Connect SDK

The Connect SDK is a TypeScript SDK for interacting with the chains Wormhole supports and the protocols built on top of Wormhole.

Warning

⚠ This package is a Work in Progress so the interface may change and there are likely bugs. Please report any issues you find.⚠

Installation

Install the primary package

```

Copy npminstall@wormhole-foundation/connect-sdk

```

As well as any other platforms you wish to use

```

Copy npminstall@wormhole-foundation/connect-sdk-evm npminstall@wormhole-foundation/connect-sdk-solana npminstall@wormhole-foundation/connect-sdk-algorand

```

And any protocols you intend to use

```

Copy npminstall@wormhole-foundation/connect-sdk-evm-tokenbridge npminstall@wormhole-foundation/connect-sdk-solana-tokenbridge npminstall@wormhole-foundation/connect-sdk-algorand-tokenbridge

```

Usage

A developer would use the connect-sdk package in conjunction with one or more of the chain context packages. Most developers don't use every single chain and may only use a couple, which allows developers to import only the dependencies they actually need.

Getting started is simple, just import and pass in the Platform modules you wish to support as an argument to the Wormhole class.

```

Copy import{ Wormhole,Signer }from"@wormhole-foundation/connect-sdk"; import{ EvmPlatform }from"@wormhole-foundation/connect-sdk-evm"; import{ SolanaPlatform }from"@wormhole-foundation/connect-sdk-solana"; import{ AlgorandPlatform }from"@wormhole-foundation/connect-sdk-algorand";

// Include the protocols you wish to use // these imports register the protocol to be used by higher level abstractions // like WormholeTransfers import"@wormhole-foundation/connect-sdk-evm-tokenbridge"; import"@wormhole-foundation/connect-sdk-solana-tokenbridge"; import"@wormhole-foundation/connect-sdk-algorand-tokenbridge";

constnetwork="Mainnet";// Or "Testnet" constwh=newWormhole(network,[EvmPlatform,SolanaPlatform,AlgorandPlatform]);

// Get a ChainContext object for a specific chain // Useful to do things like get the rpc client, make Protocol clients, // look up configuration parameters or even fetch balances constsrcChain=wh.getChain("Ethereum");

// In this SDK, the string literal 'native' is an alias for the gas token // native to the chain constbalance=awaitsrcChain.getBalance("0xdeadbeef...","native")// => BigInt

// returns a TokenBridge client for srcChain awaitsrcChain.getTokenBridge();// => TokenBridge<'Evm'> // util to grab an RPC client, cached after the first call srcChain.getRpc();// => RpcConnection<'Evm'>

```

Concepts

Understanding several higher level concepts of the SDK will help in using it effectively.

Platforms

Every chain is its own special snowflake but many of them share similar functionality. ThePlatform modules provide a consistent interface for interacting with the chains that share a platform.

Each platform can be installed separately so that dependencies can stay as minimal as possible.

Chain Context

TheWormhole class provides agetChain method that returns aChainContext object for a given chain. This object provides access to the chain specific methods and utilities. Much of the functionality in theChainContext is provided by thePlatform methods but the specific chain may have overridden methods.

The ChainContext object is also responsible for holding a cached rpc client and protocol clients.

```

Copy // Get the chain context for the source and destination chains // This is useful to grab direct clients for the protocols constsrcChain=wh.getChain(senderAddress.chain); constdstChain=wh.getChain(receiverAddress.chain);

consttb=awaitsrcChain.getTokenBridge();// => TokenBridge<'Evm'> srcChain.getRpcClient();// => RpcClient<'Evm'>

```

Addresses

Within the Wormhole context, addresses are oftennormalized to 32 bytes and referred to in this SDK as aUniversalAddresses .

Each platform comes with an address type that understands the native address formats, unsurprisingly referred to as NativeAddress. This abstraction allows the SDK to work with addresses in a consistent way regardless of the underlying chain.

```

Copy // Its possible to convert a string address to its Native address constethAddr:NativeAddress<"Evm">=toNative("Ethereum","0xbeef...");

// A common type in the SDK is theChainAddress which provides // the additional context of theChain this address is relevant for. constsenderAddress:ChainAddress=Wormhole.chainAddress("Ethereum","0xbeef..."); constreceiverAddress:ChainAddress=Wormhole.chainAddress("Solana","Sol1111...");

// Convert the ChainAddress back to its canonical string address format conststrAddress=Wormhole.canonicalAddress(senderAddress);// => '0xbeef...'

// Or if the ethAddr above is for an emitter and you need the UniversalAddress constemitterAddr=ethAddr.toUniversalAddress().toString()

