

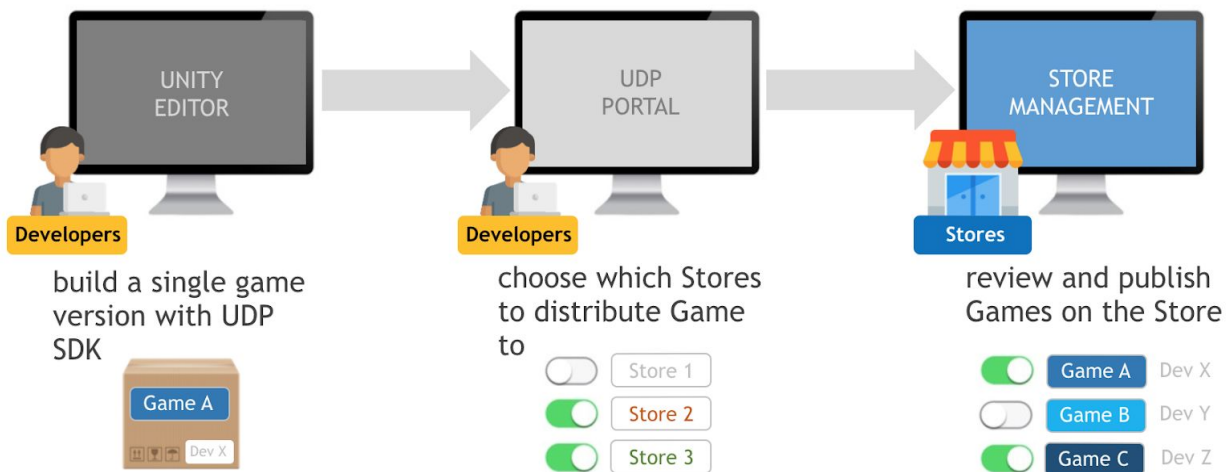
Unity Distribution Platform

1. Overview

Unity Distribution Platform (UDP) streamlines your entire app distribution process. You can seamlessly build for, and deploy to, the following partner app stores:

- Cloud Moolah
- Aptoide, OneStore, Xiaomi, and Huawei (coming soon)

UDP lets you implement your IAPs at one time only, automatically build store-specific packages, and publish these repackaged game or app directly to the stores. You can also submit to selected stores directly on the UDP portal.



To publish your game or app to a partner store:

1. [Configure your game for UDP](#)
2. [Implement IAP](#)
3. [Build your UDP game file and push it to the UDP portal](#)
4. [Manage your game on the UDP portal](#)
5. [Publish your game to partner stores](#)

Prerequisites

- UDP is supported in Unity 5.6.1 or higher.
- Currently, UDP only supports Android.

Quick start guide

This section contains a brief outline of the steps you need to take to publish your game to a partner store.

Configuring your game for UDP

Firstly, set up UDP by installing the [UDP SDK](#).

Tip: If you are currently using **Unity IAP**, UDP is included in the Unity IAP package as of version 1.21.

When you have installed UDP, generate a Unity Client and Test Accounts:

- In the Unity Editor, select **Window > Unity UDP > Game Settings**.
- Select the **Game Settings** and, in the **Inspector** window,
 - Select **Generate Unity Client** to populate the credentials UDP needs for client-to-server communication.
 - Generate test accounts by entering **Email** and **Password** details.

For detailed information, please refer to [Configuring Unity Distribution Platform](#).

Implementing IAP (in-app purchase)

If you have in-app contents to sell, implement IAP inside your game. UDP IAP lets you easily manage your items in all stores.

To configure an IAP catalog, either:

- Use the UDP IAP catalog (**Window > Unity UDP > IAP Catalog**) in the Unity Editor.
- Or, use the UDP portal.

Tip

If you use Unity IAP, configure the IAP catalogs of Unity UDP (**Window > Unity UDP > IAP Catalog**) and Unity IAP (**Window > Unity IAP > IAP Catalog**). Make sure the two IAP catalogs have the same settings.

Note that if you set up IAP items on the UDP portal, the items cannot be synchronized to the IAP catalog of Unity IAP.

For detailed information, refer to [Implementing IAP](#).

Building and pushing your game to the UDP portal

Create a local Android APK build (in the Editor, go to **File > Build Settings > Android > Build**).

For more information, see documentation on [building for Android](#).

To push your game to UDP, you can

- [Upload the APK file of your game to UDP portal directly](#) or
- [Push your game via Cloud Build](#)

For detailed information, refer to [Building your game and pushing it to the UDP portal](#).

Publishing your game to partner stores

Before you publish your game to partner stores, you must add your company information and select your stores.

To publish your game to a store:

1. Go to the [UDP portal](#).
2. Navigate to your Project and choose **Publish**.
3. Select the store and choose the **Target step**.

Target step specifies to which step UDP handles your game. You can choose either of the following steps:

- **Pack**, which indicates the step where UDP packs your game with the SDK from the selected store.
- **Test**, which indicates that the step where UDP packs and submits your game to the test environment of the selected stores.
- **Prod**, which indicates the step where UDP packs and submits your game to the production environment of the selected stores.

4. Publish your game by clicking **Publish**.

Game Information

Publish

Status

Analytics

Publish to Stores

Publish


Select Stores

☒

Stores

Target step

☒



CloudMoolah
CloudMoolah MOO Store is a commercial o...

☒ Pack

☒ Test

☒ Prod

Advanced

For detailed information, refer to [Publishing Your Game from UDP Portal to Stores](#).

Also See

- [Frequently Asked Questions](#)

2. Configuring Unity Distribution Platform

To configure Unity Distribution Platform (UDP):

1. [Set up UDP](#)
2. [Generate a Unity client and test accounts](#)

Setting up UDP

1. Open Unity and log in with a Unity account. If you don't have a Unity account yet, register one in <https://id.unity.com>.
2. Create or open a Project in the Unity Editor.
3. Download the [UDP SDK](#).
4. Double-click the UDP package. The Import Unity Package window appears.
5. Select **Import**. Unity imports the UDP SDK.

