Zenbo Store In-app Purchases

Overview

This document describes the fundamental In-app Purchases components and features that you need to understand in order to add In-app Purchases features into your application.

In-app Purchases API

Your application accesses the In-app Purchases (IAP) service using an API that is exposed by the ZenboStore that is installed on Zenbo. The ZenboStore app then conveys purchase requests and responses between your application and the ASUS server. In practice, your application never directly communicates with the ASUS server. Instead, your application sends purchase requests to the ZenboStore app over interprocess communication (IPC) and receives responses from the ZenboStore app. Your application does not manage any network connections between itself and the ASUS server.

You can implement IAP only in applications that you publish through ZenboStore. To complete IAP requests, the ZenboStore app must be able to access the ASUS server over the network.

In-app Products

The products are the digital products that you offer for sale to users from inside your application. Examples of digital products include in-game currency, application feature upgrades that enhance the user experience, and new content for your application.

You can use IAP API to sell only digital content. You can't use IAP to sell physical products, personal services, or anything that requires physical delivery. Unlike with priced applications, there is no refund window after the user has purchased an IAP product.

ZenboStore does not provide any form of content delivery. You are responsible for delivering the digital content that you sell in your applications. IAP products are always explicitly

associated with only one app. That is, one application can't purchase an IAP product that is published for another app, even if they are from the same developer.

Product types

IAP supports different product types to give you flexibility in how you monetize your application. In all cases, you define your products using the Zenbo developer Console.

- Non-consumable Items

  Typically, you would not implement consumption for in-app products that can only be purchased once in your application and provide a permanent benefit. Once purchased, these items will be permanently associated to the user's ASUS account. An example of a non-consumable in-app product is a premium upgrade or a level pack.
- Consumable items

  In contrast, you can implement consumption for items that can be made available for purchase multiple times. Typically, these items provide certain temporary effects. For example, the user's in-game character might gain life points or gain extra gold coins in their inventory. Dispensing the benefits or effects of the purchased item in your application is called provisioning the in-app product. You are responsible for controlling and tracking how in-app products are provisioned to the users.

Zenbo IAP Flow

ZenboStore uses the same backend checkout service that is used for application purchases, so your users experience a consistent and familiar purchase flow. You must have a ZenboStore payments merchant account to use the IAP service on ZenboStore.

To initiate a purchase, your application sends a purchase request for a specific product. ZenboStore then handles all of the checkout details for the transaction, including requesting and validating the form of payment and processing the financial transaction.

When the checkout process is complete, ZenboStore returns the purchase details to your application, such as the order number, the order date and time, and the price paid. At no point

does your application have to handle any financial transactions; that role belongs to ZenboStore.

IAP Library

To simplify development with the IAP API, you can use the IAP Library. This library is an IAP client developed as a wrapper on top of the Android Interface Definition Language file that interacts with the IAP API. You can use the IAP Library to help you focus your development effort on app logic, such as listing products, displaying product details, or launching purchase flows. The IAP Library provides an easier to use alternative to developing with the Android Interface Definition Language file.

IAP (In-app Purchases) Library

Overview

In-app Purchase on ZenboStore provides a straightforward and simple interface for sending IAP requests and managing IAP transactions using ZenboStore. The information below covers the basics of how to make calls from your app to the IAP service using the IAP Library.

The IAP Library provides convenience classes and features, which you can use to integrate the IAP service with your Android apps. The library is a wrapper for the Android Interface Definition Language (AIDL) file that defines the interface to the IAP service. You can use the IAP Library to simplify your development process, allowing you to focus your effort on implementing logic specific to your app, such as displaying in-app products and purchasing items.

Adding IAP library

To add the IAP library, follow these steps:

- Copy the jar file of IAP library to your Android project under /libs.
- Add a dependency “GSON” to your project, specify a dependency configuration such as compile in the dependencies block of your build.gradle file.

```
dependencies {
    //...
    compile 'com.google.code.gson:gson:2.+'
}
```

Initiate a Connection with ZenboStore

To send IAP requests to ZenboStore from your application, you must bind your Activity to the ZenboStore IAP service. The labrary includes convenience classes that handle the binding to the IAP service, so you don’t have to manage the network connection directly.

