

# Aptos Unity SDK

The Aptos Unity SDK is a .NET implementation of the [Aptos SDK](#), compatible with .NET Standard 2.0 and .NET 4.x for Unity. The goal of this SDK is to provide a set of tools for developers to build multi-platform applications (mobile, desktop, web, VR) using the Unity game engine and the Aptos blockchain infrastructure.

## Getting Started

To get started, you may check our [Quick Start Guide](#). A set of examples is also provided in the `SDK-Examples` directory. A local version of Doxygen-generated documentation for the classes can be found in `Documentation/html/index.html`. A hosted version of the latter documentation can found in [here](#).

This accompanying README file provides further details on the SDK and integration.

## Installation

1. Download the latest `aptos-unity-sdk-xx.unpackage` file from [Release](#)
2. Inside Unity, Click on Assets → Import Packages → Custom Package. and select the downloaded file.

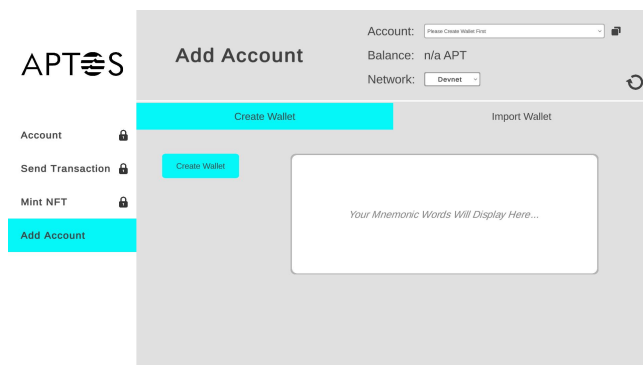
**NOTE:** As of Unity 2021.x.x, Newtonsoft Json is a common dependency. Prior versions of Unity require installing Newtonsoft.

## Quick Start Video

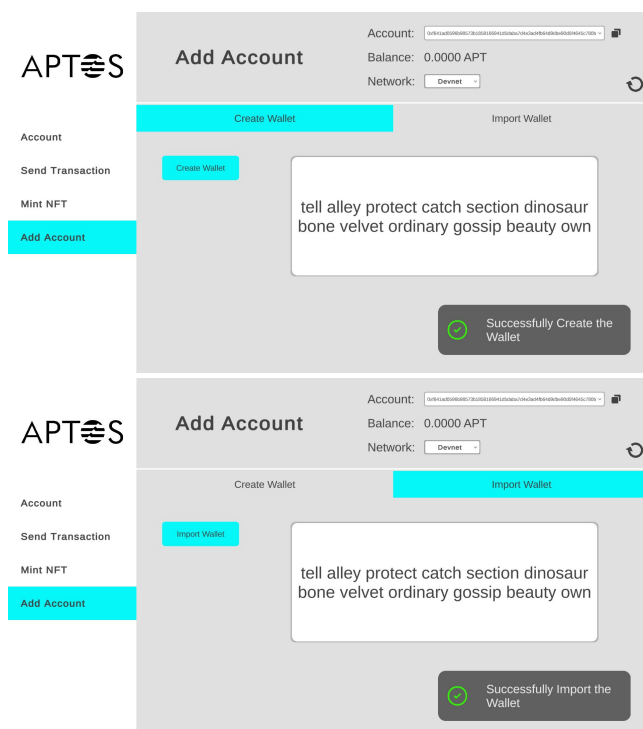
<https://user-images.githubusercontent.com/25370590/219819120-92cb2aa0-e685-4eac-bcb9-942a0cc4157f.mp4>

## Wallet Example Walthrough

You can find a set of examples under `SDK-Examples/SDK Demo` and `SDK-Examples/UI Demo` directory. We will use the scene under `UI Demo` for this walkthrough.



Once you open the demo scene, you will see all tab are locked except `Add Account`, you have the choice to create or import a wallet.



Note that we currently store the mnemonics words in `PlayerPrefs`.

