Unity SDK for Xiaomi (IAP)

1. Overview

2. Login & Purchase Flow
    2.1 Stand-alone login & purchase
    2.2 Online login & purchase

3. Technical Integration
    3.1 Onboarding to Unity
    3.2 Server side integration
        3.2.1 Verify Xiaomi Logins
        3.2.2 Check Order Status
        3.2.3 OrderAttempt Callback Notification
    3.3 Client side integration
        3.3.1 SDK Initialization
        3.3.2 SDK Binding
        3.3.3 SDK Login
        3.3.4 Catalog Setting
        3.3.5 Purchase Validation
            3.3.5.1 Local Validation (Demonstrated in IAPDemo.cs)
            3.3.5.2 Server Validation
        3.3.6 Duplicate Purchasing
            Unity 5.3 & Unity 5.4
    Appendix
        Best Practice
            Implementing local inventory

1. Overview

1.1 What is Unity IAP for Xiaomi IAP

Unity IAP provides an easy way to integrate Xiaomi IAP with Unity.

1.2 Process of using Unity IAP

Please append ".mi" to your package name, e.g. "com.unity.mygame.mi"

a. Technical Integration

I. Onboarding to Unity
II. Server side integration
III. Client side integration

b. Build and Submit APKs

2. Login & Purchase Flow

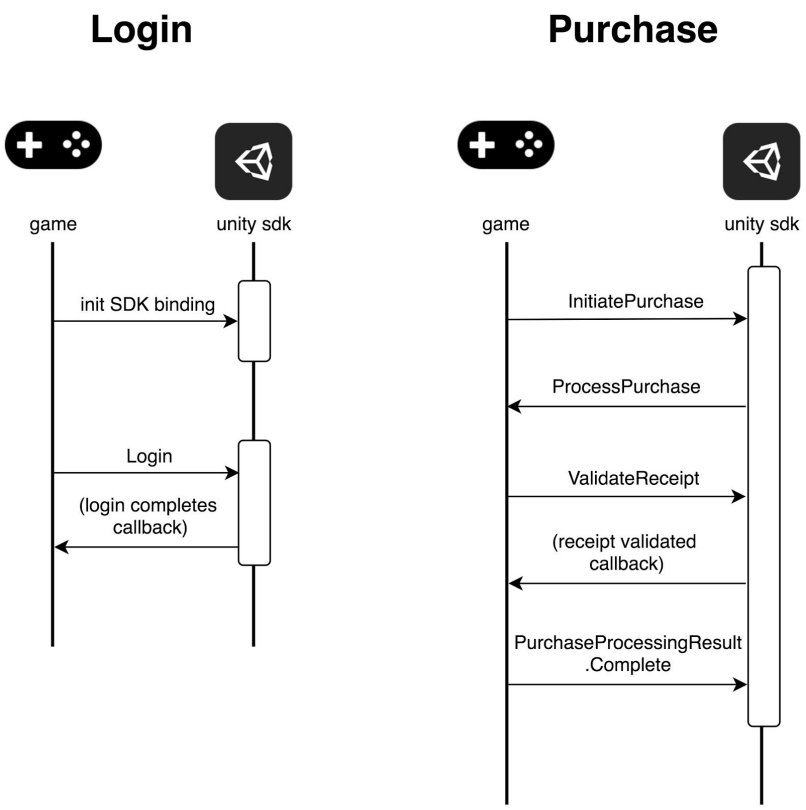
Unity IAP both supports the stand-alone game (without a game server) and online game (with a game server).

2.1 Stand-alone login & purchase

In the ‘init SDK binding’ step, the Xiaomi appId, Xiaomi appKey (both of these given by Xiaomi) and the Unity Client Id, Unity Client Key (both of these given by Unity, see 2.1 Onboarding to Unity for details) will be used to initialize the Unity IAP SDK.

Since there is no game servers, the purchase should be validated on the client side. Sample of the validation can be found in 2.3.5 Purchase Validation. After the validation succeeds, the purchase can be completed.

The flow’s details are shown as below.