```

Tokens

Similar to theChainAddress type, theTokenId type provides the Chain and Address of a given Token.

```

Copy // Returns a TokenId constsourceToken:TokenId=Wormhole.tokenId("Ethereum","0xbeef...");

// Whereas the ChainAddress is limited to valid addresses, a TokenId may // have the string literal 'native' to consistently denote the native // gas token of the chain constgasToken:TokenId=Wormhole.tokenId("Ethereum","native");

// the same method can be used to convert the TokenId back to its canonical string address format conststrAddress=Wormhole.canonicalAddress(senderAddress);// => '0xbeef...'

```

Signers

In order to sign transactions, an object that fulfils theSigner interface is required. This is a simple interface that can be implemented by wrapping a web wallet or other signing mechanism.

```
Copy // A Signer is an interface that must be provided to certain methods // in the SDK to sign transactions. It can be either a SignOnlySigner // or a SignAndSendSigner depending on circumstances. // A Signer can be implemented by wrapping an existing offline wallet // or a web wallet exporttypeSigner=SignOnlySigner|SignAndSendSigner;

// A SignOnlySender is for situations where the signer is not // connected to the network or does not wish to broadcast the // transactions themselves exportinterfaceSignOnlySigner{ chain():ChainName; address():string; // Accept an array of unsigned transactions and return // an array of signed and serialized transactions. // The transactions may be inspected or altered before // signing. // Note: The serialization is chain specific, if in doubt, // see the example implementations linked below sign(tx:UnsignedTransaction[]):Promise; }

// A SignAndSendSigner is for situations where the signer is // connected to the network and wishes to broadcast the // transactions themselves exportinterfaceSignAndSendSigner{ chain():ChainName; address():string; // Accept an array of unsigned transactions and return // an array of transaction ids in the same order as the // UnsignedTransactions array. signAndSend(tx:UnsignedTransaction[]):Promise; }
```

See the testing signers (Evm ,Solana , ...) for an example of how to implement a signer for a specific chain or platform.

Protocols

While Wormhole itself is a Generic Message Passing protocol, a number of protocols have been built on top of it to provide specific functionality.

Each Protocol, if available, will have a Platform specific implementation. These implementations provide methods to generate transactions or read state from the contract on-chain.

Wormhole Core

The protocol that underlies all Wormhole activity is the Core protocol. This protocol is responsible for emitting the message containing the information necessary to perform bridging includingEmitter address , theSequence number for the message and the Payload of the message itself.

```
Copy // ... // register the protocol import"@wormhole-foundation/connect-sdk-solana-core";

// ...

// get chain context constchain=wh.getChain("Solana"); // Get the core brige client for Solana constcoreBridge=awaitchain.getWormholeCore(); // Generate transactions necessary to simply publish a message constpublishTxs=coreBridge.publishMessage(address.address,encoding.bytes.encode("lol"),0,0);

// ... posibly later, after fetching the signed VAA, generate the transactions to verify the VAA constverifyTxs=coreBridge.verifyMessage(address.address,vaa);
```

Within the payload is the information necessary to perform whatever action is required based on the Protocol that uses it.

Token Bridge

The most familiar protocol built on Wormhole is the Token Bridge.

Every chain has aTokenBridge protocol client that provides a consistent interface for interacting with the Token Bridge. This includes methods to generate the transactions required to transfer tokens, as well as methods to generate and redeem attestations.

Using theWormholeTransfer abstractions is the recommended way to interact with these protocols but it is possible to use them directly

```
Copy import{ signSendWait }from"@wormhole-foundation/connect-sdk";

import"@wormhole-foundation/connect-sdk-evm-tokenbridge";

// ...

consttb=awaitsrcChain.getTokenBridge();// => TokenBridge<'Evm'>
```

consttoken="0xdeadbeef..."; consttxGenerator=tb.createAttestation(token);// => AsyncGenerator consttxids=awaitsignSendWait(srcChain,txGenerator,src.signer);// => TxHash[]

```

Supported protocols are defined in the[definitions module](#) .

Wormhole Transfer

While using the[ChainContext](#) and[Protocol](#) clients directly is possible, to do things like transfer tokens, the SDK provides some helpful abstractions.

TheWormholeTransfer interface provides a convenient abstraction to encapsulate the steps involved in a cross-chain transfer.

Token Transfers

Performing a Token Transfer is trivial for any source and destination chains.

We can create a newWormhole object and use it to to createTokenTransfer ,CircleTransfer ,GatewayTransfer , etc. objects to transfer tokens between chains. The transfer object is responsible for tracking the transfer through the process and providing updates on its status.

