Aptos-Unity-SDK

Aptos Made Easy in Unity

Made with Unity AptosLabs 1564 MEMBERS

Aptos-Unity-SDK is a Unity package written in C# to help developers integrate Aptos blockchain technology into their projects.

Features

- ☑ Generate new wallets.
- ☑ Create new accounts using the ED25519 Key Standard.
- ☑ Simulate and submit transaction.
- ☑ Create new collections.
- ☑ Create new tokens.
- ☑ Check account information (Token and APT balances, submitted contracts, etc).

- ✓ Import previously used wallets using BIP-39 and BIP-32 Mnemonic seeds or an ED25519 private key.
- ✓ Create arbitrary tokens.
- ✓ Compatibility with main, dev, and test networks.
- ✓ Comprehensive Unit and Integration Test coverage.

Make Account Transfer Quickly and Easily!

With how easy the library is to use, Aptos-Unity-SDK gives you the power to quickly setup a Aptos account and be able to transfer coins in only a few lines of code:

```
// Initialize Accounts.
Account alice = Account.Generate();
Account bob = Account.Generate();

// Initialize Transaction.
Transaction transferTxn = new Transaction();

// Initialize Response Info.
ResponseInfo responseInfo = new ResponseInfo();

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

// Yield Return the Coroutine.
yield return transferCor;
```

Requirements

Platforms	Unity Version	Installation	Status
Windows / Mac / iOS / Android / WebGL	2021.3.x	Unity Package Importer	Fully Tested
Windows / Mac / iOS / Android / WebGL	2022.2.x	Unity Package Importer	Fully Tested

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

Installation

Unity Package Importer

- Download the latest `aptos-unity-sdk-xx.unitypackage` file from Release
- 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.

Examples and Test Suite

A set of examples (scenes & scripts) can be found in the following directory: `Assets/Aptos-Unity-SDK/SDK-Examples/SDK Demo/Scenes/`.

Each scene contains a `Web3Controller` object with the required client scripts.

- **AptosTokenClientExample** scene - mint collections, mint tokens, update token properties, read objects
- **MultisignExample** scene - perform a multisig transaction
- **SimpleNftExample** scene - simple NFT transaction
- **SimulateTransactionExample** scene - simulate a set of transactions
- **TransferCoinExample** scene - a simplem transfer coin transaction

The SDK's test suite can be found in the following directory: `Assets/Aptos-Unity-SDK/Test/`.

The test suite covers:

- Account - private / public keys, signatures and verification
- Transactions - creation and serialization
- BCS serialization and deserialization

Using Aptos-Unity-SDK

Aptos-Unity-SDK is designed to be very easy to integrate into your own Unity projects. The main functionality comes from several key classes: `RestClient`, `FacetClient`, `TokenClient`, `EntryFunction`, `Account`, and `Wallet`.

There are three core client classes:

- **FaucetClient** - used to request for airdrops
- **RESTClient** - used to query the aptos blockchain
- **AptosClient** - a REST client wrapper used to interact with Aptos tokens

Let's go over each of the classes, along with examples for each to demonstrate their power and flexibility.

RestClient

The REST Client provides you with the fundamental transaction endpoints needed for interacting with the Aptos Blockchain. The following showcases how to initialize the `RestClient` and `AptosTokenClient`.

```
RestClient restClient =
RestClient.Instance.SetEndPoint(Constants.DEVNET_BASE_URL);
Coroutine restClientSetupCor =
StartCoroutine(RestClient.Instance.SetUp());
yield return restClientSetupCor;

AptosTokenClient tokenClient =
AptosTokenClient.Instance.SetUp(restClient);
```

As shown before, it only take a few lines of code to initialize a transfer for Aptos coins. This is the main class developers will be leveraging to interact directly with the Aptos Blockchain via REST Client calls. Here's another example showing how to create a collection:

```
// Initialize Account.
Account alice = Account.Generate();

// Initialize Response Info.
ResponseInfo responseInfo = new ResponseInfo();

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

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

Here's also how to wait for a transaction:

```
// Initialize Accounts.
Account alice = Account.Generate();
Account bob = Account.Generate();

// Initialize Response Info.
ResponseInfo responseInfo = new ResponseInfo();

// Initialize Transaction.
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;
}

// 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. Breaking out of example",
gameObject);
    yield break;
}
```

FaucetClient

The Faucet Client allows the developer to leverage the ability to fund wallets on any of the non-main networks within the Aptos Blockchain. This can easily speed up development times through automating the process of funding wallets. Here's an example on how to use the Faucet Client:

```
// Initialize Account.
Account alice = Account.Generate();

// Initialize Response Info.
ResponseInfo responseInfo = new ResponseInfo();

// Initialize Faucet Endpoint.
private string faucetEndpoint = "https://faucet.devnet.aptoslabs.com";

// Initialize Funding Request.
Coroutine fundAliceAccountCor =
StartCoroutine(FaucetClient.Instance.FundAccount((_success, _responseInfo)
=>
{
}, aliceAddress.ToString(), 100000000, faucetEndpoint));
yield return fundAliceAccountCor;
```

TokenClient

The Token Client provides the ability for the developer to have an easier time implementing NFT mechanics into their projects. Here's an example on minting an NFT using the client:

```
// Initialize Account.
Account alice = Account.Generate();

// Initialize Response Info.
ResponseInfo responseInfo = new ResponseInfo();

// Token Details.
string collectionName = "Alice's";
string tokenName = "Alice's first token";

// Initialize Mint Token Transaction Digest.
string mintTokenTxn = "";

// Initialize Token Mint.
Coroutine mintTokenCor =
StartCoroutine(tokenClient.MintToken((_mintTokenTxn, _responseInfo) =>
{
    mintTokenTxn = _mintTokenTxn;
    responseInfo = _responseInfo;
},
    alice,
    collectionName,
    "Alice's simple token",
    tokenName,
    "https://aptos.dev/img/nyan.jpeg",
    new PropertyMap(new List<Property> { Property.StringProp("string",
"string value") })
));
yield return mintTokenCor;
```

EntryFunction

If a developer needs more flexibility with how they want to shape their transactions, e.g., arbitrary, generic, custom, using EntryFunction is the key class, along with the usage of the REST Client, to submit those types of transactions that aren't defined already. This is how the developer would initialize the transaction arguments, create the EntryFunction payload, and submit the transaction using BCS:

```
// Initialize Account.
Account Alice = Account.Generate();
```

```
// Initialize Variables.
PropertyMap Properties = new PropertyMap(new List<Property> {
Property.StringProp("string", "string value") });
string Collection = "My New Collection";
string Description = "This is my first Collection.";
string Name = "Aptos";
string Uri = "https://aptos.dev";
Tuple<List<BString>, List<BString>, List<byte[]>> propertiesTuple =
Properties.ToTuple();

// Initialize Transaction Arguments.
ISerializable[] transactionArguments =
{
    new BString(Collection),
    new BString(Description),
    new BString(Name),
    new BString(Uri),
    new Sequence(propertiesTuple.Item1.ToArray()),
    new Sequence(propertiesTuple.Item2.ToArray()),
    new BytesSequence(propertiesTuple.Item3.ToArray())
};

// Initialize the Payload.
EntryFunction payload = EntryFunction.Natural(
    new ModuleId(AccountAddress.FromHex("0x4"), "aptos_token"),  //
Package ID and Module Name.
    "mint",  // Function Name.
    new TagSequence(new ISerializableTag[0]),  // Type Arguments.
    new Sequence(transactionArguments)  // Arguments.
);

// Execute Tranaction via BCS.
SignedTransaction signedTransaction = null;

Coroutine cor_createBcsSIgnedTransaction =
StartCoroutine(Client.CreateBCSSignedTransaction((_signedTransaction) =>
{
    signedTransaction = _signedTransaction;
}, Alice, new BCS.TransactionPayload(payload)));

yield return cor_createBcsSIgnedTransaction;
```

Account

Accounts within the SDK represent the core part of a Wallet, that give ease of access to the needed information you'd need for communicating with the Aptos Blockchain. Here are some example initializations of accounts:

```
// Generate Random Account.
alice = Account.Generate();
```

```
// Initialize Account Using Hexadecimal Private Key.
const string PrivateKeyHex =
"0x64f57603b58af16907c18a866123286e1cbce89790613558dc1775abb3fc5c8c";
bob = Account.LoadKey(PrivateKeyHex);

// Initialize Account Using Private and Public Key Bytes.
private static readonly byte[] PrivateKeyBytes = {
    100, 245, 118, 3, 181, 138, 241, 105,
    7, 193, 138, 134, 97, 35, 40, 110,
    28, 188, 232, 151, 144, 97, 53, 88,
    220, 23, 117, 171, 179, 252, 92, 140
};
private static readonly byte[] PublicKeyBytes = {
    88, 110, 60, 141, 68, 125, 118, 121,
    34, 46, 19, 144, 51, 227, 130, 2,
    53, 227, 61, 165, 9, 30, 155, 11,
    184, 241, 161, 18, 207, 12, 143, 245
};
Account chad = new Account(PrivateKeyBytes, PublicKeyBytes);
```

From there, you're able to retrieve the byte array of the private and public keys, along with signing and verifying messages and transactions, as the core function of an Account object, as shown here:

```
// Retrieve Private Key.
var privateKey = chad.PrivateKey;

// Retrieve Public Key.
var publicKey = chad.PublicKey;
```

The developer can now use the Account to sign various messages and transactions for interacting with the Aptos Blockchain, as shown here:

```
// Create a Signature Object Storing the Signature of the Signed Message.
private static readonly byte[] MessageUt8Bytes = {
    87, 69, 76, 67, 79, 77, 69, 32,
    84, 79, 32, 65, 80, 84, 79, 83, 33
};
var signature = privateKey.Sign(MessageUt8Bytes);
```

Developers can also verify the integrity of the message using the public key, as shown here:

```
// Initiailize Verified Bool Object.
bool verified = publicKey.Verify(MessageUt8Bytes, signature);
```

Wallet

Wallets will be the primary method of accessing accounts on the Aptos Blockchain via Mnemonic Keys, since they'll allow you to generate multiple accounts with ease. Here's an example on how to initialize a wallet using a mnemonic key:

```
// Initializing Wallet.
string mnemo = "stadium valid laundry unknown tuition train december
camera fiber vault sniff ripple";
Wallet wallet = new Wallet(mnemo);

// Initialize A Random Wallet.
Mnemonic mnemo = new Mnemonic(Wordlist.English, WordCount.Twelve);
Wallet wallet = new Wallet(mnemo);
```

This provides the developer with what's known as an HD Wallet (Hierarchical Deterministic Wallet), which is what will enable to generate as many private keys from the wallet as they want. Here's different ways on how to retrieve the account(s) from the Wallet, along with deriving the mnemonic seed from the Wallet; which is the seed that's derived from the input mnemonic phrase and is what allows the developer to generate a number accounts from the Wallet:

```
// Get the Initial Main Account.
var mainAccount = wallet.Account;

// Get Any Other Accounts Created / Derived From the Wallet (i represents
the index from 0).
var account = wallet.GetAccount(i);