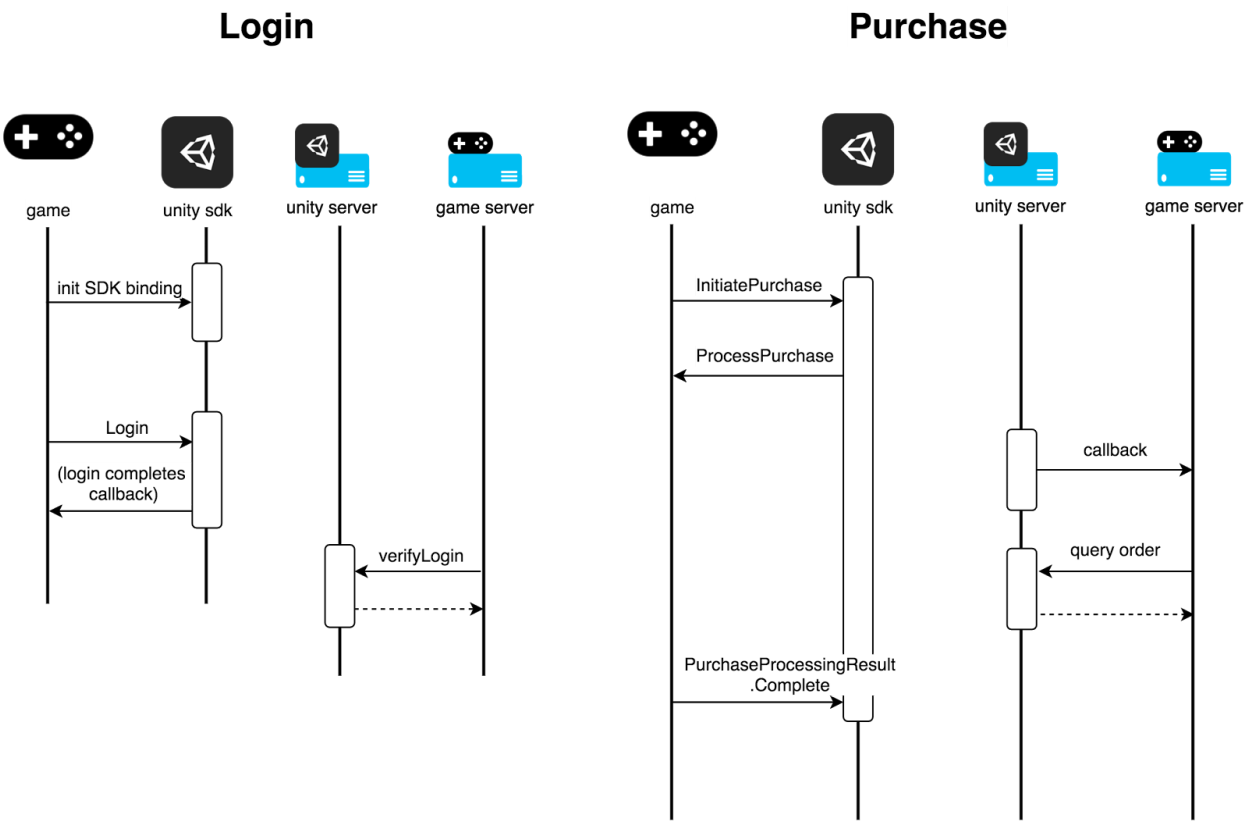


2.2 Online login & purchase

Different from stand-alone flows, the game server can verify the login and receive purchase order callback or initiatively query purchase order to/from the Unity IAP server (the details of these APIs are shown in 2.2 Server side integration).

The flow's details are shown as below.

(Note: Unity IAP for Xiaomi does not support inventory tracking, hence please do not use *PurchaseProcessingResult.Pending,* which will not work for the purchase to be called again when the app restart. Meanwhile, in *Appendix: Best Practice* there provides an implementation of local inventory to prevent the losing of unfinished transaction when server-side validation meets problems, for example, network issues.)



3. Technical Integration

3.1 Onboarding to Unity

Before integration can start, Game developer first needs to onboard to Unity IAP and will obtain the following developer credentials (these keys are binding to the project and bundle id):

Unity Client Id

Unity Client Key

Unity Client Secret

Unity RSA public key

Meanwhile, Game needs to provide the appId, appKey and appSecret offered by Xiaomi, and a callback URL of the Game server (if there exists one) to receive order status update.

Before going live, Game developer will also need to provide Game server IP addresses to be whitelisted on Unity IAP's production server.

3.2 Server side integration

Product environment: https://cn-api.unity.com
Debug environment: https://cn-api-debug.unity.com
Server sample code (Java) could be found at https://unitytech.github.io/channel/server-sample-code, it contains verifying Xiaomi logins (see 2.2.1), checking order status (see 2.2.2), signing request data, and verifying Unity IAP server callbacks.

