Chain Abstraction Guide

A chain abstraction integration will generally involve two steps: writing an adapter contract and constructingcalldata for users to send.

TheChain Abstraction Reference is an example repository with adapter contracts and a frontend that implements the full stack of the integration. We'll be referencing code from it in this guide.

Adapter Contract

The adapter is a smart contract that integrators must build. It should implement thexReceive interface and be deployed to all destination chains that the integrating protocol supports. After the user's transaction has been dispatched and their funds/data arrive on the destination chain, this contract will be called to handle all the execution logic for the protocol.

The execution logic includes a swap and a "forward call". The swap converts the Connext-bridged asset into the asset that will be used for the protocol's function call (a.k.a "target function"). The forward call is where the adapter contract actually calls the "target function".

The following sections will walk through how to implement these two parts.

Installation

Assuming you're usingFoundry , you can add the Connext contracts to your project by running the following command to install theconnext-integration repository as a submodule:

```

Copy forgeinstallconnext/connext-integration

```

The library will be installed tolib/connext-integration .

SwapForwarderXReceiver

TheSwapForwarderXReceiver is an audited abstract contract that should be implemented by adapter contracts.

```

Copy abstractcontractSwapForwarderXReceiverisForwarderXReceiver,SwapAdapter{ usingAddressforaddress;

///@devThe address of the Connext contract on this domain. constructor(address_connext)ForwarderXReceiver(_connext) {}

/// INTERNAL / *@noticePrepare the data by calling to the swap adapter. Return the data to be swapped*@devThis is called by the xReceive function so the input data is provided by the Connext bridge. *@param_transferId The transferId of the transfer.* @param_data The data to be swapped. *@param_amount The amount to be swapped.*@param_asset The incoming asset to be swapped. */ function_prepare( bytes32_transferId, bytesmemory_data, uint256_amount, address_asset )internaloverridereturns(bytesmemory) { //highlight-start (address_swapper,address_toAsset,bytesmemory_swapData,bytesmemory_forwardCallData)=abi.decode( _data, (address,address,bytes,bytes) ); //highlight-end

uint256_amountOut=this.exactSwap(_swapper,_amount,_asset,_toAsset,_swapData);

returnabi.encode(_forwardCallData,_amountOut,_asset,_toAsset,_transferId); } }

```

Notice that the_prepare function expects abytes memory _data parameter that will be decoded into:

- _swapper
- : The specific swapper that will be used for the swap on the destination domain.
- _toAsset
- : The asset that should be swapped into.
- _swapData
- : The encoded swap data that will be constructed offchain (using theChain Abstraction SDK
- in the next section).
- _forwardCallData
- : The forward call data that the receiver contract will call on the destination domain.
-

Adapter Contract

The adapter inheritsSwapForwarderXReceiver and implements the_forwardFunctionCall method.

For example, let's take a look at theGreeter from theChain Abstraction Reference repo and pretend that this is the existing contract for a protocol that wants to use the chain abstraction pattern.

```

Copy pragmasolidity^0.8.19;

import{IERC20}from"@openzeppelin/contracts/token/ERC20/IERC20.sol";

interfaceIGreeter{ functiongreetWithTokens(address_token,uint256_amount,stringcalldata_greeting)external; }

contractGreeterisIGreeter{ stringpublicgreeting; IERC20publicWETH;

eventGreetingUpdated(string_greeting);

constructor(address_WETH) { WETH=IERC20(_WETH); }

functiongreetWithTokens(address_token,uint256_amount,stringcalldata_greeting)externaloverride{ IERC20 token=IERC20(_token);

require(_token==address(WETH),"Token must be WETH"); require(_amount>0,"Amount cannot be zero"); require( token.allowance(msg.sender,address(this))>=_amount, "User must approve amount" );

token.transferFrom(msg.sender,address(this),_amount);

greeting=_greeting; emitGreetingUpdated(_greeting); } }

```

The "target function" we want to call cross-chain isgreetWithTokens which takes payment in any amount of WETH to update the greeting.