In code, you can create a wallet as follows:

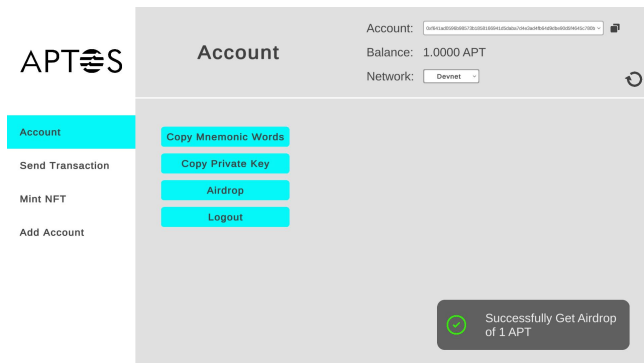
```
// Create Wallet

Mnemonic mnemo = new Mnemonic(Wordlist.English, WordCount.Twelve);
wallet = new Wallet(mnemo);

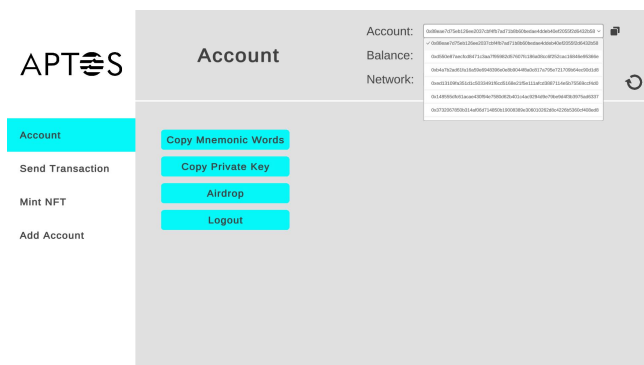
PlayerPrefs.SetString(mnemonicsKey, mnemo.ToString());
```

This wallet object can be used to derive multiple accounts which show in code further down.

## Account



Once you create the wallet, you will be able to unlock rest of the panel, on [Account](#) Panel.



## Deriving Accounts from HD Wallet

In code, you can derive accounts from the HD Wallet by selecting an account index as follows:

```
// Create sub-wallets

Mnemonic mnemo = new Mnemonic(Wordlist.English, WordCount.Twelve);
wallet = new Wallet(mnemo);

for (int i = 0; i < accountNumLimit; i++)
{
    var account = wallet.GetAccount(i);
    var addr = account.AccountAddress.ToString();

    addressList.Add(addr);
}
```

You can also generate an account from a random seed / private key as follows:

```
// Create new account

Account alice = new Account();
AccountAddress aliceAddress = alice.AccountAddress;
```

## Airdrop

When using Devent , you can airdrop 1 APT to your account address as follows:

```
// Airdrop

bool success = false;
ResponseInfo responseInfo = new ResponseInfo();

Coroutine fundAliceAccountCor = StartCoroutine(
    FaucetClient.Instance.FundAccount((_success, _responseInfo) =>
    {
        success = _success;
        responseInfo = _responseInfo;
    }, aliceAddress.ToString(), 10000000, faucetEndpoint));

yield return fundAliceAccountCor;

// Check if funding the account was succesful
if(responseInfo.status != ResponseInfo.Status.Success)
{
    Debug.LogError("Faucet funding for Alice failed: " + responseInfo.message);
    yield break;
}
```

## NFT Minter

[illegible]

On the Mint NFT tab, You can mint a NFT of your own. In order to do that, you need to Create Collection first, then Create NFT .

Note that you must confirm that the creation of the collection was successful before creating the token, you can use the `WaitForTransaction` corouting for this.

```

// Create Collection

string collectionName = "Alice's";
string collectionDescription = "Alice's simple collection";
string collectionUri = "https://aptos.dev";

Transaction createCollectionTxn = new Transaction();

Coroutine createCollectionCor = StartCoroutine(
    RestClient.Instance.CreateCollection((_createCollectionTxn, _responseInfo) =>
    {
        createCollectionTxn = _createCollectionTxn;
        responseInfo = _responseInfo;
    }, alice, collectionName, collectionDescription, collectionUri));
yield return createCollectionCor;

// Check if collection creation was successful
if(responseInfo.status != ResponseInfo.Status.Success)
{
    Debug.LogError("Cannot create collection. " + responseInfo.message);
}

// Check response and transaction hash
Debug.Log("Create Collection Response: " + responseInfo.message);
string transactionHash = createCollectionTxn.Hash;
Debug.Log("Create Collection Hash: " + createCollectionTxn.Hash);