3.2.1 Verify Xiaomi Logins

GET
/v1/login-attempts/verifyLogin?
userLoginToken=<userLoginToken>&sign=<sign>

Parameters:

Attribute Name	Required?	Description	format	Sample
userLoginToken	Yes	Unity user login token returned by client SDK when finishing login	String	eyJleHRlcm5hbEFwcElkIjogIjQwMzgwNTU0MzkiLCAiZXh0ZXJuYWxUeXBlIjogIkZBS0VDSEwiLCAiZXh0ZXJuYWxVaWQiOiAiRkFLRUNITF

				9seXhoayIsICJleHRlcm5hbFNlc3Npb24iOiAiRkFLRUNITF9TRVNTSU9OIn0=
sign	Yes	md5 with userLoginToken and Unity Client Secret. Check the algorithm in the end of the doc	String	dc03e329e44c80b73d5bbfd30207de8f

Response: in Json

Attribute Name	Required?	Description	Format	Sample
success	Yes	result of verify Xiaomi login	bool	true
errMsg	Yes	the error message of the login if login failed	String	Invalid appId error

Sample:

Request:

GET
/v1/login-attempts/verifyLogin?
userLoginToken=eyJleHRlcm5hbEFwcEIkIjogIjQwMzgwNTU0MzkiLCAiZXh0ZXJuYWxUeXBlIjogIkZBS0VDSEwiLCAiZXh0ZXJuYWxVaWQiOiAiRkFLRUNITF9seXhoayIsICJleHRlcm5hbFNlc3Npb24iOiAiRkFLRUNITF9TRVNTU9OIn0=&sign=ea24156d9c6fa3d527e99b9d644068ab

Response:

{

“success”: true

}

3.2.2 Check Order Status

GET

/v1/order-attempts/query?cpOrderId=<cpOrderId>&clientId=<clientId>&orderQueryToken=<orderQueryToken>&sign=<sign>

Parameters:

Attribute Name	Required ?	Description	Format	Sample
cpOrderId	Yes	order id assigned by game , or Unity if game will not generate it	String	66mea52wne
clientId	Yes	client id assigned by Unity after onboarding	String	Q4AnJDW2-rxLAPujqrk1zQ
orderQueryToken	Yes	order query token returned by client SDK when finishing purchase	String	XIAOMI
sign	Yes	md5 with cpOrderId, clientId, orderQueryToken and Unity Client Secret. Check the algorithm in the end of the	String	7277da780d1102864ee5818c2730c74e

		doc		

Response: in Json

Attribute Name	Required?	Description	Format	Sample
id	Yes	orderAttempt id assigned by Unity	String	27487794382 6
clientId	Yes	client id returned after Game onboarding to Unity IAP	String	Q4AnJDW2-r xLAPujqrk1z Q
cpOrderId	Yes	order id assigned by game , or Unity if game will not generate it	String	66mea52wne
uid	Yes	user id assigned by Xiaomi	String	15400813498
status	Yes	status of the orderAttempt	String	SUCCESS
productId	Yes	product id of the product associated with the orderAttempt	String	Product_1
payFee	Yes	pay fee of this order (in channel unit)	Integer	1
quantity	Yes	quantity of the product	Integer	1
notified	Yes	whether the orderAttempt's status is notified to Game server	Boolean	No

currency	Yes	currency	ISO 4217	CNY
country	Yes	country	ISO 3166-2	CN
paidTime	Yes	orderAttempt paid time	ISO8601 yyyy-MM-dd Thh:mm:ssXXX, UTC timezone	2017-03-08T06:43:20Z
createdTime	Yes	orderAttempt created time	ISO8601 yyyy-MM-dd Thh:mm:ssXXX, UTC timezone	2017-03-08T06:43:20Z
updatedTime	Yes	orderAttempt updated time	ISO8601 yyyy-MM-dd Thh:mm:ssXXX, UTC timezone	2017-03-08T06:43:20Z
createdBy	Yes	created by which user (only for support use)	String	274877943822
updatedBy	Yes	updated by which user (only for support use)	String	0
rev	Yes	the revision of the orderAttempt (only for update)	String	0
extension	No	extesion	Json String	{"abc" : "123"}

Sample:

Request:

GET

/v1/order-attempts/query?cpOrderId=<66mea52wne>&clientId=<Q4AnJDW2-rxLAPujqrk1zQ>&orderQueryToken=<eyJjaGFubmVsVHlwZSI6ICJGQUtFQ0hMIiwgImNoYW5uZWxVaWQiOiAiRkFLRUNITF9oeGQwNSJ9>&sign=<debd0018911d263e23dfb729207b905c>

Response:


```
{
	"createdBy" : "0",
	"updatedBy" : "0",
	"createdTime" : "2017-03-08T06:43:20Z",
	"updatedTime" : "2017-03-08T06:43:20Z",
	"id" : "274877943826",
	"userId" : "274877943822",
	"status" : "SUCCESS",
	"cpOrderId" : "66mea52wne",
	"clientId" : "Q4AnJDW2-rxLAPujqrk1zQ",
	"uid" : "15400813498",
	"productId" : "Product_1",
	"payFee" : 1,
	"quantity" : 1,
	"paidTime" : "2017-03-08T06:43:20Z",
	"notified" : false,
	"currency" : "CNY",
	"country" : "CN",
	"rev" : "1"
}
```


3.2.3 OrderAttempt Callback Notification

Unity IAP server will notify the Game server with the order payment result. Please implement the http GET and accept following query parameters.