SoGreeterAdapter below is the adapter contract that needs to be created:

```

Copy pragmasolidity^0.8.19;

import{IERC20}from"@openzeppelin/contracts/token/ERC20/IERC20.sol"; import {SwapForwarderXReceiver} from "lib/chain-abstraction-integration/contracts/destination/xreceivers/Swap/SwapForwarderXReceiver.sol"; import{IGreeter}from"./Greeter.sol";

contractGreeterAdapterisSwapForwarderXReceiver{ IGreeterpublicimmutablegreeter;

constructor(address_connext,address_greeter)SwapForwarderXReceiver(_connext) { greeter=IGreeter(_greeter); }

function_greetWithTokens(address_token,uint256_amount,stringmemory_greeting)internal{ IERC20 token=IERC20(_token); token.approve(address(greeter),_amount); greeter.greetWithTokens(_token,_amount,_greeting); }

function_forwardFunctionCall( bytesmemory_preparedData, bytes32/*_transferId*/, uint256/*_amount*/, address/*_asset*/ )internaloverridereturns(bool) { (bytesmemory_forwardCallData,uint256_amountOut,,)=abi.decode( _preparedData, (bytes,uint256,address,address) ); (address_token,stringmemory_greeting)=abi.decode(_forwardCallData,(address,string));

// Forward the call _greetWithTokens(_token,_amountOut,_greeting);

returntrue; } }

```

The_forwardFunctionCall function unwraps data which includes arguments for the forward call (greetWithTokens) and the amount of tokens received after the swap. It then "forwards" the call to thegreeter contract.

That's it!

Origin Domain Transaction

The second part of the integration is to set up the transaction that users send on the origin domain. This involves a few steps but Connext'sChain Abstraction SDK makes this process easier:

1. UsegetPoolFeeForUniV3
2. to fetch thepoolFee
3. for a specific swap route on the destination domain
4. Construct theforwardCallData
5. for the integrating protocol's "target function"
6. UsegetXCallCallData
7. to construct thecalldata
8. that will be passed intoxcall

9. UseprepareSwapAndXCall
10. to combine the swap and thexcall
11. into a single transaction request
12.

With Connext'sCore SDK :

1. UseestimateRelayerFee
2. to fetch therelayerFee
3. for thexcall
4. UsecalculateAmountReceived
5. to show users the expected amount received on the destination domain
6.

Installation

For installing the SDKs, useNode.js v18 .

```

Copy npminstall@connext/chain-abstraction@connext/sdk

```

1) Get the pool fee

First we need to retrieve inputs for the destination swap. The functiongetPoolFeeForUniV3 returns the poolFee of the UniV3 pool for a given token pair which will be used in the UniV3 router execution. The poolFee is the fee that is charged by the pool for trading tokens.

```

Copy exportconstgetPoolFeeForUniV3=async( domainId:string, rpc:string, token0:string, token1:string, ):

```

The function takes four parameters:

- domainId
- : The target domain ID.
- rpc
- : The RPC endpoint for a given domain.
- token0
- : The first token address.
- token1
- : The second token address.
-

The function returns aPromise that resolves to a string representing the poolFee of the UniV3 pool.

Example

```

Copy // asset address constPOLYGON_WETH="0x7ceB23fD6bC0adD59E62ac25578270cFf1b9f619";
constPOLYGON_USDC="0x2791bca1f2de4661ed88a30c99a7a9449aa84174"; // Domain details
constPOLYGON_DOMAIN_ID="1886350457"; constPOLYGON_RPC_URL="https://polygon.llamarpc.com";

constpoolFee=awaitgetPoolFeeForUniV3(POLYGON_DOMAIN_ID,POLYGON_RPC_URL,POLYGON_WETH,POLYGON_USDC);

```

2) Encode the forward call

This step depends on the target function. In ourGreeter example, the encoded calldata is quite simple:

```

Copy constforwardCallData=utils.defaultAbiCoder.encode( ["address","string"], [POLYGON_WETH,"hello world"], );

```

3) Construct the xcall

ThegetXCallCallData function generates calldata to be passed intoxcall . This combines the destination swap and the forward call.

```

Copy exportconstgetXCallCallData=async( domainId:string, swapper:Swapper, forwardCallData:string, params:DestinationCallDataParams, )

```

It takes four parameters.

