Dmarket

Make Virtual Assets Real

DMarket Widget SDK Documentation

DMarket and Unity Integration

Table of Contents

Introduction

The DMarket SDK for Unity complements Unity Technologies' cross-platform support, providing a pure-Unity write-once, run-everywhere experience across the key gaming platforms. By maintaining a single codebase, it’s easy to deploy integrated in-game trading experiences to the players, regardless of platform. The DMarket SDK for Unity provides a comprehensive collection of DMarket's features, giving players of Unity games the ability to buy, sell and trade in-game items. Dmarket SDK keeps the code clean, providing Unity games with a unified and consistent model for in-game items trading.

DMarket Widget Guide

Step 1. Game Developer Registration

To initiate integration, game developer company should get in touch with DMarket through an email integration@dmarket.com to register one or more games. DMarket representatives will set everything up and register a game within a brief timeline.

Step 2. Sending List of In-game Item Classes

After successful confirmation of a game registration you should send a list of virtual item classes in *json* format. For now, virtual items are added manually by DMarket. You can find an example below:

```
{
    "id": "100285f4-1c8b-11e8-accf-0ed5f89f718b", // unique class id
    "meta": {
      "title": "Spaceship",
      "images": [
        {
          "image": "", // specify image name from archive here!
          "type": ""
        }
      ],
      "category": "Common",
      "description": ""
    }
  }
```

NOTE:

- ❏ **`id`** parameter is **required,** as well as **`title`**
- ❏ All other parameters in **`meta`** are optional. You can specify category and description of the asset if needed (i.e., category *Common,* description *Common blue spaceship*), and you can add your parameters if needed, i.e., *Rarity.*

Also you need to share item images archive with DMarket. To do that, save all item images in *.png* format, archive them and send the archive via email to gameintegration@dmarket.com along with *json* file.

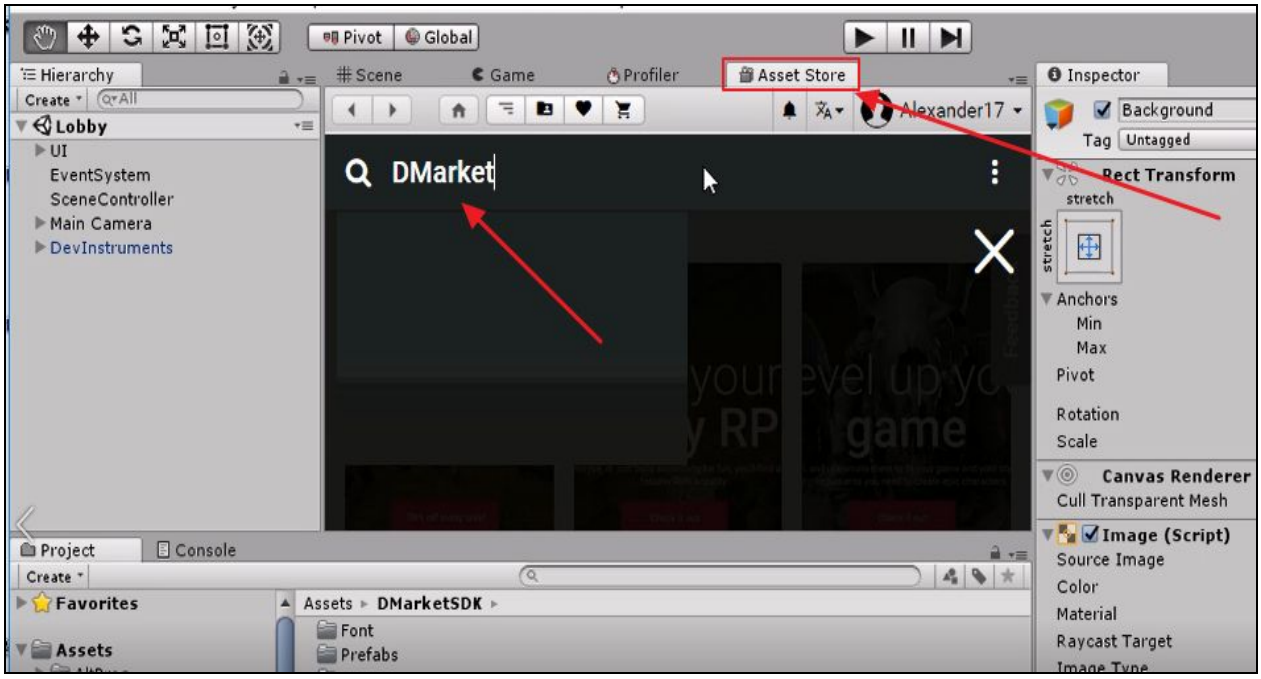
Please synchronise namings of an item in JSON and in archive. Be sure that your item's `image` parameter in JSON has the same name as a corresponding image from an archive.

Step 3. Game Token Receiving

After item list was successfully submitted, you receive **GameToken** from DMarket via email.