Attribute Name	Required?	Description	Format
signData	Yes	the order content which is in Json String format (details can be seen in the table below)	Json String
signature	Yes	RSA signature with the signData. Check the algorithm in the end of the doc	String

signData content:

Attribute Name	Required?	Description	Format	Sample
orderAttemptId	Yes	orderAttempt id assigned by Unity	String	274877943826
status	Yes	status of the orderAttempt	String	SUCCESS
productId	Yes	product id of the product associated with the orderAttempt	String	Product_1
quantity	Yes	quantity of the product	Integer	1
extension	No	extesion	Json String	{"abc" : "123"}
paidTime	Yes	orderAttempt paid time	yyyy-MM-ddThh:mm:ssXXX, GMT+8 timezone	2017-03-08 06:43:20
cpOrderId	Yes	order id assigned by Game	String	66mea52wne
clientId	Yes	client id returned	String	P-wU0n9pbyQ

		after Game onboarding to Unity IAP		1ulks4kHX_Q
country	Yes	currency	String of ISO 4217	CNY
currency	Yes	country	String of ISO 3166-2	CN
payFee	Yes	pay fee of this order (in channel unit)	String	1

Response:

Please return "ok". Or there will be retry notification in a time range.

Enum types:

1. status: SUCCESS, FAILED, UNCONFIRMED

Signature verification:

1. Api call:
    a. Any api call to Unity IAP server should have the parameter of signature.
    b. Use all the parameters to calculate the sign if they are not null or empty.
    c. The parameters should be urlDecoded.
    d. Sort the parameters by alphabet of the attribute names, and then concatenate the attribute values with the delimiter of &, finaly append the *Unity Client Secret* at the end of the string to be signed
    e. Do not use the sign in the sign string.
    f. Use md5 to hash the string. Use the lower case result to verify sign

Example:

When there is a call look like:

GET

/v1/login-attempts/verifyLogin?

userLoginToken=eyJleHRlcm5hbEFwcElkIjogIjQwMzgwNTU0MzkiLCAiZXh0ZXJuYWxUeXBlIjogIkZBS0VDSEwiLCAiZXh0ZXJuYWxVaWQiOiAiRkFLRUNITF9seXhoayIsICJleHRlcm5hbFNlc3Npb24iOiAiRkFLRUNITF9TRVNTSU9OIn0=&sign=ea24156d9c6fa3d527e99b9d644068ab

Then the string to be signed is
eyJleHRlcm5hbEFwcElkIjogIjQwMzgwNTU0MzkiLCAiZXh0ZXJuYWxUeXBlIjogIkZBS0VDSEwiLCAiZXh0ZXJuYWxVaWQiOiAiRkFLRUNITF9seXhoayIsICJleHRlcm5hbFNlc3Npb24iOiAiRkFLRUNITF9TRVNTSU9OIn0=&{SECRET}
where {SECRET} is the *Unity Client Secret* returned after onboarding.

2. Callback:

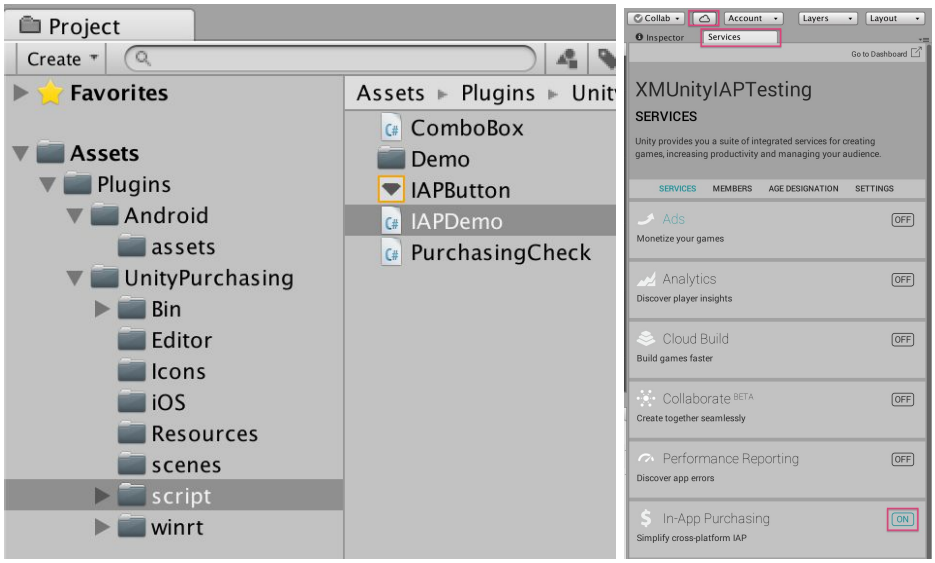
When Unity IAP server gives the callback notification to Game server, it will will use RSA signature for verification. Please use the Unity RSA public key returned after onboarding to verify the *signData* with the *signature*.