// Derive Mnemonic Seed from Wallet.
var seed = wallet.DeriveMnemonicSeed();
```

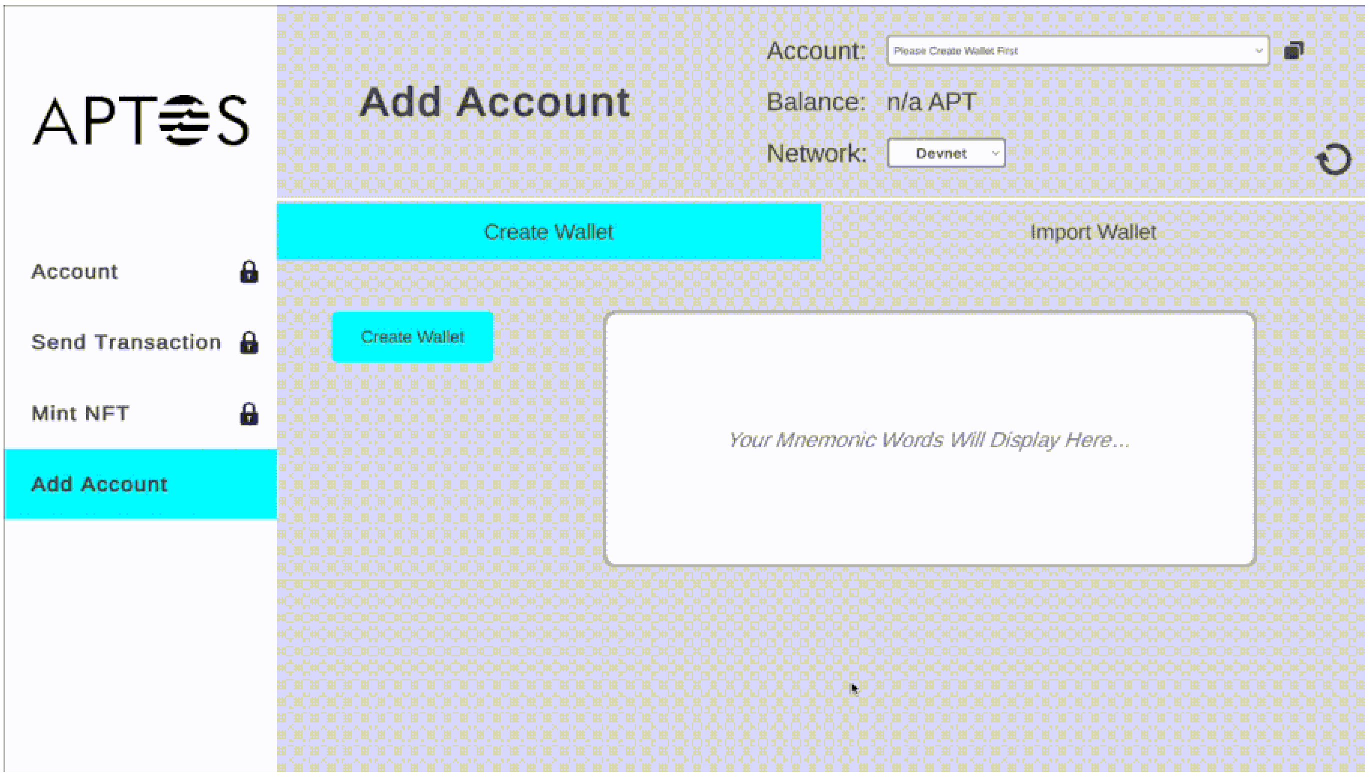
The Wallet object can also allow the main account to sign and verify data, as shown here:

```
// Initialize a Signature Object.
private static readonly byte[] MessageUt8Bytes = {
    87, 69, 76, 67, 79, 77, 69, 32,
    84, 79, 32, 65, 80, 84, 79, 83, 33
};
var signature = wallet.Account.Sign(MessageUt8Bytes);

// Initialize a Boolean Verified.
bool verified = wallet.Account.Verify(MessageUt8Bytes, signature);
```

Examples

The SDK comes with several examples and a demo project that show how to leverage the SDK to its full potential. The examples include `AptosToken`, `Multisig`, `SimulateTransferCoin`, `TransferCoin`, and `SimpleNftExample`. As well, the demo project, shown below, is an example project that gives you various options to test out.



Add Account

Account: Please Create Wallet First

Balance: n/a APT

Network: Devnet

Your Mnemonic Words Will Display Here...

License

Aptos-Unity-SDK is released under the Apache 2.0 license. See LICENSE for details.