Step 4. Getting DMarket SDK from Asset Store

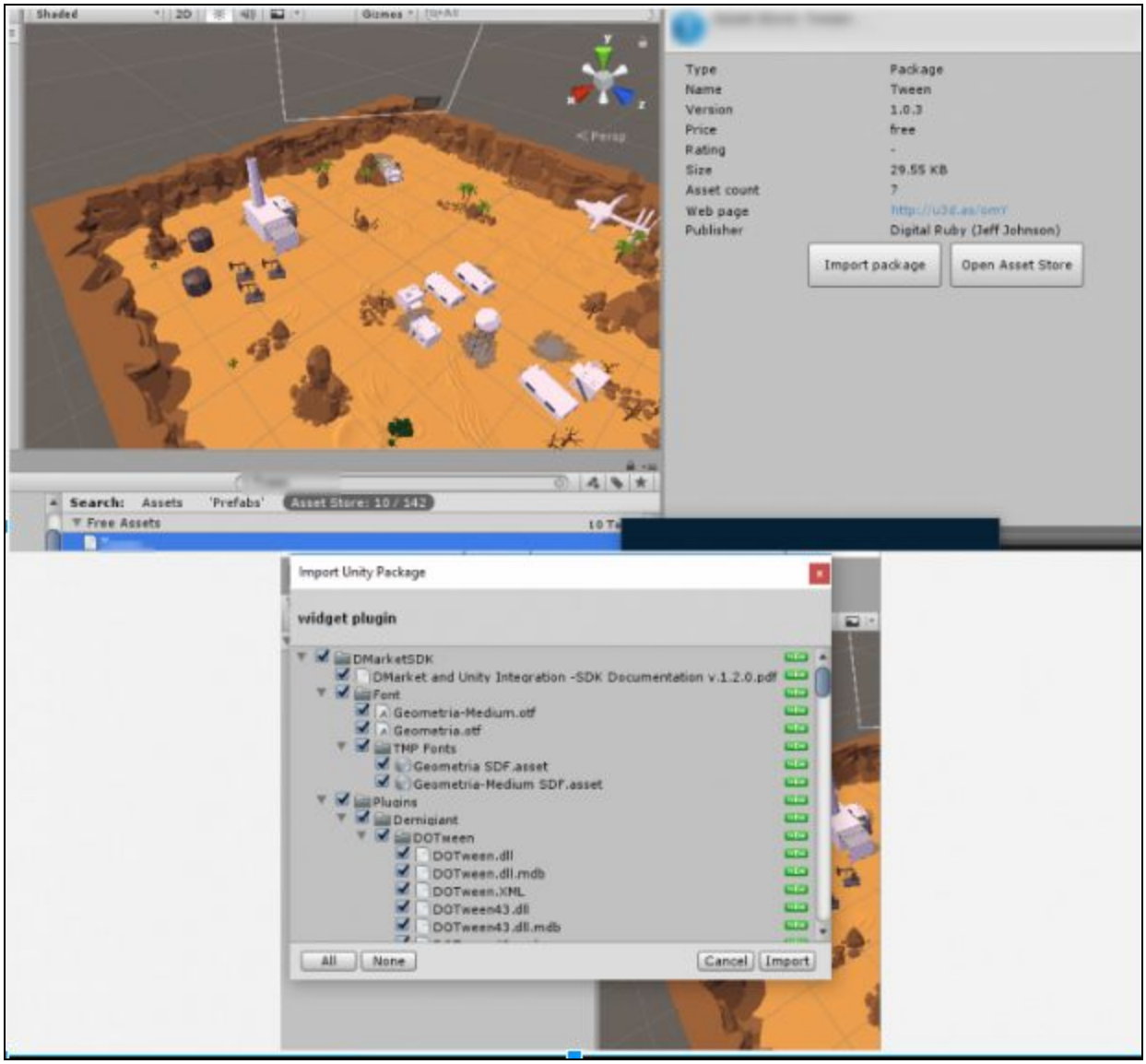
After you receive **GameToken** from DMarket via email, you can download **DMarket SDK** from Unity Asset Store. Enter Asset Store in your Unity Editor, and then enter "DMarket SDK" to the search input field. There will be suggestions in dropdown list, pick correct one by clicking on it.



After you find “DMarket SDK” you can download it. Scroll down a little bit and click **Download** button.