3.3 Client side integration

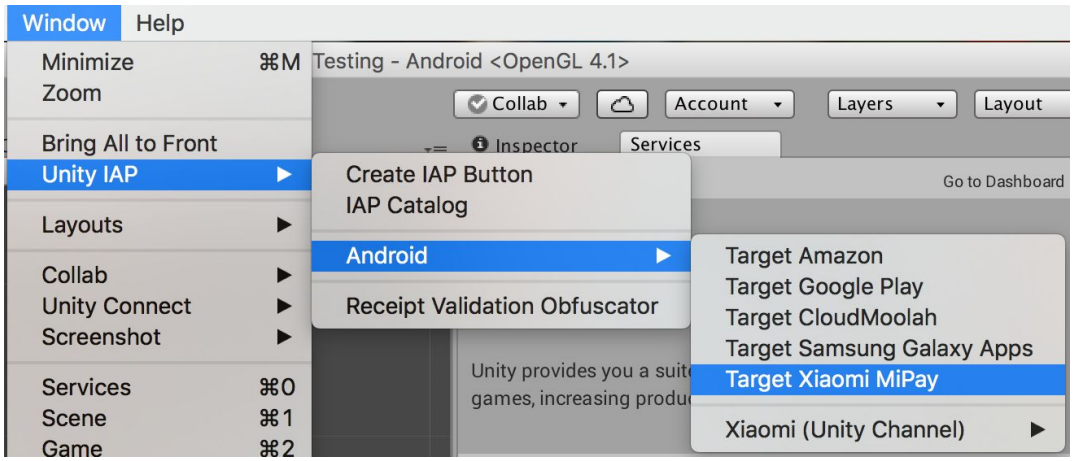
Install the “UnityChannel + Xiaomi UnityIAP beta plugin” in the game project.

Look at the sample script in /Assets/Plugins/UnityPurchasing/script/IAPDemo.cs. This is copied into the Unity game project when the UnityIAP plugin is installed.

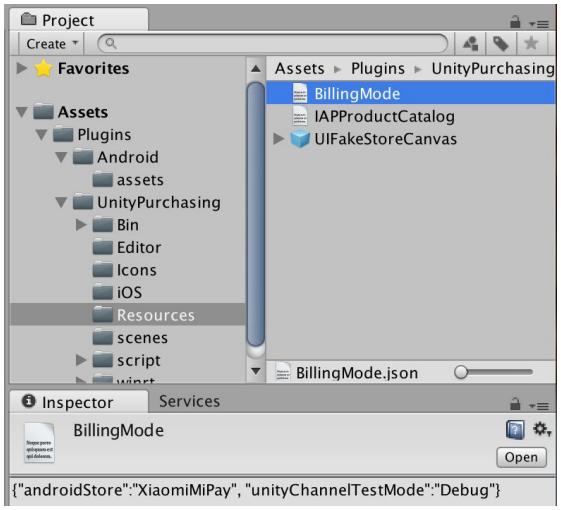
See Getting Started with Unity IAP to enable the “In-App Purchasing Service” for the project.



To build APK for Xiaomi MiPay store (not Samsung GALAXY, or GooglePlay) target the store with menu, “Window” -> “Unity IAP” -> “Android” -> “Target Xiaomi MiPay”. This enables Xiaomi + UnityChannel Java SDK. Use this to target different supported Android app stores, before creating an APK build.



