetOriginalJson

Description

This function will return the original JSON string related to a purchase. You supply the unique purchase token (a string) and the function will return a JSON object string that can be decoded into a DS map using the `json_decode()` function. This map will contain all the details about the given purchase. If the function fails, then an empty string "" will be returned.

Syntax

```
GPBilling_Purchase_GetOriginalJson(purchase_token);
```

Argument	Description	Data Type
purchase_token	The purchase token for the purchase to check.	String

Returns

String

Example

```
for (var i = 0; i < ds_list_size(global.CurrentTokens); ++i;)
    {
    var _sig = GPBilling_Purchase_GetSignature(global.CurrentTokens[ i]);
    var _json = GPBilling_Purchase_GetOriginalJson(global.CurrentTokens[ i]);
    if GPBilling_Purchase_VerifySignature(_json, _sig)
        {
        GPBilling_ConsumeProduct(global.CurrentTokens[| i]);
        }
    }
```