To complete the UDP IAP integration, refer to [Implementing UDP IAP](#).

Tip: Unity IAP supports UDP as of version 1.21. You can enable UDP by [setting up Unity IAP](#). To complete the Unity IAP integration, refer to [Implementing Unity IAP for UDP](#).

Generating a Unity client and test accounts

The UDP SDK contains a **GameSettings** file. You can use the file to generate a Unity client and test accounts.

1. Create a **GameSettings** file by selecting **Window > Unity UDP > Game Settings**.

2. View the Game Settings.asset file in the **Inspector** window and generate a Unity client by clicking **Generate Unity Client**.

The screenshot shows the 'GameSettings' window with the following sections:

- Unity Project ID:** A text field containing '87491149-3881-4682-905c-2dee11b61eb6' and a 'Copy to Clipboard' button.
- Link Project to Existed Client:** A section with a 'Client ID' text field and a 'Link' button.
- UDP Client Settings:** A section with five rows: 'Client ID' (None), 'Client Key' (None), 'Client RSA Public Key' (None), 'Client Secret' (None), and 'Callback URL' (empty text field).
- Buttons:** A 'Generate Unity Client' button (highlighted with a red rectangle), an 'Update Client Settings' button, and an 'Update Game' button.
- Game Settings:** A section with three text fields: 'Game Id:', 'Game Title:', and 'Game Revision:'.
- Test Account Settings:** A section with 'Email' and 'Password' text fields and a '+' button.
- Footer:** An 'Edit Game Information on Portal' button.

Property	Function
Client ID	The client identifier
Client Key	Which is used when initializing the UDP SDK
Client RSA Public Key	Which is used to verify the callback notification
Client Secret	A Unity key to signing your request that your game send to the UDP server
Callback URL	Specifies the server that receives the callback notification

3. Implementing IAP

You can implement either UDP IAP or [Unity IAP](#). UDP IAP is a service provided by UDP that lets you sell in-app contents to players. UDP IAP and Unity IAP provide different APIs.

If you are currently using Unity IAP, we recommend you continue with Unity IAP.

This section shows you how to:

- [Implement UDP IAP](#)
- [Implement Unity IAP for UDP](#)
- [Manage IAP on the UDP portal](#)

3.1 Implementing UDP IAP

To use UDP IAP, implement the integration on the client and the server sides. If your game is offline, you only need to implement the client-side integration. If your game is online, you can choose whether to implement the server-side integration.

- [Implementing UDP IAP on the client side](#)

The implementation on the client side includes initializing the UDP SDK and integrating with the in-app purchase flow of UDP.

- [Implementing UDP IAP on the server side](#)

The implementation on the server side lets you query the UDP server about orders, receive callback notifications, and send back the acknowledgements.

3.1.1 Implementing UDP IAP on the client side

To implement UDP IAP, follow these steps:

1. [Configure UDP IAP](#)
2. [Initialize UDP SDK](#)
3. [Query the partner stores' inventory](#)
4. [Purchase a product](#)
5. [Consume a purchase](#)
6. [Validate the client-side integration](#)

Configuring UDP IAP

To configure UDP IAP:

1. Open the IAP Catalog (**Window > Unity UDP > IAP Catalog**).
2. Enter your product information.
 - **Name**, which indicates the name of the IAP item
 - **Product ID**, which indicates the unique ID that is used to identify the IAP item

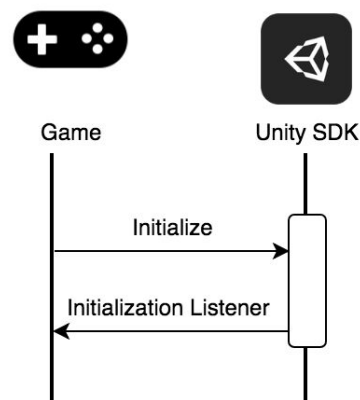
- **Type**, which indicates whether the IAP item is consumable
- **Price**, which indicates the price of the IAP item. You must specify the product price, otherwise players can't purchase products in game stores.
- **Description**, which provides you with a reference

3. Click **Save Product Draft and Add New Product Draft** to create the product and sync with the UDP server. UDP generates the ID when you add the product successfully.

4. To add more products, click **Save Product Draft and Add Product Draft**.

Initializing the UDP SDK

To be able to initialize the UDP SDK, your game provides the Unity client ID and Unity client key to the UDP SDK. The initialization listener lets you know whether the initialization succeeds by returning a success or failure message.



Initializing game integration with UDP

Call the `Initialize` method with `IInitCallback` and `AppInfo` in your game code.

```
StoreService.Initialize(IInitCallback, AppInfo)
```

Note that you don't need to send `AppInfo` if you have configured the `GameSetting` file in the Unity Editor.

The `InitListener` then tells your game whether the initialization has succeeded or failed.

```

public class InitCallback : IInitListener
{
    public void OnInitialized(UserInfo userInfo)
    {
        Debug.Log("Initialization succeeded");
        // You can call the QueryInventory method here
        // to check whether there are purchases that haven't be consumed.
    }

    public void OnInitializeFailed(string message)
    {
        Debug.Log("Initialization failed: " + message);
    }
}

```

Querying the partner store's inventory

To prevent a product from being bought but not delivered, your game needs to query the partner stores' inventory after the initialization is completed. This allows you to, for example, restore non-consumable products when users reinstall your game.

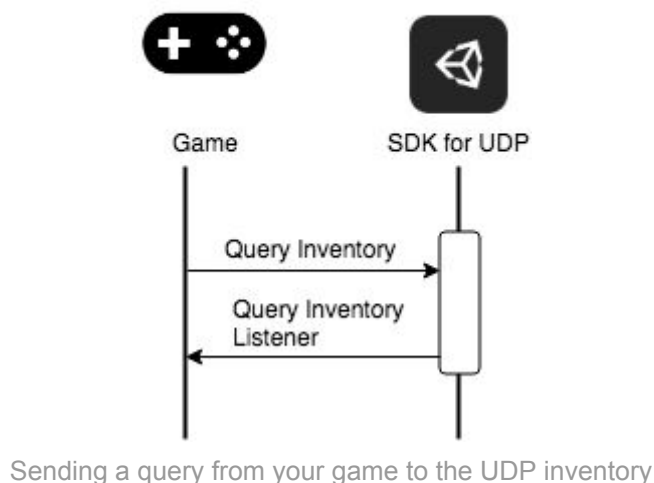
Consumable items

Consumable items provide temporary effects, such as game currency and extra experience points. You can make them available to your players multiple times.