This is saved in a file:



Below will show some code samples for how to use Unity IAP, all the details could be found in the IAPDemo.cs

3.3.1 SDK Initialization

```
private IUnityChannelExtensions m_UnityChannelExtensions;
public void OnInitialized (IStoreController controller, IExtensionProvider extensions) {
        m_UnityChannelExtensions = extensions.GetExtension<IUnityChannelExtensions> ();
}
```

3.3.2 SDK Binding

```
Action initializeUnityIap = () => {
        // Now we're ready to initialize Unity IAP.
        UnityPurchasing.Initialize(YourImplementationOfIStoreListener, builder);
};

AppInfo unityChannelAppInfo = new AppInfo();
// Xiaomi appId
unityChannelAppInfo.appId = "abc123";
// Xiaomi appKey
unityChannelAppInfo.appKey = "efg456";
// UnityChannel Client Id
unityChannelAppInfo.clientId = "hij789";
```

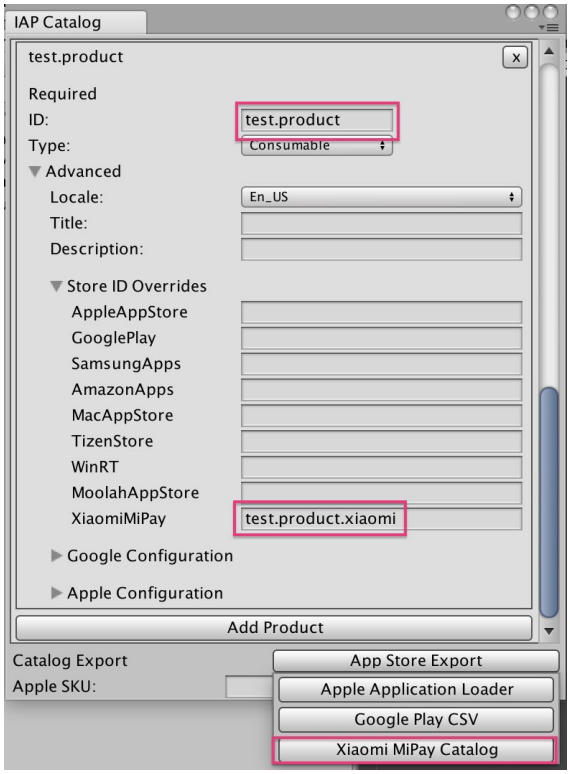
```
// UnityChannel Client Key
unityChannelAppInfo.clientKey = "klm012";

unityChannelLoginHandler = new UnityChannelLoginHandler();
unityChannelLoginHandler.initializeFailedAction = (string message) => {
        Debug.LogError("Failed to initialize and login to UnityChannel: " + message);
};
unityChannelLoginHandler.initializeSucceededAction = () => {
        initializeUnityIap();
};
// Please do below initializations in the Awake() method
StoreService.Initialize(unityChannelAppInfo, unityChannelLoginHandler);
```

3.3.3 SDK Login

```
// The login succeeded callback will the userInfo as the parameter, in which:
// string userInfo.channel: the channel name, e.g. 'XIAOMI'
// string userInfo.userId: the channel name and the channel uid,
// string userInfo.userLoginToken: Unity user login token used for server side 'verifyLogin' api
unityChannelLoginHandler.loginSucceededAction = (UserInfo userInfo) => {
        m_IsLoggedIn = true;
        // get channel type (XIAOMI) by userInfo.channel
        // get channel uid by userInfo.userId
        // get user login token by userInfo.userLoginToken
};
unityChannelLoginHandler.loginFailedAction = (string message) => {
        m_IsLoggedIn = false;
        Debug.LogError("Failed logging into UnityChannel. " + message);
};
StoreService.Login(unityChannelLoginHandler);
```

3.3.4 Catalog Setting

On the menu, “Window” -> “Unity IAP” -> “IAP Catalog”. Set the product “ID”.
In the “Advanced” -> “Store ID Overrides”, set the product id in “XiaomiMiPay”. Note: Default “XiaomiMiPay” value is the same as for product “ID”.



Next, to use the IAP Catalog product list in the script of the game project, either:

a) Read the catalog file. Remove all calls to “builder.AddProduct(...)” in your copy of IAPDemo.cs and replace that code with this code which will read the IAP Catalog:

```
var catalog = ProductCatalog.LoadDefaultCatalog();
foreach (var product in catalog.allProducts) {
        if (product.allStoreIDs.Count > 0) {
                var ids = new IDs();
                foreach (var storeID in product.allStoreIDs) {
                        ids.Add(storeID.id, storeID.store);
                }
                builder.AddProduct(product.id, product.type, ids);
        } else {
                builder.AddProduct(product.id, product.type);
        }
}
```

Or:

b) Manually input each product in the catalog. Add calls to “builder.AddProduct(...)” for each Product ID in IAP Catalog.