After download is completed, click **Import.**

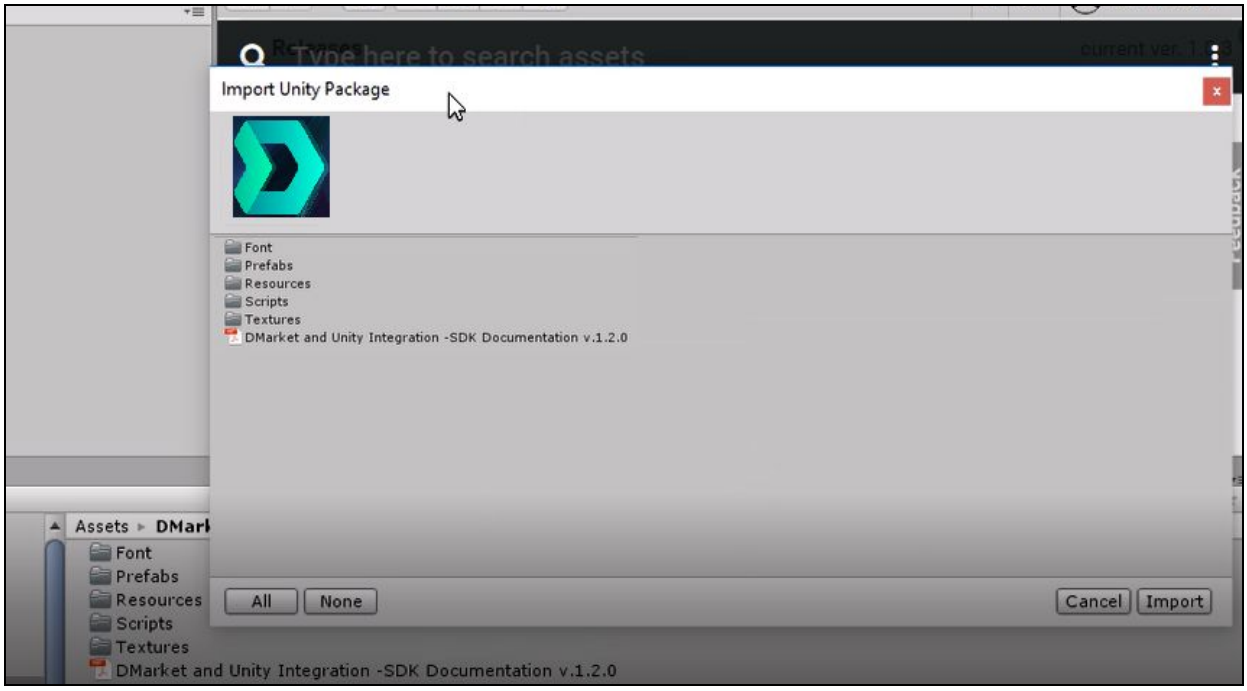
After successful import, start implementation of the package. You can pick folders that you need or don’t need by ticking checkboxes. By default, all folders and items are ticked.



Step 5. Importing DMarket SDK to the Project

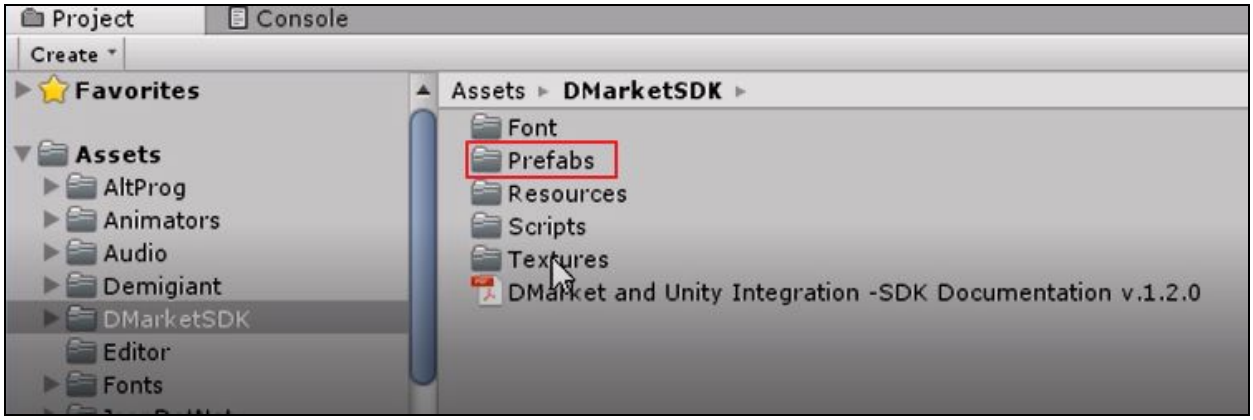
After a first stage of importing you see a modal window with available folders.

Click **Import** in a bottom right corner to finalize importing.