Non-consumable items

Non-consumable items provide permanent effects. Players can only purchase them once.

Your game determines whether there is an unconsumed product by calling the `QueryInventory` method immediately after the initialization succeeds.



You can query for the product details by calling the `QueryInventory` method. If you don't specify `productIds`, you get the information of all products that are purchased but not consumed. If you specify `productIds`, you get the product information for your specified products.

```
PurchaseService.QueryInventory(productIds, callback);  
PurchaseService.QueryInventory(callback);
```

Reminder: Call this method after a successful initialization. This step prevents a product being paid for but not delivered for some reason, such as an application crash.

Implementing listeners for events that are related to the purchase service

`IPurchaseListener` provides the following listeners that tell you the result of all purchase-related events:

- `OnPurchase` - The purchase succeeded.
- `OnPurchaseFailed` - The purchase failed.
- `OnPurchaseRepeated` - Used when a player buys a non-consumable product several times. You can implement this listener when the partner store doesn't support `QueryInventory`.
- `OnPurchaseConsume` - The consumption succeeded.
- `OnPurchaseConsumeFailed` - The consumption failed.
- `OnQueryInventory` - The query succeeded.
- `OnQueryInventoryFailed` - The query failed.

Here is an example:

```

public class PurchaseCallback : IPurchaseListener
{
    public void OnPurchase(PurchaseInfo purchaseInfo)
    {
        // The purchase has succeeded.
        // If the purchased product is consumable, you should consume it here.
        // Otherwise, deliver the product.
    }

    public void OnPurchaseFailed(string message, PurchaseInfo purchaseInfo)
    {
        Debug.Log("Purchase Failed: " + message);
    }

    public void OnPurchaseRepeated(string productCode)
    {
        // Some stores don't support queryInventory.
        // When a user buys a non-consumable product repeatedly, this listener will be
        triggered.
    }

    public void OnPurchaseConsume(PurchaseInfo purchaseInfo)
    {
        // The consumption succeeded.
        // You should deliver the product here.
    }

    public void OnPurchaseConsumeFailed(string message, PurchaseInfo
purchaseInfo)
    {
        // The consumption failed.
    }

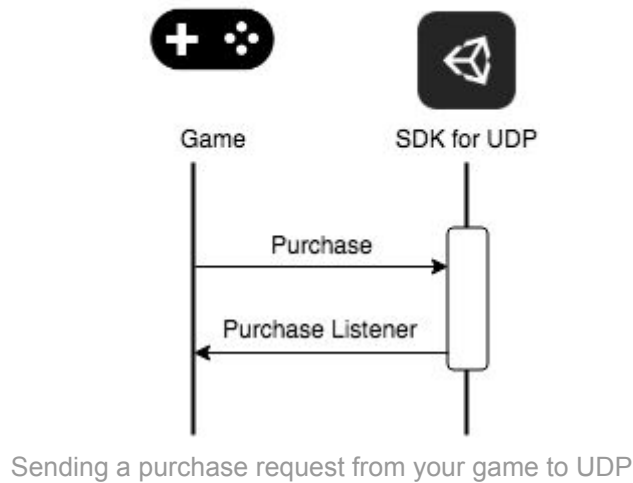
    public void OnQueryInventory(Inventory inventory)
    {
        // Querying inventory succeeded.
    }

    public void OnQueryInventoryFailed(string message)
    {
        // Querying inventory failed.
    }
}

```

Purchasing a product

To start a purchase request from your game, call the Purchase method. The UDP automatically checks the purchase receipt to see whether the purchase is valid.



When you call the Purchase method, provide the:

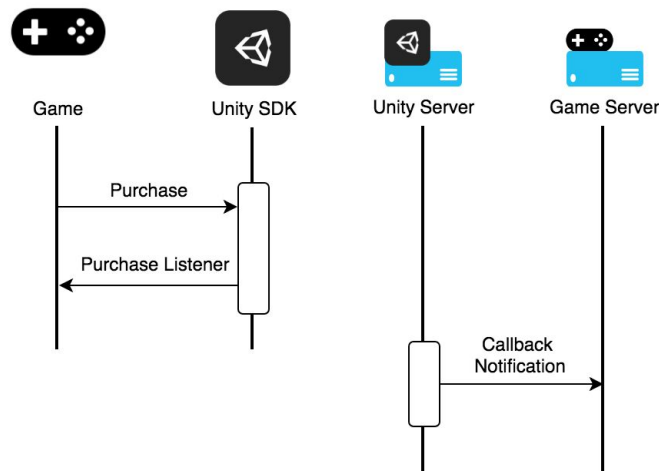
- **productId** - The unique identifier of the product that the player wants to buy.
- **cpOrderId** - A unique order identifier defined by you. If you provide a null for this argument, the UDP SDK generates a UUID automatically.
- **developerPayload** - The information you want to send to the UDP SDK.
- **PurchaseCallback** - An instance of the **PurchaseCallback** class

For example

```
PurchaseService.Purchase("productId", "cpOrderId", "developerPayload",  
PurchaseCallback);
```

The UDP returns information to your game after the purchase is complete.

If your game is an online game, you can verify the purchase on your game server by receiving a callback notification. UDP sends the callback notification to the callback URL that you have specified either in the Unity Editor or on the UDP portal.