3.3.5 Purchase Validation

3.3.5.1 Local Validation (Demonstrated in IAPDemo.cs)

If your game doesn’t have a server, you can validate receipt on the client

1) Add you public key in “Windows” -> “Unity IAP” -> “Receipt Validation Obfuscator” -> “Unity Channel Public Key”, and click “Obfuscate Secrets”
2) Set “fetchReceiptPayloadOnPurchase” to true

```
builder.Configure<IUnityChannelConfiguration>().fetchReceiptPayloadOnPurchase = true
```

3) Using CrossPlatformValidator to validate the receipt. For example:

```
private CrossPlatformValidator validator = new CrossPlatformValidator(GooglePlayTangle.Data(),
                                        AppleTangle.Data(), UnityChannelTangle.Data(), appIdentifier);

public PurchaseProcessingResult ProcessPurchase(PurchaseEventArgs e)
  {
    ……
    ……
    // Local validation is available for GooglePlay, Apple, and UnityChannel stores
     if (m_IsUnityChannelSelected && m_FetchReceiptPayloadOnPurchase) {
        try {
Debug.log(...)
            var result = validator.Validate(e.purchasedProduct.receipt);
            Debug.Log("Receipt is valid. Contents:");
            foreach (IPurchaseReceipt productReceipt in result) {
               Debug.Log(productReceipt.productID);
               Debug.Log(productReceipt.purchaseDate);
               Debug.Log(productReceipt.transactionID);

               UnityChannelReceipt unityChannel = productReceipt as UnityChannelReceipt;
               if (null != unityChannel) {
                  Debug.Log(unityChannel.productID);
                  Debug.Log(unityChannel.purchaseDate);
                  Debug.Log(unityChannel.transactionID);
               }
          }
```

```
            // For improved security, consider comparing the signed
            // IPurchaseReceipt.productId, IPurchaseReceipt.transactionID, and other data
            // embedded in the signed receipt objects to the data which the game is using
            // to make this purchase.
        }
    } catch (IAPSecurityException ex) {
        Debug.Log("Invalid receipt, not unlocking content. " + ex);
        return PurchaseProcessingResult.Complete;
    }
}
```

3.3.5.2 Server Validation

Please refer to Section 3.2. The **signData** and **signature** can be obtained 1) through "ValidateReceipt" API, as shown in "IAPDemo.cs":

```
string txId = m_LastTransationID;
m_UnityChannelExtensions.ValidateReceipt(txId, (bool success, string signData, string signature) =>
{
    Debug.LogFormat("ValidateReceipt transactionId {0}, success {1}, signData {2}, signature {3}",
        txId, success, signData, signature);

    // May use signData and signature results to validate server-to-server
});
```

2) through Product.receipt Payload part. Payload is a JSON hash with the following keys and values:

gameOrderId	i.e. cpOrderId
productCode	The product code.
orderQueryToken	A token used to check order status as discussed in section 3.2.2
json	The **singData**, JSON encoded string contain detailed purchase information
signature	A signature for the json parameter.

signData content:

Attribute Name	Required?	Description	Format	Sample
orderAttemptId	Yes	orderAttempt id assigned by Unity	String	274877943826

status	Yes	status of the orderAttempt	String	SUCCESS
productId	Yes	product id of the product associated with the orderAttempt	String	Product_1
quantity	Yes	quantity of the product	Integer	1
extension	No	extesion	Json String	{"abc" : "123"}
paidTime	Yes	orderAttempt paid time	yyyy-MM-dd hh:mm:ss, GMT+8 timezone	2017-03-08 06:43:20
cpOrderId	Yes	order id assigned by Game	String	66mea52wne
clientId	Yes	client id returned after Game onboarding to Unity IAP	String	P-wU0n9pbyQ 1ulks4kHX_Q
country	Yes	currency	String of ISO 4217	CNY
currency	Yes	country	String of ISO 3166-2	CN
payFee	Yes	pay fee of this order (in channel unit)	String	1

3.3.6 Duplicate Purchasing

Unity 5.5+

```
// When user repeats purchasing a non-consumable they already own,
// the PurchaseFailureReason will be set as DuplicateTransaction.
public void OnPurchaseFailed(Product item, PurchaseFailureReason r) {
```

```
        if (PurchaseFailureReason.DuplicateTransaction == r) {
                // do something for duplicate pruchasing
        }
        m_PurchaseInProgress = false;
}
```

