DeepBookV3 SDK

The DeepBook typescript SDK abstracts away the transaction calls, allowing for direct interactions with the DeepBook package.

To use the SDK in your projects, install the @mysten/deepbook package.

The DeepBookV3 SDK includes a constants file (/utils/constants.ts) that maintains the latest deployed addresses for DeepBook, as well as a few staple coins and pools.

constants.ts

To work with DeepBook, you must create a DeepBookClient . To construct the DeepBookClient , pass in a SuiClient , the sender address, and environment. The Sui TypeScript SDK provides the SuiClient and key functionality necessary to process transactions. The following example imports those libraries, as well.

Functions that require the input of a coin, pool, or a manager require the key of any such object as the parameter. The SDK manages a key :value relationship of this data in memory. Some default data comes with the SDK (as seen in utils/constants.ts). Coins are stored in a CoinMap and pools in a PoolMap in the config.

Before placing any trade, you must supply a balance manager address to the client. The manager key points to an object defined by the BalanceManager interface in the client. BalanceManager docs . Initialize the balance manager with the client. If you don't create a balance manager, you can rely on the client to create one, but then the user must reinitialize the client.

Example using an existing balance manager:

Example creating a balance manager:

The SDK comes with four default coins on Testnet and five default coins on Mainnet.

Default Testnet coins

Default Mainnet coins

You can also initialize the SDK with custom coins to interact with pools that are not supported by default. To do this, create a CoinMap object and pass it to the constructor of the client.

Similar to coins, the SDK comes with default pools. You can provide a PoolMap during construction to override this behavior.

The following example uses the default pools and coins provided.

Install

To use the SDK in your projects, install the @mysten/deepbook package.

The DeepBookV3 SDK includes a constants file (/utils/constants.ts) that maintains the latest deployed addresses for DeepBook, as well as a few staple coins and pools.

constants.ts

To work with DeepBook, you must create a DeepBookClient . To construct the DeepBookClient , pass in a SuiClient , the sender address, and environment. The Sui TypeScript SDK provides the SuiClient and key functionality necessary to process transactions. The following example imports those libraries, as well.

Functions that require the input of a coin, pool, or a manager require the key of any such object as the parameter. The SDK manages a key :value relationship of this data in memory. Some default data comes with the SDK (as seen in utils/constants.ts). Coins are stored in a CoinMap and pools in a PoolMap in the config.

Before placing any trade, you must supply a balance manager address to the client. The manager key points to an object defined by the BalanceManager interface in the client. BalanceManager docs . Initialize the balance manager with the client. If you don't create a balance manager, you can rely on the client to create one, but then the user must reinitialize the client.

Example using an existing balance manager:

Example creating a balance manager:

The SDK comes with four default coins on Testnet and five default coins on Mainnet.

Default Testnet coins

Default Mainnet coins

You can also initialize the SDK with custom coins to interact with pools that are not supported by default. To do this, create a CoinMap object and pass it to the constructor of the client.

Similar to coins, the SDK comes with default pools. You can provide a PoolMap during construction to override this behavior.

The following example uses the default pools and coins provided.

Constants

The DeepBookV3 SDK includes a constants file (/utils/constants.ts) that maintains the latest deployed addresses for DeepBook, as well as a few staple coins and pools.

constants.ts

To work with DeepBook, you must create a DeepBookClient . To construct the DeepBookClient , pass in a SuiClient , the sender address, and environment. The [Sui TypeScript SDK](#) provides the SuiClient and key functionality necessary to process transactions. The following example imports those libraries, as well.

Functions that require the input of a coin, pool, or a manager require the key of any such object as the parameter. The SDK manages a key :value relationship of this data in memory. Some default data comes with the SDK (as seen in utils/constants.ts). Coins are stored in a CoinMap and pools in a PoolMap in the config.

Before placing any trade, you must supply a balance manager address to the client. The manager key points to an object defined by the BalanceManager interface in the client. [BalanceManager docs](#) . Initialize the balance manager with the client. If you don't create a balance manager, you can rely on the client to create one, but then the user must reinitialize the client.

Example using an existing balance manager:

Example creating a balance manager:

The SDK comes with four default coins on Testnet and five default coins on Mainnet.

Default Testnet coins

Default Mainnet coins

You can also initialize the SDK with custom coins to interact with pools that are not supported by default. To do this, create a CoinMap object and pass it to the constructor of the client.

Similar to coins, the SDK comes with default pools. You can provide a PoolMap during construction to override this behavior.

The following example uses the default pools and coins provided.

DeepBookClient

To work with DeepBook, you must create a DeepBookClient . To construct the DeepBookClient , pass in a SuiClient , the sender address, and environment. The [Sui TypeScript SDK](#) provides the SuiClient and key functionality necessary to process transactions. The following example imports those libraries, as well.

Functions that require the input of a coin, pool, or a manager require the key of any such object as the parameter. The SDK manages a key :value relationship of this data in memory. Some default data comes with the SDK (as seen in utils/constants.ts). Coins are stored in a CoinMap and pools in a PoolMap in the config.

Before placing any trade, you must supply a balance manager address to the client. The manager key points to an object defined by the BalanceManager interface in the client. [BalanceManager docs](#) . Initialize the balance manager with the client. If you don't create a balance manager, you can rely on the client to create one, but then the user must reinitialize the client.

Example using an existing balance manager:

Example creating a balance manager:

The SDK comes with four default coins on Testnet and five default coins on Mainnet.

Default Testnet coins

Default Mainnet coins

You can also initialize the SDK with custom coins to interact with pools that are not supported by default. To do this, create a CoinMap object and pass it to the constructor of the client.

Similar to coins, the SDK comes with default pools. You can provide a PoolMap during construction to override this behavior.

The following example uses the default pools and coins provided.

Keys: Coin, Pool, and Manager

Functions that require the input of a coin, pool, or a manager require the key of any such object as the parameter. The SDK manages a key :value relationship of this data in memory. Some default data comes with the SDK (as seen in utils/constants.ts). Coins are stored in a CoinMap and pools in a PoolMap in the config.

Before placing any trade, you must supply a balance manager address to the client. The manager key points to an object defined by the BalanceManager interface in the client. BalanceManager docs . Initialize the balance manager with the client. If you don't create a balance manager, you can rely on the client to create one, but then the user must reinitialize the client.

Example using an existing balance manager:

Example creating a balance manager:

The SDK comes with four default coins on Testnet and five default coins on Mainnet.

Default Testnet coins

Default Mainnet coins

You can also initialize the SDK with custom coins to interact with pools that are not supported by default. To do this, create a CoinMap object and pass it to the constructor of the client.

Similar to coins, the SDK comes with default pools. You can provide a PoolMap during construction to override this behavior.

The following example uses the default pools and coins provided.