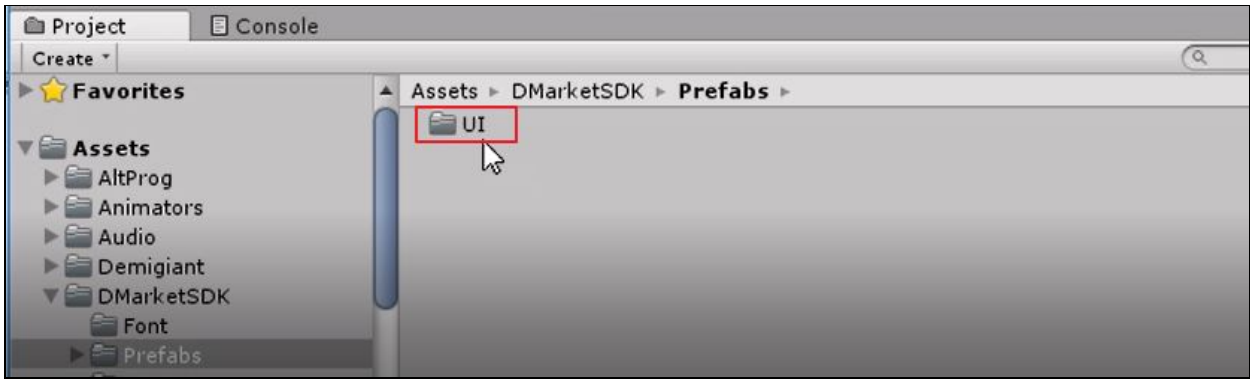


Step 6. Adding DMarket Widget to a Scene

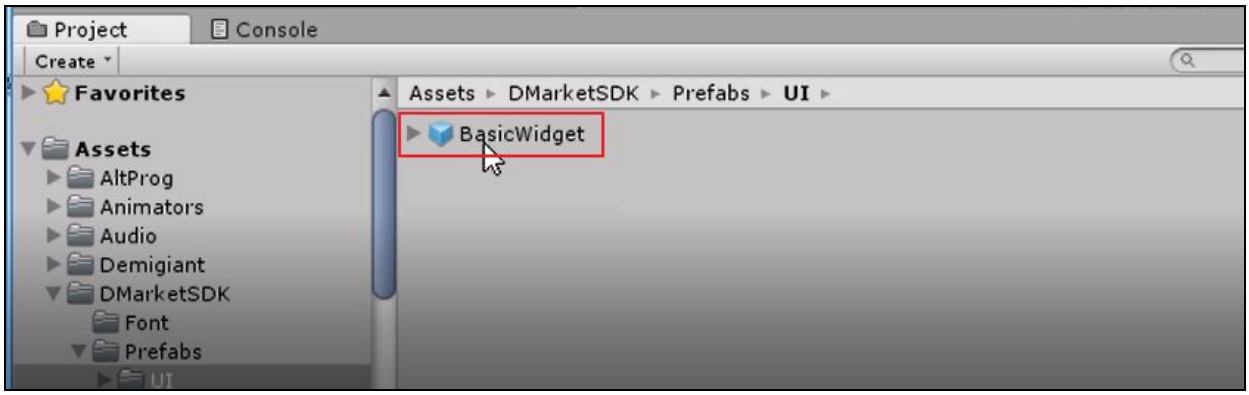
After you've imported the package, open it and then open **Prefabs** folder.



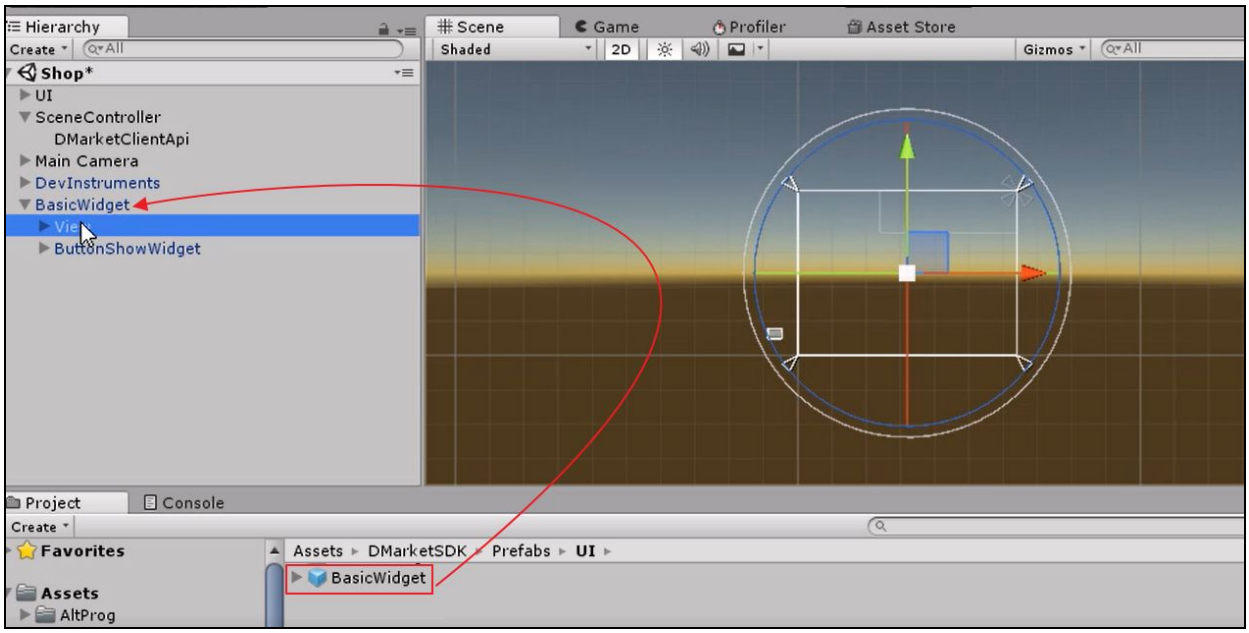
In **Prefabs** folder open **UI** folder.