- domainId
- : A string representing the destination domain ID.
- swapper
- : A string representing which swapper should be used. It can beUniV2
- ,UniV3
- , orOneInch
- .
- forwardCallData
- : encoded data for passing into the target contract usingabiencoder
- .
- params
- : An object containing the following fields.
- ```
- Copy
- {
- fallback:string;
- swapForwarderData:{
- toAsset:string;
- swapData:{
- amountOutMin:string;
- }|{
- amountOutMin:string;
- poolFee:string;
- };
- forwardCallData:{
- cTokenAddress:string;
- underlying:string;
- minter:string;
- }|{}|{};
- }
- }
- ```
- fallback
- : The fallback address to send funds to if the forward call fails on the destination domain.
- swapForwarderData
- : An object with the following fields.
-
    - toAsset
-
    - : Address of the token to swap into on the destination domain.
-
    - swapData
-
    - : Calldata that the swapper contract on the destination domain will use to perform the swap.
-
    - forwardCallData
-
    - : Calldata that the xReceive target on the destination domain will use in the forward call.
- *
-

The function returns the encoded calldata as a string.

Example

```

Copy constparams:DestinationCallDataParams={ fallback:USER_ADDRESSas0x{string}, swapForwarderData:{ toAsset:POLYGON_WETH, swapData:{ amountOutMin:"0", poolFee, }, }, };

constxCallData=awaitconnextService.getXCallCallDataHelper( destinationDomain, forwardCallData, params, );

```

4) Prepare swap and xcall

TheprepareSwapAndXCall function constructs theTransactionRequest that contains the origin swap andxcall .

```

Copy exportconstprepareSwapAndXCall=async( signerAddress:string, params:SwapAndXCallParams, ):

```

It takes two parameters:

- signerAddress
- (required): The address of the signer to send a transaction from.
- params
- : An object containing the following fields:
-
    - originDomain
-
    - (required): The origin domain ID.
-
    - destinationDomain
-
    - (required): The destination domain ID.
-
    - fromAsset
-
    - (required): The address of the asset to swap from.
-
    - toAsset
-
    - (required): The address of the asset to swap to.
-
    - amountIn
-
    - (required): The number of fromAsset tokens.
-
    - to
-
    - (required): The address to send the asset and call with the calldata on the destination.
-
    - delegate
-
    - (optional): The fallback address on the destination domain which defaults toto
-
    - .
-
    - slippage
-
    - (optional): Maximum acceptable slippage in BPS which defaults to 300. For example, a value of 300 means 3% slippage.
-
    - route
-
    - (optional): The address of the swapper contract and the data to call the swapper contract with.
-
    - callData
-
    - (optional): The calldata to execute (can be empty: "0x").
-
    - relayerFeeInNativeAsset
-
    - (optional): The fee amount in native asset.
-
    - relayerFeeInTransactingAsset
-
    - (optional): The fee amount in the transacting asset.
- *
-

The function returns a Promise that resolves to aTransactionRequest object to be sent to the RPC provider.

Example

```

Copy constswapAndXCallParams={ originDomain:"1886350457", destinationDomain:"6450786", fromAsset:OPTIMISM_WETH toAsset:OPTIMISM_USDC, amountIn:"1000000000000000000";// 1 WETH to: signerAddress, relayerFeeInTransactingAsset:"100000",// 0.1 USDC };

consttxRequest=awaitprepareSwapAndXCall(swapAndXCallParams,signerAddress);

```

5) Estimate relayer fee

Relayer fees must be paid by the user initiatingxcall . The Core SDK exposes anestimateRelayerFee function (see Estimating Fees) that returns an estimate given current gas prices on the origin and destination domains.

6) Estimate amount received

Showing an estimate of the final amount to be used in the destination target function can be informative for users. ThegetEstimateAmountReceived function returns this estimate which accounts for slippage from the origin and destination swaps as well as the bridge operation itself.

Previous Chain Abstraction Next xGovernance Last updated9 months ago On this page *Adapter Contract * Installation * SwapForwarderXReceiver * Adapter Contract * Origin Domain Transaction * Installation * 1) Get the pool fee * 2) Encode the forward call * 3) Construct the xcall * 4) Prepare swap and xcall * 5) Estimate relayer fee * 6) Estimate amount received

Edit on GitHub