To set up synchronous communication with ZenboStore, create an ZenboIabHelper instance in your activity's onCreate method, as shown in the following example. In the constructor, pass

the Context for the activity, a string containing the public license key (AppID) that was generated earlier by the Zenbo Developer Console, and the application version.

```
ZenboIabHelper mHelper;

@Override
public void onCreate(Bundle savedInstanceState) {
    // ...
    String appID;
    String appVersion;
    mHelper = new ZenboIabHelper(this, appID, String.valueOf(BuildConfig.VERSION_CODE));
}
```

Next, perform the service binding by calling the initIabHelper method on the ZenboIabHelper instance that you created, as shown in the following example. Pass the method an IabInitListener instance, which is called once the ZenboIabHelper completes the asynchronous setup operation. The listener is notified by onIabInitResult with the result and the message.

```
@Override
protected void onResume() {
    super.onResume();
    mHelper.initIabHelper(new ZenboIabHelper.IabInitListener {
        @Override
        public void onIabInitResult(boolean success, String message) {
            //...
        }
    });
}
```

To unbind and free your system resources, call the **ZenboIabHelper's dispose** method when your Activity is destroyed, as shown in the following example.

```
@Override
protected void onDestroy() {
    super.onDestroy();
    if(mHelper != null) {
        mHelper.dispose();
        mHelper = null;
    }
}
```

Query Items Available for Purchase

You can query ZenboStore to retrieve details of the IAP products that are associated with your

application (such as the product's price, title, description, and type). This is useful, for example, when you want to display a listing of unowned items that are still available for purchase to users.

To retrieve the product details, call getProductList (String skuID, String itInType, String pubState, IStoreIabCbk listener) on your ZenboIabHelper instance.

- The arguments "skuID", "itInType", and "pubState" indicates which product details that you want to query.
- Finally, the IStoreIabCbk argument specifies a listener that is notified when the query operation has completed and handles the query response.

If you use the convenience classes provided in the sample, the classes will handle background thread management for IAP requests, so you can safely make queries from the main thread of your application.
The following code shows how you can retrieve the details for the product with skuID and itInType that you previously defined in the Zenbo Developer Console.

```
mHelper.getProductList("005 ", "2", "", mGetProductListener);

IabResultListener mGetProductListener = new IabResultListener() {
    @Override

    public void onIabResult(final IabResult result) {
        if(result.isSuccess()) {
            Gson gson = new Gson();
            ProductList list = gson.fromJson(result.getMessage(), ProductList.class);
            //... use data
        } else {
            //... handle error
        }
    }
};
```

If you give empty string parameters for the skuID, itInType, and pubState to the API, the result of the API will be the list of all products defined in the Zenbo Developer Console.

```
mHelper.getProductList("", "", "", mGetProductListener);
```

Purchasing In-app Billing Products

Once your application is connected to ZenboStore, you can initiate purchase requests for in-app products. ZenboStore provides a checkout interface for users to enter their payment method, so your application does not need to handle payment transactions directly.

You can also query ZenboStore to quickly retrieve the list of purchases that were made by the user. This is useful, for example, when you want to restore the user's purchases when your user launches your app.

Purchase an item

To start a purchase request from your app, call public void **setOrderAndPay(Activity activity, int requestCode, String skuID, String totalAmount, String developerPayload)** on your **ZenboIabHelper** instance. You must make this call from the main thread of your Activity.

- The first argument is the calling Activity.
- The second argument is a "request code" that identifies your request. When you receive the result Intent, the callback provides the same request code so that your app can properly identify the result and determine how to handle it.
- The third argument is the SKU ID of the item to purchase.
- The fourth argument is the price of the item to purchase.
- The fifth argument contains a 'developer payload' string that you can use to send supplemental information about an order (it can be an empty string).
-

```
int REQ_SET_ORDER = 1000;

mHelper.setOrderAndPay(MainActivity.this, REQ_SET_ORDER, "sku_001", "20", “Test payload");
```

ZenboStore returns this request code to the calling Activity’s onActivityResult along with the purchase response. The data intent contains IabResult, which is the status of this purchase action for the product item.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
	super.onActivityResult(requestCode, resultCode, data);
	if(requestCode == REQ_SET_ORDER) {
		if(data != null) {
			IabResult result = data.getParcelableExtra(IabResult .EXTRA_IAB_API_RESULT);
			//handle result. getResponse() and result.getMessage ()
		}
	}
}
```

Query purchased items

To retrieve the user's purchases from your app, call **getOrderList(final String orderNo, final IabResultListener listener)** on your **ZenboIabHelper** instance. The **IabResultListener** argument specifies a listener that is notified when the query operation has completed and handles the query response. It is safe to make this call from your main thread. If the character string parameter of **orderNo** has been set empty, the result of the API will be the list of all order purchases.

```
mHelper.getOrderList("", mGetOrderListener);

//...

IabResultListener mGetOrderListener = new IabResultListener() {
	@Override
	public void onIabResult(final IabResult result) {
		if(result.isSuccess()) {
			Gson gson = new Gson();
			OrderList list = gson.fromJson(result.getMessage(), OrderList.class);
		  //... use data
		} else {
		  //... handle error
		}
	}
};
```