In **UI** folder, find **BasicWidget** item.



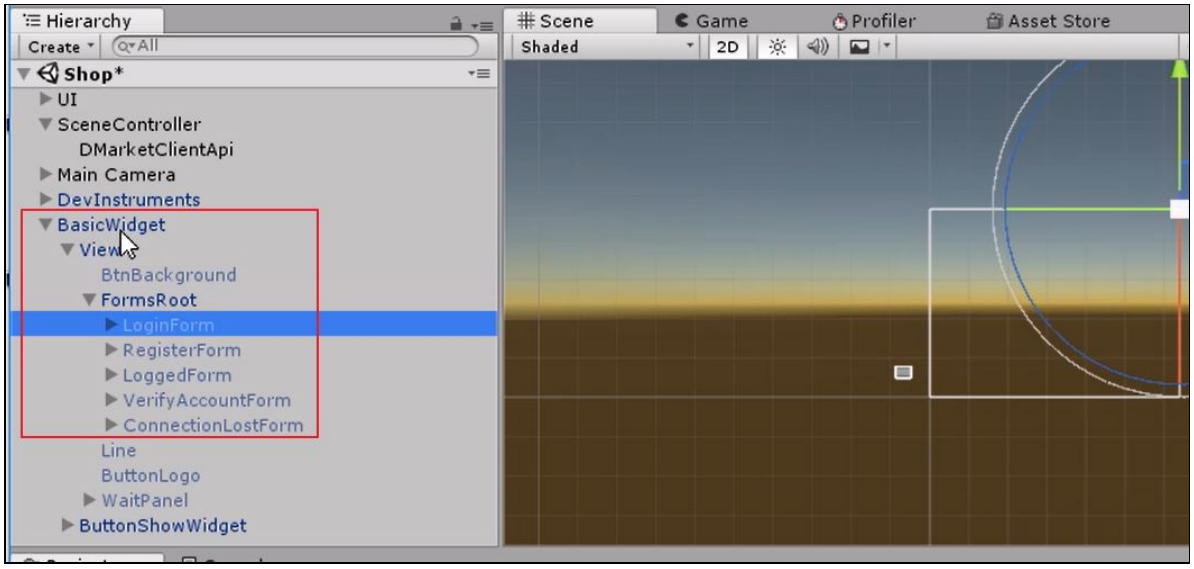
Open your project and move **BasicWidget** item to a specific *Scene.*



Add class name and class type `[SerializeField] public Widget BasicWidget;`

You can also move **BasicWidget** to **Resources** folder and use `Resource.Load(` *yourPath*`)` and then `GameObject.Instance.`

Step 7. Checking Available Functionality Elements

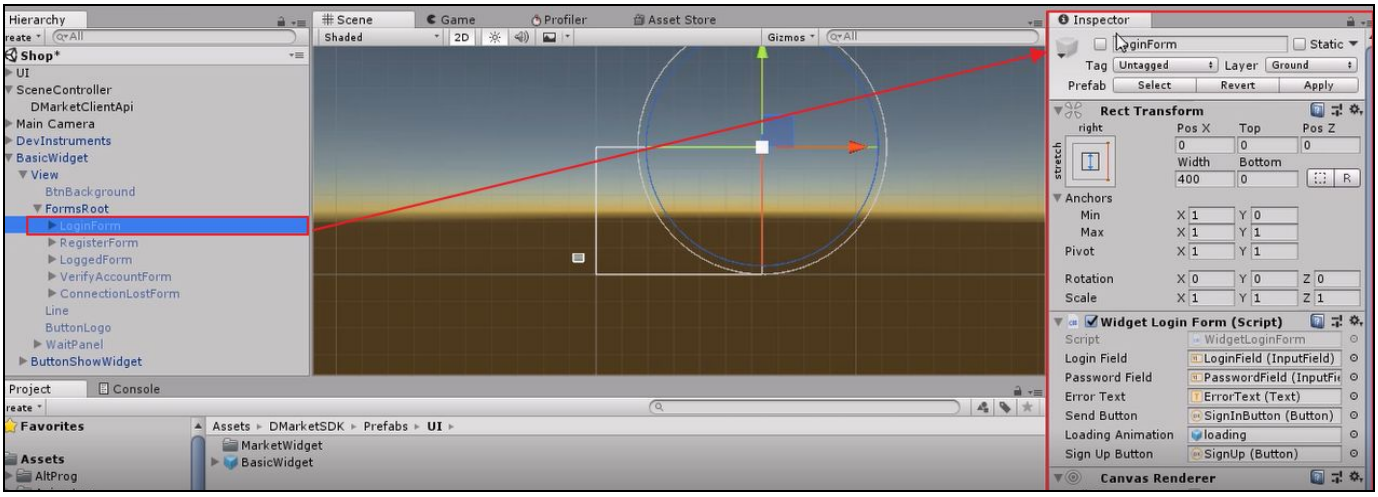


Click **BasicWidget** to open elements tree. In a **View** folder you can find **FormsRoot** folder which contains list of available functions and elements, such as:

- *LoginForm*
- *RegisterForm*
- *LoggedForm*
- *VerifyAccountForm*
- *ConnectionLostForm*

By click on each of these items you can check its properties via Unity's **Inspector,** which appears in a right column. For example, let's check *LoginForm* properties:

LoginForm properties



As you can see, *WidgetLoginForm* section contains all elements that will be available for a player on **Login** screen, such as login and password input fields, error text in case player enters wrong credentials, ***Sign In*** and ***Sign Up*** buttons. You can access *RegisterForm, LoggedForm, VerifyAccountForm* or *ConnectionLostForm* in a same way via **Inspector.**

Step 8. Setting Up DMarket SDK API Clients

These API clients have different purposes. ServerAPI communicates with data on your game's server side, ClientAPI communicates with data on your game's client side. Please take into account that both of the API clients have the option to switch between development (sandbox) environment and production environment. In a sandbox environment all calls to API are logged. URLs for sandbox and production environment were sent along with SDK package.

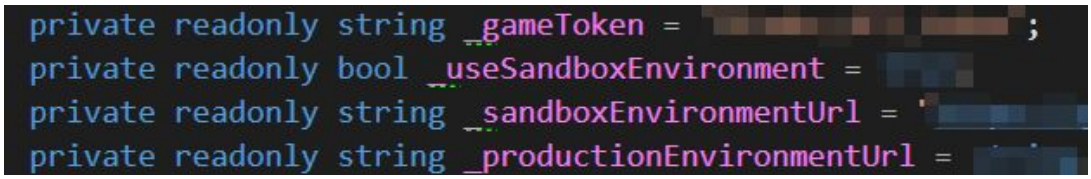
To set up clients, follow these steps:

1. Create a new Unity project.
2. Import **DMarket Unity SDK** to this project.
3. Open SDK's ***Inspector*** and check DMarket API Clients. There are two API clients in it:
   - 3.1. Server API: *DMarketSDK.IntegrationAPI.ServerApi*
   - 3.2. Client API: *DMarketSDK.IntegrationAPI.ClientApi*
     - *3.2.1.* *DMarketSDK.IntegrationAPI.ServerApi* is used for setting up **Game Token** you've received from DMarket.
4. Create a new empty *GameObject* in *Scene*.
   - 4.1. If *Scene* is for game **client**, then
     - 4.1.1. Add a component script *DMarketSDK.IntegrationAPI.ClientApi*

4.2. If *Scene* is for game **server**, then

4.2.1. Add a component script *DMarketSDK.IntegrationAPI.ServerApi*

If you decided to use *DMarketSDK.IntegrationAPI.ServerApi,* you must specify **Game Token**; to do that, please add a **Game Token** string to BaseApi.cs:



Also you need add your own URLs for environment.

Assemblies (namespaces) for the APIs:

- `namespace DMarketSDK.Widget`
- `namespace DMarketSDK.IntegrationAPI`

Step 9. Initializing DMarket Widget

Whether you decide to set up everything on production or sandbox environment, you should initialize DMarket Widget. To do that, you need to receive *BasicAccessToken* from ServerAPI. To start DMarket widget initialization use the following call:

```
void Init(ClientApi clientApi, string basicAccessToken, string refreshToken);
```

Step 10. Open DMarket Widget

After initialization you can open DMarket Widget through method

```
void Widget.Open();
```

Step 11. Close DMarket Widget

In case you want to apply widget force closing for player, you can apply method

```
void Widget.Close();.
```

Step 12. BasicAccessToken Receive and Setup

BasicAccessToken is the authorization key for the specific game and the specific user. To get it, you should make a call to Server API:

```
ServerApi.GetBasicAccessToken (gameUserId) ;
```

`gameUserId` is the unique identifier for the user within the game. It can be either user ID, or session ID, or client ID.

BasicAccessToken is taken from Server API and transferred to Client API for initialization widget (see ***Step 9***). After initialization, player can open and login through widget. After successful login you receive the following event*:*

```
public event Action <LoginEventData> LoginEvent;
```

Within `LoginEventData` you receive `MarketAccessToken`

```
public class LoginEventData
{
    public readonly string MarketAccessToken;
    public readonly string MarketRefreshAccessToken;
    public readonly string Username;
```

In case you want to apply custom login, Registration and Login methods are the following:

- **Registration**: `void RegisterMarketAccount (string basicAccessToken, string email, string password, string username = null)`
- **Login**: `void GetMarketAccessToken (string basicAccessToken, string email, string password)`
  ***Username** parameter is optional

Step 13. Market Access Token Description

MarketAccessToken token ensures player's authorization within DMarket and allows to execute all operations with in-game virtual items, including item transferring from a game side to DMarket, and transferring item back from DMarket to a game side. After successful receive of *MarketAccessToken* you can execute virtual items sending and receiving, and also get a player's inventory list.