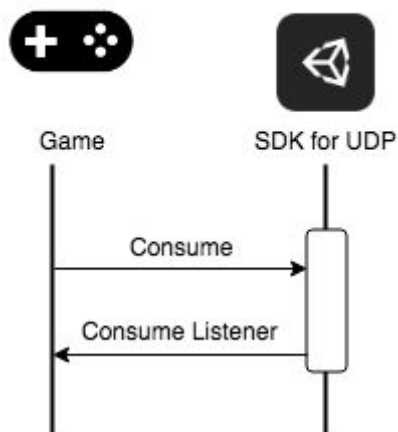


Consuming a purchase

When users purchase a consumable product, they cannot repurchase that product until UDP have consumed the product. So you can use the consumption to make sure the purchased product is successfully delivered.

Note: This step is only necessary when the product is consumable.

To consume a product, your game needs to send a **Consume** request to the UDP SDK. We recommend that your game redelivers a product after it has been initially consumed, or the product may be delivered repeatedly.



Sending a consume request from your game to UDP

```
PurchaseService.ConsumePurchase(PurchaseInfo, IPurchaseListener);
```

PurchaseInfo is returned by OnPurchase.

Validating the client-side integration

UDP performs client-side validations automatically. When partner stores return the payload and signature after a successful purchase, the UDP SDK validates the signature. If the validation fails, the purchase fails accordingly.

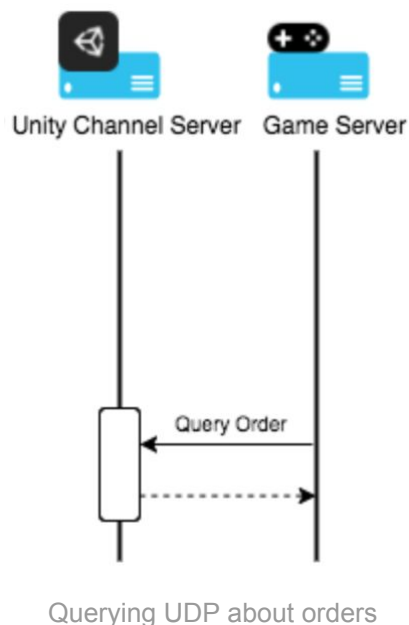
3.1.2 Implementing UDP IAP on the server side

The server-side integration consists of the following steps:

- [Query orders](#)
- [Receive callback notifications](#)
- [Send acknowledgeSending acknowledgements \(optional\)](#)

Querying Orders

Your game can query UDP about orders by calling an HTTP GET request.



GET

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

Parameters in the request

Attribute name	Format	Required/Optional	Description	Example
cpOrderId	String	Required	The order ID assigned by your game, or Unity if the game does not generate it.	66mea52wne
clientId	String	Required	The client ID assigned by Unity after onboarding.	Q4AnJDW2-rxLAPujqrk1zQ
orderQueryToken	String	Required	The order query token returned by the client SDK when finishing a purchase.	ebaseakc=
sign	String	Required	An md5 hash value containing the cpOrderId, clientId, orderQueryToken and Unity Client Secret .	7277da780d1102864ee5818c2730c74e

Parameters in the response

Attribute name	Format	Required/optional	Description	Example
id	String	Required	The order ID assigned by Unity.	274877943826
clientId	String	Required	The clientId that is returned by Unity after the game has created a client in the Unity IAP.	Q4AnJDW2-rxLAPujqrk1zQ
cpOrderId	String	Required	The order ID assigned by your game, or Unity if the game does not	66mea52wne

			generate it.	
uid	String	Required	A user identifier assigned by Unity.	15400813498
status	String	Required	Indicates the status of the order.	SUCCESS, FAILED, UNCONFIRMED
productId	String	Required	The product id associated with the order	Product_1
payFee	Integer	Required	The payment amount of the order	1
quantity	Integer	Required	Indicates the quantity of the product.	1
notified	Boolean	Required	Indicates whether Unity has notified the Game server with the order information.	No
currency	ISO 4217	Required	The currency used to purchase the product	CNY
country	ISO 3166-2	Required	The country or geographic region in which the user is located	CN
paidTime	ISO8601 yyyy-MM-ddThh:mm:ssXXX , UTC timezone	Required	Specifies the time when the order is paid.	2017-03-08T06:43:20Z
createTime	ISO8601 yyyy-MM-ddThh:mm:ssXXX , UTC timezone	Required	Specifies the time when the order is created.	2017-03-08T06:43:20Z
updateTime	ISO8601 yyyy-MM-	Required	Specifies the time when the order is	2017-03-08T06:43:20Z

	ddThh:mm:ssXXX , UTC timezone		updated.	
createdBy	String	Required	The name of the user that created the purchase order. This field is only used for technical support.	274877943822
updatedBy	String	Required	The name of the user that updated the purchase order. This field is only used for technical support.	0
rev	String	Required	The revision of the order (only for update)	0
extension	Json String	Optional	the developer payload which is used to contain reference information for developers	{"abc" : "123"}

Here is an example request from your game server to the UDP server and response from the UDP server back to your game server.

Request:

GET

/v1/order-attempts/query?cpOrderId=<66mea52wne>&clientId=<Q4AnJDW2-rxLAPujqrk1zQ>&orderQueryToken=<eyJJaGFubmVsVHlwZSI6ICJGQUtFQ0hMliwglmNoYW5uZWxVaWQiOiAiRkFLRUNlTF9oeGQwNSJ9>&sign=<debd0018911d263e23dfb729207b905c>

Response:

```
{
  "createdBy" : "0",
```



```

    "updatedBy" : "0",
    "createTime" : "2017-03-08T06:43:20Z",
    "updateTime" : "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"
}

```

Receiving callback notifications

After a purchase succeeds, if you have specified a callback URL, the UDP server notifies the game server with the payment result. Implement an HTTP GET request and accept the following query parameters:

Attribute Name	Format	Required/Optional	Description
payload	Json String	Required	The contents of the purchase order. See more in Json payload .
signature	String	Required	The RSA signature of the payload
certificate	Base64 encoded String	Required	The X.509 certificate that contains the RSA public key for the signature.

Using the certificate

You can verify the certificate using the Unity Client RSA Public Key. If the certificate passes verification, extract the RSA public key from the certificate and use this key to verify the signature.