```

```

// Wait for Transaction
bool waitForTxnSuccess = false;
Coroutine waitForTransactionCor = StartCoroutine(
    RestClient.Instance.WaitForTransaction((_pending, _responseInfo) =>
    {
        waitForTxnSuccess = _pending;
        responseInfo = _responseInfo;
    }, transactionHash)
);
yield return waitForTransactionCor;

if(!waitForTxnSuccess)
{
    Debug.LogWarning("Transaction was not found.");
}

```

```
// Create NFT

string tokenName = "Alice's first token";
string tokenDescription = "Alice's simple token";
string tokenUri = "https://aptos.dev/img/nyan.jpeg";

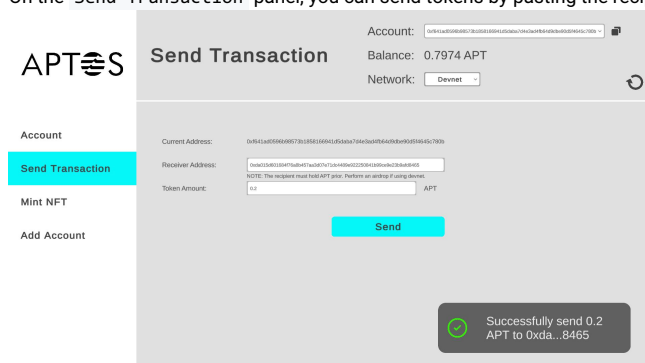
Transaction createTokenTxn = new Transaction();
Coroutine createTokenCor = StartCoroutine(
    RestClient.Instance.CreateToken((_createTokenTxn, _responseInfo) =>
    {
        createTokenTxn = _createTokenTxn;
        responseInfo = _responseInfo;
    }, alice, collectionName, tokenName, tokenDescription, 1, 1, tokenUri, 0)
);
yield return createTokenCor;

if(responseInfo.status != ResponseInfo.Status.Success)
{
    Debug.LogError("Error creating token. " + responseInfo.message);
}

Debug.Log("Create Token Response: " + responseInfo.message);
string createTokenTxnHash = createTokenTxn.Hash;
Debug.Log("Create Token Hash: " + createTokenTxn.Hash);
```

## Transaction Executer

On the **Send Transaction** panel, you can send tokens by pasting the recipient address and token amount.



Below we demonstrate how to send APT to another account.

```
Account alice = new Account();
Account bob = new Account();

Transaction transferTxn = new Transaction();
Coroutine transferCor = StartCoroutine(
    RestClient.Instance.Transfer((_transaction, _responseInfo) => {
        transferTxn = _transaction;
        responseInfo = _responseInfo;
    }, alice, bob.AccountAddress.ToString(), 1000));

yield return transferCor;

if(responseInfo.status != ResponseInfo.Status.Success)
{
    Debug.LogWarning("Transfer failed: " + responseInfo.message);
    yield break;
}

Debug.Log("Transfer Response: " + responseInfo.message);
string transactionHash = transferTxn.Hash;
Debug.Log("Transfer Response Hash: " + transferTxn.Hash);
```

# Technical Details

---

## Core Features

- HD Wallet Creation & Recovery
- Account Management
  - Account Recovery
  - Message Signing
  - Message Verification
  - Transaction Management
  - Single / Multi-signer Authentication
  - Authentication Key Rotation
- Native BCS Support
- Faucet Client for Devnet

## Unity Support

Supported Version:		Tested		
2021.3.x		✓		
2022.2.x		✓		
Windows	Mac	iOS	Android	WebGL
✓	✓	✓	✓	✓

## Dependencies

- [Chaos.NaCl.Standard](#)
- Microsoft.Extensions.Logging.Abstractions.1.0.0 — required by NBitcoin.7.0.22
- Newtonsoft.Json
- NBitcoin.7.0.22
- [Portable.BouncyCastle](#)

## Support

---

For additional support, please join our community [Discord Server](#), and ask questions in the `#dev-discussion` channel.