Step 14. Getting an Inventory List

After player is authorized, a game server can get an inventory list of virtual items which player owns through the following method:

`ServerApi.GetInMarketInventory(marketAccessToken);.`
*__Note__: ServerAPI already has gameToken and adds it automatically to every request, so there is no need to add it by yourself.

Step 15. Sending and Receiving Virtual Items

To send an item to DMarket, apply the following method:
`ServerApi.ToMarket(marketAccessToken, assetId, classId);`

- ❏ Class ID is a unique identifier for your asset's class. It aggregates specific asset IDs (for example, Class ID can be *Sword, Axe, Knife,* or *Shotgun*).
- ❏ Asset ID is the unique identifier of virtual asset itself. Every single asset has its own asset ID, e.g. if user has 5 knives, each of them will have each own asset ID but with the same class ID.

***Note**: ServerAPI already has gameToken and adds it automatically to every request, so there is no need to add it by yourself.

To get an item back from DMarket, apply the following method: `ServerApi.FromMarket(marketAccessToken, assetId);`

***Note**: ServerAPI already has gameToken and adds it automatically to every request, so there is no need to add it by yourself.

Each of these methods also transfer two callbacks:

1. Successful operation callback
2. Error callback.

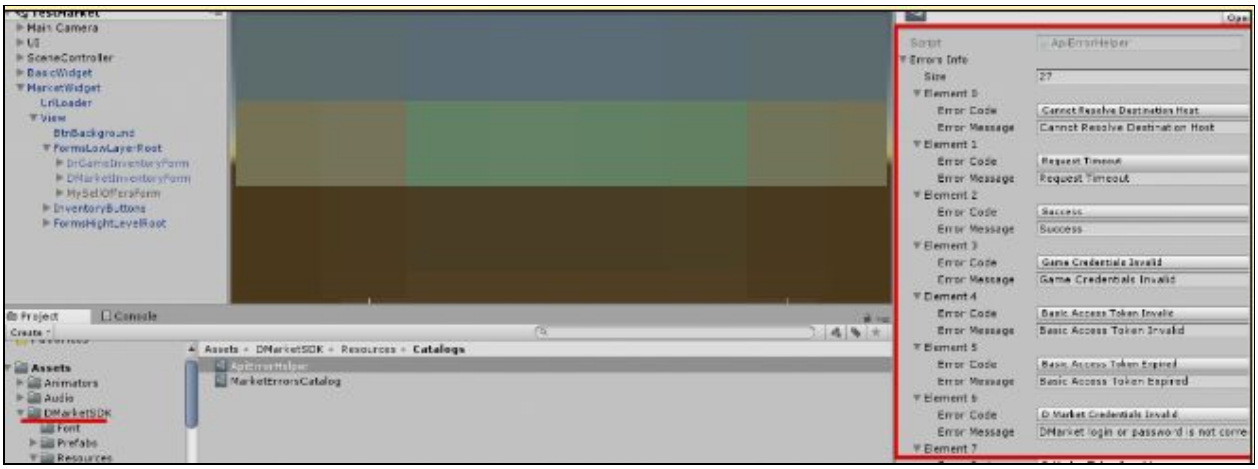
Step 16. Error Handling

Now let's get into handling errors in the widget. To set up error description:

1. Go to DMarket SDK folder, open it.
2. Open **Resources** folder.
3. Open **Catalog** folder and click ***ApiErrorHelper.***

4. On a right you can see a menu for setting up error texts for specific cases (see *"Error Handling"* screenshot).
5. Set up preferred error message for each error code within each **item** in **Error Message** input field.

Error Handling



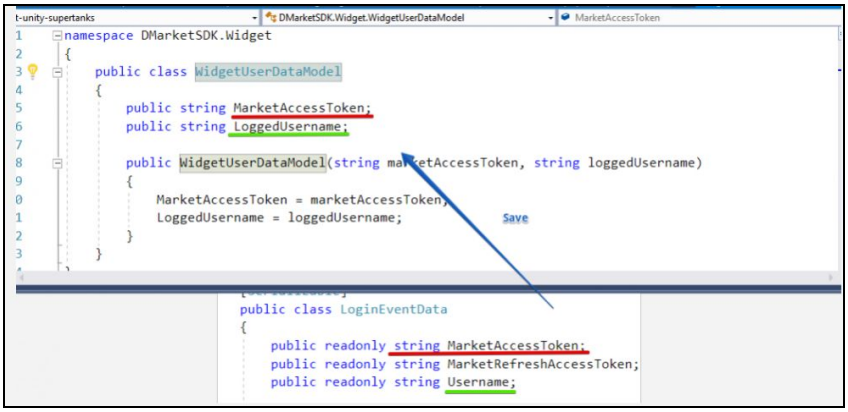
In case error is occurred, you'll receive an event:
`public event Action<Error> ErrorEvent;`
Error list can be found in a chapter **Error Codes**.