The signature is generated by encrypting the payload with the by RSA-SHA1 algorithm.

Here is a code sample that shows how to verify the certificate in Java:

```
import java.util.Base64;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.security.spec.X509EncodedKeySpec;
import java.security.KeyFactory;
import java.security.PublicKey;
import java.security.Signature;

void verify (String publicKey, String certificate, String payload, String signature) {
    // verify certificate
    Base64.Decoder decoder = Base64.getDecoder();
    byte[] certBytes = decoder.decode(certificate.getBytes("UTF-8"));
    CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
    X509Certificate cert = (X509Certificate)certFactory.generateCertificate(new
ByteArrayInputStream(certBytes));
    byte[] keyBytes = decoder.decode(publicKey.getBytes("UTF-8"));
    X509EncodedKeySpec keySpec = new X509EncodedKeySpec(keyBytes);
    KeyFactory fact = KeyFactory.getInstance("RSA");
    PublicKey pubKey = fact.generatePublic(keySpec);
    if (cert.notAfter.before(new Date())) {
        // expired certificate, verification failed
    }
    try {
        cert.verify(pubKey);
    } catch (Exception e) {
        // verification failed
    }

    // verify signature
    Signature signature = Signature.getInstance("SHA1WithRSA");
    signature.initVerify(cert.getPublicKey());
    signature.update(payload.getBytes("UTF-8"));
    boolean verified =
signature.verify(decoder.decode(signature.getBytes("UTF-8")));
    if (!verified) {
        // verification failed
    }
}
```

Here is the content of a JSON payload:

Attribute Name	Format	Required/Optional	Description	Example
cpOrderId	String	Required	The unique order identifier assigned by your game.	66mea52wne
status	String	Required	Indicates the status of the order.	SUCCESS
amount	String	Required	Specifies the amount of money that the order cost.	1
productId	String	Required	Specifies the unique identifiers of the products that belong to the order.	Product_1
payTime	ISO8601 yyyy-MM-ddThh:mm:ssZ, UTC timezone	Required	The time when the order was paid.	2017-03-08T06:43:20Z
country	ISO 3166-2	Required	The country where the order was paid.	US
currency	ISO 4217 or cryptocurrency type	Required	The currency of the country where the order was placed.	USD
cryptocurrency	Boolean	Required	Indicates whether the currency is cryptocurrency.	FALSE
quantity	Integer	Required	The number of products in the order.	1
clientId	String	Required	The unique client identifier that is returned after your game generates a client in Unity IAP.	P-wU0n9pbyQ1ulks4kHX_Q
extension	String	Optional	The developer payload which is used to contain reference information for developers.	"{"abc" : "123"}"

Sending acknowledgements (optional)

After your game server receives a callback notification, it needs to send an acknowledgement to the Unity server. Implement an HTTP GET request and use the query parameters in the following table.

GET

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

Parameters in the request

Attribute Name	Format	Required/Optional	Description	Example
cpOrderId	String	Required	The order ID assigned by your game, or Unity if the game does not generate it.	66mea52wne
clientId	String	Required	The client ID assigned by Unity after your game generates a client in UDP.	Q4AnJDW2-rxLAPujqrk1zQ
orderQueryToken	String	Required	The order query token returned by the client SDK when finishing the purchase.	ebaseakc=
sign	String	Required	The md5 hash with cpOrderId, clientId, orderQueryToken and Unity Client Secret .	7277da780d1102864ee5818c2730c74e

Response

The UDP server sends a status code back to your game server.

3.2 Implementing Unity IAP for UDP

If you want to use Unity IAP instead of UDP IAP, [set up Unity IAP](#).

Note: If you choose to implement Unity IAP instead of UDP IAP, you just need to implement Unity IAP APIs.

When the Unity IAP is enabled, take the following steps to set up Unity IAP for UDP.

Step 1: In the Unity Editor, to choose UDP as the target platform for the game to build, select **Unity IAP > Android > Target Unity Distribution Platform (UDP)**.

Step 2: To configure the IAP catalog, selecting **Window > Unity IAP > IAP Catalog**.

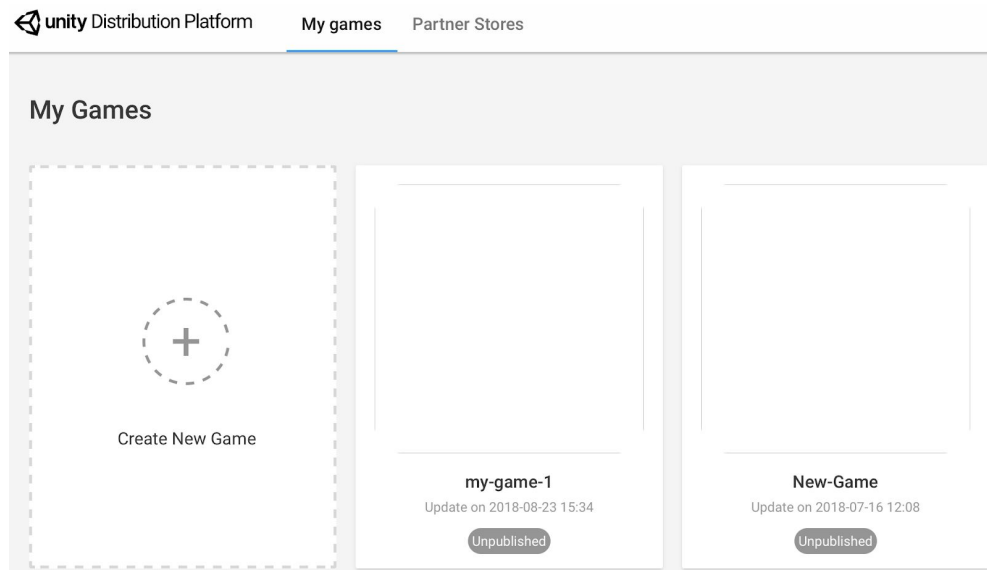
Step 3: Make sure that Unity IAP and UDP IAP have the same configuration.