```

Copy // We'll send the native gas token on source chain consttoken="native";

// Format it for base units constamount=baseUnits(parseAmount(1,srcChain.config.nativeTokenDecimals));

// Create a TokenTransfer object, allowing us to shepherd the transfer through the process and get updates on its status constmanualXfer=wh.tokenTransfer(token,// TokenId of the token to transfer or 'native' amount,// Amount in base units senderAddress,// Sender address on source chain recipientAddress,// Recipient address on destination chain false,// No Automatic transfer);

// 1) Submit the transactions to the source chain, passing a signer to sign any txns constsrcTxids=awaitmanualXfer.initiateTransfer(src.signer);

// 2) Wait for the VAA to be signed and ready (not required for auto transfer) // Note: Depending on chain finality, this timeout may need to be increased. // See https://docs.wormhole.com/wormhole/reference/constants#consistency-levels for more info on specific chain finality. consttimeout=60_000; constattestIds=awaitmanualXfer.fetchAttestation(timeout);

// 3) Redeem the VAA on the destination chain constdestTxids=awaitmanualXfer.completeTransfer(dst.signer);

```

Internally, this uses the[TokenBridge](#) protocol client to transfer tokens. TheTokenBridge protocol, like other Protocols, provides a consistent set of methods across all chains to generate a set of transactions for that specific chain.

See the example[here](#) .

Native USDC Transfers

We can also transfer native USDC using[Circle's CCTP](#)

```

Copy // OR for an native USDC transfer constusdcXfer=wh.cctpTransfer(1_000_000n,// amount in base units (1 USDC) senderAddress,// Sender address on source chain recipientAddress,// Recipient address on destination chain false,// Automatic transfer);

// 1) Submit the transactions to the source chain, passing a signer to sign any txns constsrcTxids=awaitusdcXfer.initiateTransfer(src.signer);

// 2) Wait for the Circle Attestations to be signed and ready (not required for auto transfer) // Note: Depending on chain finality, this timeout may need to be increased. // See https://developers.circle.com/stablecoin/docs/cctp-technical-reference#mainnet for more consttimeout=120_000; constattestIds=awaitusdcXfer.fetchAttestation(timeout);

// 3) Redeem the Circle Attestation on the dest chain constdestTxids=awaitusdcXfer.completeTransfer(dst.signer);

```

See the[example here](#) .
```

Automatic Transfers

Some transfers allow for automatic relaying to the destination, in that case only theinitiateTransfer is required. The status of the transfer can be tracked by periodically checking the status of the transfer object.

```

Copy // OR for an automatic transfer constautomaticXfer=wh.tokenTransfer( "native",// send native gas on source chain amt,// amount in base units senderAddress,// Sender address on source chain recipientAddress,// Recipient address on destination chain true,// Automatic transfer );

// 1) Submit the transactions to the source chain, passing a signer to sign any txns constsrcTxids=awaitautomaticXfer.initiateTransfer(src.signer); // 2) If automatic, we're done, just wait for the transfer to complete if(automatic)returnwaitLog(automaticXfer);

```

Gateway Transfers

Gateway transfers are transfers that are passed through the Wormhole Gateway to or from Cosmos chains.

See the examplehere .

Recovering Transfers

It may be necessary to recover a transfer that was abandoned before being completed. This can be done by instantiating the Transfer class with thefrom static method and passing one of several types of identifiers.

ATransactionId may be used

```

Copy // Note, this will attempt to recover the transfer from the source chain // and attestation types so it may wait depending on the chain finality // and when the transactions were issued. consttimeout=60_000; constxfer=awaitTokenTransfer.from( { chain:"Ethereum", txid:"0x1234...", }, timeout, );

constdstTxIds=awaitxfer.completeTransfer(dst.signer);

```

Or aWormholeMessageId if aVAA is generated

```

Copy constxfer=awaitTokenTransfer.from({ chain:"Ethereum", emitter:toNative("Ethereum",emitterAddress).toUniversalAddress(), sequence:"0x1234...", });

constdstTxIds=awaitxfer.completeTransfer(dst.signer);

```

See also

The tsdoc is availablehere

Last updated1 month ago