Unity 5.3 & Unity 5.4

```
/// <summary>
/// This will be called is an attempted purchase fails.
/// </summary>
public void OnPurchaseFailed(Product item, PurchaseFailureReason r)
{
    Debug.Log("Purchase failed: " + item.definition.id);
    Debug.Log(r);

    if (m_IsUnityChannelSelected && r == PurchaseFailureReason.Unknown)
    {
        // In Unity 5.3 and 5.4 the enum PurchaseFailureReason.DuplicateTransaction is
        // not available (is available in 5.5+) and can be substituted with
        // this call. Use this in OnPurchaseFailed for Unknown to determine
        // if the user already owns this item and that it can be added to
        // their inventory, if not already present.
        var extra = m_UnityChannelExtensions.GetLastPurchaseError();
        if (extra.Contains("DuplicateTransaction"))
        {
            // Unlock `item` in inventory if not already present.
        }
    }

    m_PurchaseInProgress = false;
}
```

Note:

Developers can use the *product.availableToPurchase* bool field to determine if a product is available from the Store's Product Catalog. This field does not indicate ownership. Developers can use the *product.hasReceipt* bool immediately after purchasing to determine if a product is owned by the user, but this field will be reset to false when the user restarts the application. There is essentially no support for Product Inventory in Unity IAP. Therefore developers need to implement their own persistent Inventory for some purpose, such as, restoring transaction.

Appendix

Best Practice

a. Implementing local inventory

Unity IAP for Xiaomi does not support inventory tracking, which can lead to a pending transaction being lost i.e. returning *PurchaseProcessingResult.Pending* will not lead method *ProcessPurchase* called again. If you want to do server side integration and make an unfinished transaction able to be handled again after a temporary network issue, you can implement a simple local inventory to record the status of transactions. Below shows a code sample:

First, store the transaction with the device storage when the purchase completes:

```
public PurchaseProcessingResult ProcessPurchase(PurchaseEventArgs e) {
        ...
        if (m_IsUnityChannelSelected) {
                var unifiedReceipt = JsonUtility.FromJson<UnifiedReceipt>(e.purchasedProduct.receipt);
                if (unifiedReceipt != null && !string.IsNullOrEmpty(unifiedReceipt.Payload)) {
                        var purchaseReceipt =
                                JsonUtility.FromJson<UnityChannelPurchaseReceipt>(unifiedReceipt.P
                        ayload);
                        // Store the purchase with persistent inventory
                        InventoryItem inventoryItem = new InventoryItem();
                        inventoryItem.gameName = "...";
                        inventoryItem.productId = purchaseReceipt.productCode;
                        inventoryItem.productStatus = "PURCHASED";
                        inventoryItem.productReceipt = e.purchasedProduct.receipt;
                        saveInventory (e.purchasedProduct.transactionID, inventoryItem);

                        // client-to-server validate purchase
                        // unlock content

                        inventoryItem.productStatus = "PROVISIONED";
                        saveInventory (e.purchasedProduct.transactionID, inventoryItem);
                }
        }
        return PurchaseProcessingResult.Complete;
}

[System.Serializable]
public class InventoryItem {
        public string gameName;
        public string productId;
        public string productStatus; // purchased -> provisioned
        public string productReceipt; // product receipt generated by Unity IAP
```

```
}

private Dictionary<string, InventoryItem> inventory = new Dictionary<string, InventoryItem>();

private void saveInventory(string transactionID, InventoryItem item) {
        inventory [transactionID] = item;
        BinaryFormatter bf = new BinaryFormatter();
        // custom file name
        FileStream file = File.Open (Application.persistentDataPath + "/localInventory.data",
                                     FileMode.Create);
        bf.Serialize(file, inventory);
        file.Close();
}
```

Then, each time the application restarts, you can load the inventory in initialization and check if some transaction needs further handling:

```
public void OnInitialized(IStoreController controller, IExtensionProvider extensions) {
        ...
        // load persistent inventory and treat unfinished transaction
        loadInventory();
        List<string> keys = new List<string> (inventory.Keys);
        foreach (string key in keys) {
                string transactionID = key;
                InventoryItem inventoryItem = inventory[key];
                if (inventoryItem.productStatus == "PURCHASED") {

                        // Step 1. validate purchase with inventoryItem.productReceipt
                        // validator.Validate(...)
                        // Step 2. after validation succeeded, unlock content
                        // Step 3. update product status
                        inventoryItem.productStatus = "PROVISIONED";
                        Debug.Log ("continue transaction in initialization. product id: " +
                                   inventoryItem.productId);
                        // Step 4.
                        saveInventory(transactionID, inventoryItem);
                }
        }
        …
}

private void loadInventory() {
        if (File.Exists(Application.persistentDataPath + "/localInventory.data")) {
                BinaryFormatter bf = new BinaryFormatter();
                FileStream file = File.Open(Application.persistentDataPath + "/localInventory.data",
                                            FileMode.Open);
                inventory = (Dictionary<string, InventoryItem>)bf.Deserialize(file);
                file.Close();
        }
}
```