You can find **IAP Catalog** under both **Unity IAP** and **Unity UDP**. Make sure these IAP catalogs have the same settings.

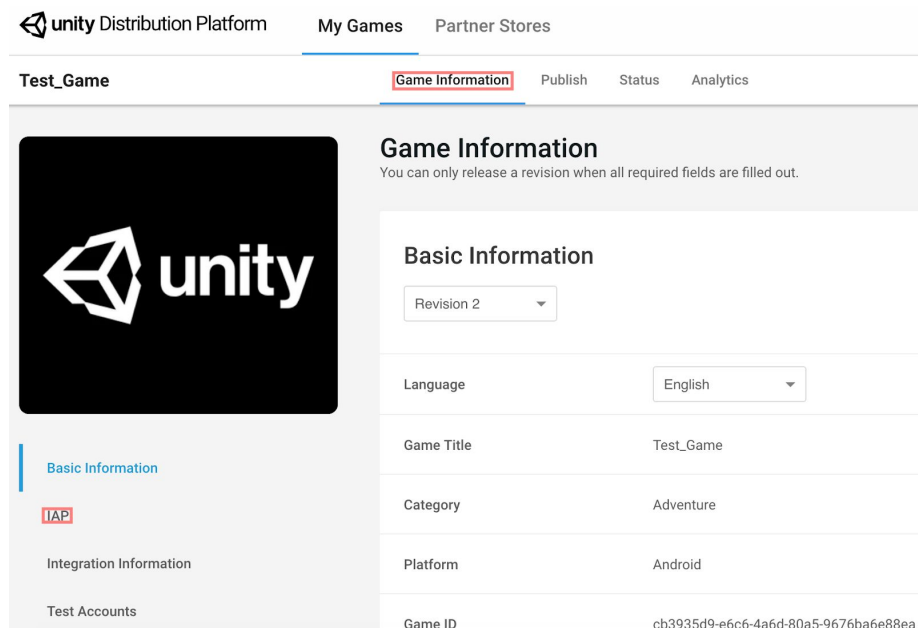
3.3 Managing IAP on the UDP portal

You can manage IAP on the [UDP portal](#).

1. Login in to the UDP portal and select your game.



2. Go to IAP by choosing **Game Information > IAP**. If you have created IAP items in the Unity Editor, you can see these items here:



3. You can add, edit, or delete IAP items.

4. Building your game and pushing it to the UDP portal

Building your game

Create a local Android APK build in the Editor (**File > Build Settings > Android > Build**). For more information, refer to [Getting started with Android development](#).

Uploading your game file to the UDP portal directly

You can create a local build in the Unity Editor and upload it to the UDP portal directly.

1. On the [UDP portal](#), navigate to your Project.
2. Select **GAME INFO > Basic**.
3. Click **APK Files** to upload the APK file of your game.

Pushing the Project to the UDP portal via Cloud Build

In the Editor, enable Cloud Build through the Unity Services window (see documentation on [Cloud Build implementation](#)).

If you use Unity Team Advanced, you can generate builds automatically (see documentation on [Automated Build Generation](#)). Otherwise, upload the build either from the Editor or the Unity Cloud Build Developer Dashboard.

Upload the build

Upload your build to the build history for your Project, using one of the following methods:

From the Editor:

1. In the **Cloud Build Services** window, if you haven't uploaded any build before, select **Upload Build**. Otherwise, select **Manage Build Targets > Add new build target**.
2. In the **TARGET SETUP** window, set the **PLATFORM** field to your intended platform and enter a useful **TARGET LABEL**. Then select **Next: Save**.
3. Select **Start Cloud Build**, then select the target build you just created.

From the [Unity Cloud Build Developer Dashboard](#):

1. Navigate to your Project's **Cloud Build > History**.
2. Select **Upload**, then select the APK file you built from the Editor.



Uploading a Cloud Build via the Developer Dashboard.

Push the build to UDP

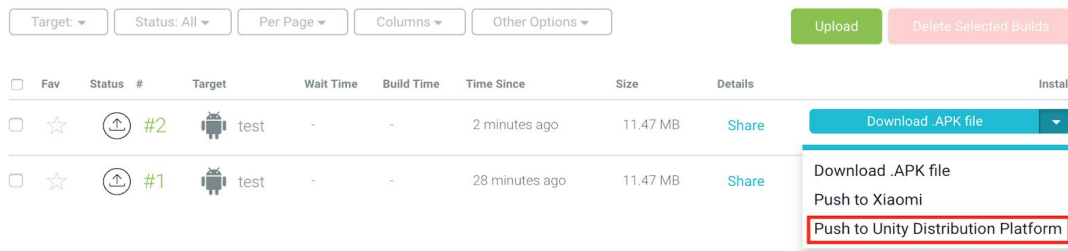
Push the hosted build to the UDP portal, using one of these methods:

From the Editor:

In the Cloud Build Services window, locate the desired build from the build history timeline and select **Push to Unity Distribution Platform**. Verify that you want to push, and that the action completes.

From the [Unity Cloud Build Developer Dashboard](#):

1. Navigate to your Project's **Cloud Build History**.
2. Click **Download .APK** file, then select **Push to Unity Distribution Platform** from the drop-down menu.



Pushing a hosted build to UDP via the Unity Developer Dashboard.

Note: Only the owner of the organization for your Project can deploy to UDP.