Step 17. Setting Up Auto-Login for a Player

DMarket Widget does not store user data on its side, therefore it cannot assure auto-login of a player.
You can save received in `public event Action <LoginEventData> LoginEvent;` data to ensure auto login for player, and initiate player data setup for auto-login through a method:

```
DmarketWidget.SetLoggedUserData(UserDataModel model);
```

User data model example



Step 18. Forced Logout of a Player

To ensure forced logout of a player use a method:
`DmarketWidget.Logout();`

After logging out you will receive the following event: `public event Action LogoutEvent;`

Step 19. Handling Market Token Expiration

Apply `ServerApi.GetMarketRefreshToken(string marketRefreshToken);` method to refresh token when you receive a `DMarketTokenExpired` mistake in `errorCallback`. After you make the call, you will receive a new pair of tokens: `BasicAccessToken` and `BasicRefreshToken`.

Error Codes

Name	Code	Description	Places where can be thrown
Internal	**5000**	Internal Server Error, which is returned in case something went wrong on a server side and therefore normal workflow is impossible.	Anywhere
InvalidRequest	**4000**	Invalid request	Anywhere
GameCredentialsInvalid	**4001**	Invalid game id or game key	Game token request, user token request
BasicAccessTokenInvalid	**4002**	Basic access token not valid, not exists, etc	Not used now
BasicAccessTokenExpired	**4003**	Basic access token expired	Not used now
DMarketCredentialsInvalid	**4004**	Invalid DMarket login or password	User token request
DMarketTokenInvalid	**4005**	Dmarket token not valid	Asset to market, asset from market, in-market inventory request
DMarketTokenExpired	**4006**	Dmarket token expired	Asset to market, asset from market, in-market inventory request

EmptyBasicAccessToken	**4007**	Empty basic access token	Register user, sign-in DMarket user
EmptyDMarketAccessToken	**4008**	Empty DMarket access token	Asset to market, asset from market, get user inventory
EmptyGameUserId	**4101**	Empty game user id	User token request, DMarket account registration
EmptyDMarketCredentials	**4102**	Empty login or password during DMarket sign-in or registration	User token request, DMarket account registration
DMarketLoginAlreadyUsed	**4103**	DMarket login (email) already used by existing user	DMarket account registration
DMarketAccountNotVerified	**4104**	DMarket account should be verified before moving assets to/from market	Asset to market, asset from market
EmptyAssetId	**4105**	Empty asset id	Asset to market, asset from market
EmptyClassId	**4106**	Empty class id	Asset to market
AssetNotInMarket	**4201**	Thrown when tried to move asset from market which is not in market	Asset from market

AssetAlreadyInMarket	**4202**	Thrown when tried to move asset to market which is in market already	Asset to market
AssetNotFound	**4301**	Asset is not found in repository	Get user inventory
ClassNotFound	**4302**	Class is not found in repository	Asset to market
TokensMismatch	**4109**	Game id from dmarket-access token don't march to game id from game-token	Asset to market, asset from market, get user inventory
BasicRefreshTokenInvalid	**4009**	Basic refresh token invalid	Refresh basic token
BasicRefreshTokenExpired	**4010**	Basic refresh token expired	Refresh basic token
DmarketRefreshTokenInvalid	**4011**	Dmarket refresh token invalid	Refresh dmarket token
DmarketRefreshTokenExpired	**4012**	Dmarket refresh token expired	Refresh dmarket token
EmptyAssetClass	**4107**	Invalid request body, empty assetId and classId	Asset to market
InvalidClassId	**4108**	Thrown when tried to move asset to market with invalid classId	Asset to market

PasswordNotEnoughStrong	**4013**	Password is not enough strong	Sign up
SellOfferNotFound	**4400**	Sell offer not found	Get aggregated sell offers, cancel sell offer
AggregatedClassNotFound	**4401**	Aggregated class not found	
AssetAlreadyOnSale	**4402**	Asset already on sale	Create sell offer
EmailIsInvalid	**4019**	Email is invalid	Access token, register user, restore password
NotEnoughMoney	**4403**	Not enough money to make purchase	Buy sell offer
OwnProduct	**4404**	Trying to buy own sell offer	Buy sell offer
GameNotFound	**4405**	Game not found	
ZeroPrice	**4406**	Price less or equal than 0	Create sell offer
EmptyBalance	**4407**		
SellOfferNotBelongToUser	**4408**	Sell offer does not belong to user	Cancel sell offer
SellOfferAlreadyCanceled	**4409**	Sell offer already canceled	Cancel sell offer
SellOfferClosed	**4410**	Sell offer closed	Cancel sell offer
EmptySellOfferId	**4411**	Empty sell offer id	Change sell offer's price

InventoryItemNotFound	**4412**	Inventory item not found	Create sell offer
MiningLimitExceeded	**4112**	Mining limit exceeded	Asset to market
GameTokenEmpty	**4014**	Game token is empty	Get class, save class

Swagger API

Link: https://integration-swagger-ui-e35ab932-e24f-4b69-9fd8-a52f844941ce.dmarket.com/

[End of a Document]