5. Managing your game on the UDP portal

UDP portal is a web-based platform where you can manage your game, as follows:

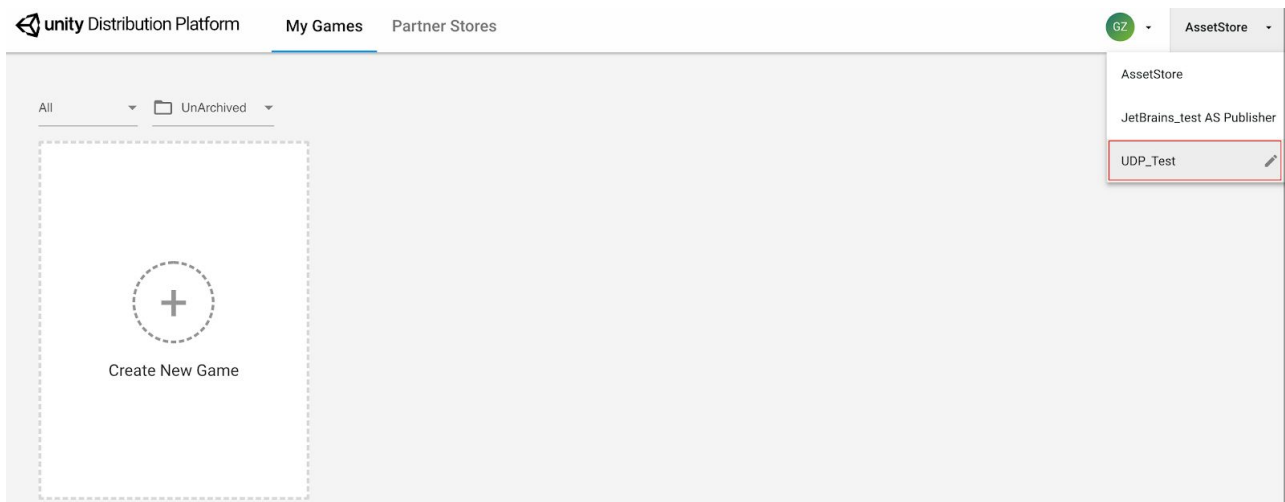
- [Create and edit your profile](#)
- [Create and edit your games](#)

You can go to the UDP portal via the URL <https://connect.unity.com/udp>. Alternatively, you can view the `GameSetting.asset` file in the `Inspector` window in Unity Editor, and then go to the UDP portal by clicking **Edit Game Information on Portal**.

5.1 Creating and editing your profile

Before publishing your game to the UDP portal, you need to create a company profile for the game. UDP sends the the company profile to the partner stores as the developers' information.

1. Choose your organization and click on the pencil icon to edit the profile.




Choosing your organization

2. Add the company information by populating the following fields:

- **Company Name**, which specifies the name of your company
- **Location**, which specifies the place where your company is located in
- **Company Size**, which specifies the size of your company
- **Official Website**, which specifies the official website of your company
- **Support Email Address**, which specifies the support email that you provide to players
- **About**, which contains more information for the company

Add Company Information



Company Name

Location

Company Size

Official Website

Support Email Address

About

CANCEL

CREATE

Populating the company information

5.2 Creating and editing your games on the UDP portal

If you haven't created a game in the Editor, the UDP portal allows you to create games on the UDP portal and update the [project ID](#) generated in the Editor. When you [generate a Unity client in the Unity Editor](#), the game is synchronized on the UDP portal automatically.

Warning: If you have created your game in the Editor, don't create the same game on the UDP portal. Navigate to the game that you have created in the Editor and edit the game information. You only need to create a game in the Editor or on the UDP portal, not both.

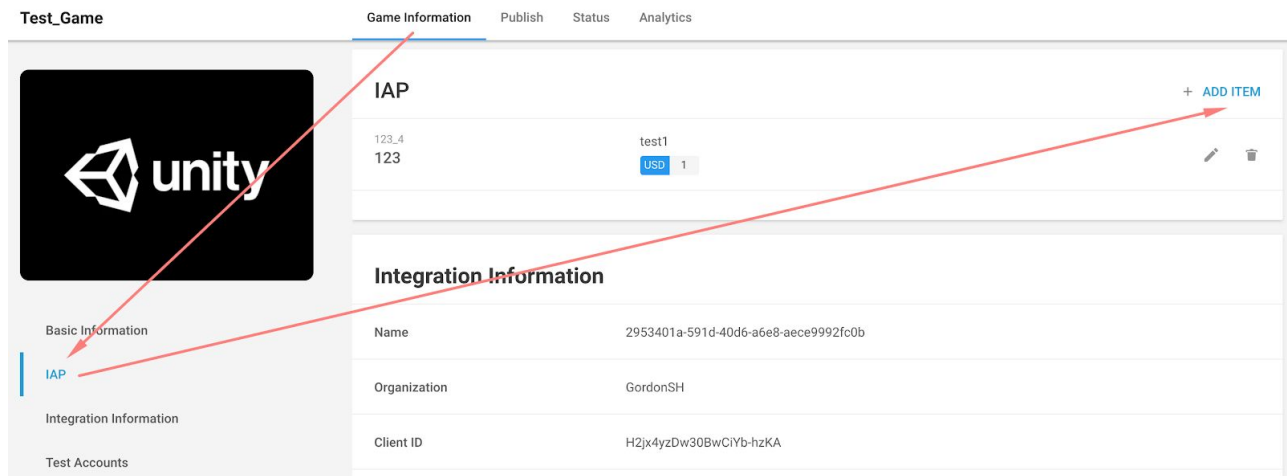
To create a game on the UDP portal:

1. Click **Create New Game** and enter a name for it
2. Click **Basic Information** and populate the following fields:
 - **Language**, which specifies the language for the game. Currently we only support Chinese and English.
 - **Game Title**, which specifies the title of the game
 - **Category**, which indicates the category that the game belongs to. You can choose among **Action, Adventure, Arcade, Board, Card, Casino, Casual, Educational, Music, Puzzle, Racing, Role Playing, Simulation, Sports, Strategy, Trivia, and Word**.
 - **Platform**, which specifies the platform that the game is published to. Currently the platform is Android by default.
 - **Game ID**, the unique identifier of the game.
 - **Game Icon**
 - **Description**, which is an introduction of the game to players
 - **Main Image**
 - **Feature Image**, which allows you to upload feature images or videos of your game
3. Click **Unity Project ID** and enter the project ID.
4. Upload the APK files.

5.3 Creating In-App Purchasable items

You can create In-App Purchasable (IAP) items both in the Unity Editor and on the UDP portal. To create IAP items in the Unity Editor, refer to [Configuring UDP IAP](#). To create IAP items on the UDP portal:

1. Select your game on the UDP portal.
2. Select **Game Information > IAP > ADD ITEM**.



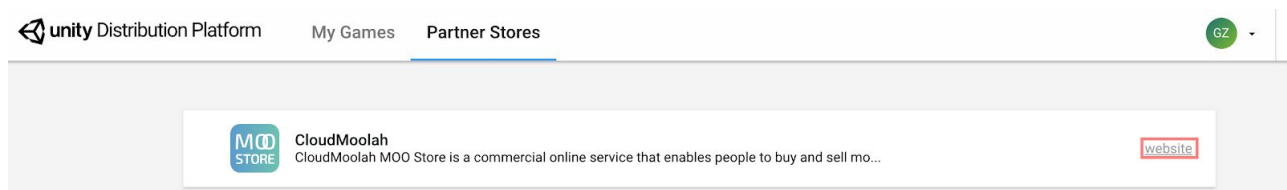
Adding IAP items

3. Specify your product information:

- Product ID, which indicates the unique identifier for the IAP item.
- Product Name, which is the name of the IAP item.
- Price, which is the price of the IAP item in the appropriate currency.
- Description, which provides you with a reference.

6. Publishing your game from the UDP Portal to partner stores

With the UDP portal, you can publish your game to a partner store. If you have not registered an account in the partner store, register one by clicking **website**.



Currently the supported stores include Google, Amazon, Samsung, CloudMoolah, and Xiaomi.

1. Select your game on the [UDP Portal](#).

2. In the **Publish** panel, select the store you want to publish to, enter the package name, and save the store.

3. Click on **Advanced** to configure more settings.



Enter the following package properties depending on which store you are configuring:






Property	Function
Target SDK	The version of the SDK for the store that you publish your game to
Available Step	Specifies to which step UDP handles your game. You can choose either of the following steps: <ul style="list-style-type: none">• Pack, which indicates the step where UDP packs your game with the SDK from the selected store.• Test, which indicates that the step where UDP packs and submits your game to the test environment of the selected stores.• Prod, which indicates the step where UDP packs and submits your game to the production environment of the selected stores.
Package Name	The name of the package
Price	The price of the game

3. Publish your game to the stores by selecting **Publish**.

Publish to Stores

Publish

Select Stores

<input checked="" type="checkbox"/>		Amazon amazon	<input checked="" type="radio"/> Pack	Advanced
<input type="checkbox"/>		Samsung Download the apps designed for your Gala...	<input checked="" type="radio"/> Pack	
<input type="checkbox"/>		CloudMoolah CloudMoolah MOO Store is a commercial ...	<input checked="" type="radio"/> Pack <input type="radio"/> Test <input type="radio"/> Prod	
<input type="checkbox"/>		Xiaomi Xiaomi Market is an app store, alternative t...	<input checked="" type="radio"/> Pack	
<input type="checkbox"/>		Google A digital distribution service operated and ...	<input checked="" type="radio"/> Pack	

4. In the **Status** panel, you can see the statuses of your game.

- **Packed**, which indicates that your game has been packed with the SDK from the selected stores.
- **Synced**, which indicates that your game has been packed and synced to the test environment of the selected stores.
- **Promoted**, which indicates that your game has been packed and synced to the productive environment of the selected stores.
- **Pending**, which indicates the packing is ongoing
- **Failed**, which indicates the packing has failed



Status

Store ▼ Target ▼ Status ▼

0
Pack

0
Beta

0
Published

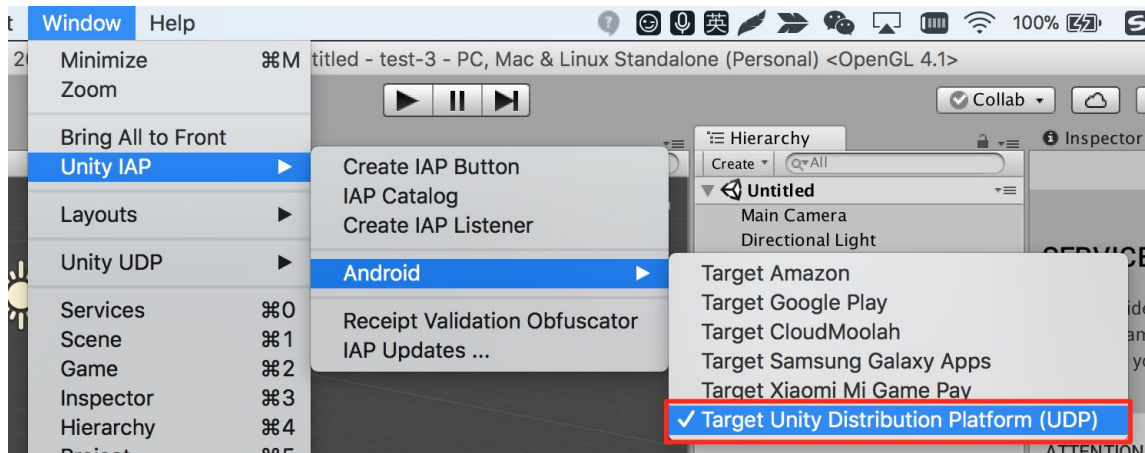
Store	Publish Time	Revision	Target	Status	Action
 AMAZON	2018-09-05 11:23	Revision 1	Pack	 Pack Failed	-

7. Frequently Asked Questions

- Why is an exception (**ClassNotFoundException**) raised when I initialize the UDP SDK?

Cause

You may encounter the error because you didn't target UDP when using Unity IAP.



Solution

In the Unity Editor, select **Window > Unity IAP > Android > Target Unity Distribution Platform (UDP)**.

For detailed information, refer to [Implementing Unity IAP](#).

- What is the difference between consumable and non-consumable items?

UDP IAP supports two kinds of items: consumable items and non-consumable items. Consumable items can be made available to your players multiple times. They provide temporary effects, such as game currency and extra experience points.

Non-consumable items can be purchased only once and provide permanent effects.

- Is there any problem if I try the initialization only once before purchasing?

No, there won't be any problem because Initialization (`StoreService.Initialize()`) should only be called once.