Tracing Deposits and Withdrawals

In this tutorial, you'll learn how to use the Optimism SDK(opens in a new tab) to trace a Standard Bridge deposit or withdrawal between L1 and L2. You'll specifically learn how to determine the status of a deposit or withdrawal and how to retrieve the transaction receipt for the executed transaction on L1 (for withdrawals) or L2 (for deposits).

Check out the tutorial on Bridging ERC-20 Tokens With the Optimism SDK to learn how to create deposits and withdrawals. You can also check out the tutorial on Viewing Deposits and Withdrawals by Address to find deposits and withdrawals to trace.

Dependencies

- node(opens in a new tab)
- pnpm(opens in a new tab)

Create a Demo Project

You're going to use the Optimism SDK for this tutorial. Since the Optimism SDK is a Node.js(opens in a new tab) library, you'll need to create a Node.js project to use it.

Make a Project Folder

mkdir

op-sample-project cd

op-sample-project

Initialize the Project

pnpm

init

Install the Optimism SDK

pnpm

add

@eth-optimism/sdk

Install ethers.js

pnpm

add

ethers@^5

Add RPC URLs to Your Environment

You'll be using thegetMessageReceipt function from the Optimism SDK during this tutorial. This function use event queries to retrieve the receipt for a deposit or withdrawal. Since this function uses large event queries, you'll need to use an RPC provider like Alchemy(opens in a new tab) that supports indexed event queries. Grab an L1 and L2 RPC URL for Sepolia and OP Sepolia, respectively.

export L1_RPC_URL = ...

Sepolia RPC URL

export L2_RPC_URL = ...

OP Sepolia RPC URL

The Optimism SDK may be updated in the future to use a different method of retrieving deposits and withdrawals under the hood that does not require indexed event queries. This tutorial will be updated to reflect those changes if and when they occur.

Start the Node REPL

You're going to use the Node REPL to interact with the Optimism SDK. To start the Node REPL run the following command in your terminal:

node This will bring up a Node REPL prompt that allows you to run javascript code.

Import Dependencies

You need to import some dependencies into your Node REPL session.

Import the Optimism SDK

const

optimism

=

require ("@eth-optimism/sdk")

Import ethers.js

const

ethers

=

require ("ethers")

Set Session Variables

You'll need a few variables throughout this tutorial. Let's set those up now.

Import RPC URLs

const

l1RpcUrl

=

process . env . L1_RPC_URL const

l2RpcUrl

=

process . env . L2_RPC_URL

Set the deposit to trace

You'll be tracing a specific deposit in this tutorial. Deposit tracing is generally based on the transaction hash of the transaction that triggered the deposit. You can replace this transaction hash with your own if you'd like.

const

deposit

=

'0x5896d6e4a47b465e0d925723bab838c62ef53468139a5e9ba501efd70f90cccb'

Set the withdrawal to trace

You'll also be tracing a specific withdrawal in this tutorial. Like with deposits, withdrawal tracing is generally based on the transaction hash of the transaction that triggered the withdrawal. You can replace this transaction hash with your own if you'd like.

const

withdrawal

=

'0x18b8b4022b8d9e380fd89417a2e897adadf31e4f41ca17442870bf89ad024f42'

Create the RPC providers

const

l1Provider

=

new

ethers . providers .StaticJsonRpcProvider (l1RpcUrl) const

l2Provider

=

new

ethers . providers .StaticJsonRpcProvider (l2RpcUrl)

Create the CrossDomainMessenger

The Optimism SDK exports aCrossChainMessenger class that makes it easy to interact with the Standard Bridge contracts.

Create an instance of theCrossChainMessenger class:

const

messenger

=

new

optimism .CrossChainMessenger ({ l1ChainId :

11155111 ,

// 11155111 for Sepolia, 1 for Ethereum l2ChainId :

11155420 ,

// 11155420 for OP Sepolia, 10 for OP Mainnet l1SignerOrProvider : l1Provider , l2SignerOrProvider : l2Provider , })

Trace a Deposit

You can use theCrossChainMessenger instance to trace a deposit.

Get the deposit status

It's often useful to know the status of a deposit. Deposits can have a number of statuses depending on where the deposit is within its lifecycle. Refer to theOptimism SDK documentation(opens in a new tab)for a full list of potential deposit statuses.

const

depositStatus

=

await

messenger .getMessageStatus (deposit) console .log (depositStatus)

Get the deposit transaction receipt

You can also use theCrossChainMessenger instance to get the transaction receipt for the L2 transaction that executes and completes the deposit. Note that this function will also return the message status of the deposit and is therefore an alternative to the previous function. The result of this function is an object with amessageStatus property and atransactionReceipt property that is identical to the result of theethers functiongetTransactionReceipt .

const

depositReceipt

=

await

messenger .getMessageReceipt (deposit) console .log (depositReceipt)

Parse deposit logs

Like any other transaction receipt, the deposit transaction receipt contains logs that can be parsed to get additional information about the deposit. You can iterate through these logs and parse them just as you would parse any other receipt. For instance, you may wish to search forMint events to determine the amount of tokens that were minted as a result of the deposit.

Get the deposit transaction

Once you have the transaction receipt, you can directly query for the actual L2 transaction that executed the deposit.

const

depositTx

=

await

l2Provider .getTransaction (depositReceipt . transactionReceipt .transactionHash) console .log (depositTx)

Trace a Withdrawal

You can also use theCrossChainMessenger instance to trace a withdrawal.

Get the withdrawal status

Like deposits, withdrawals can have a number of statuses depending on where the withdrawal is within its lifecycle. Refer to theOptimism SDK documentation(opens in a new tab)for a full list of potential withdrawal statuses.

const

withdrawalStatus

=

await

messenger .getMessageStatus (withdrawal) console .log (withdrawalStatus)

Get the withdrawal transaction receipt

You can also use theCrossChainMessenger instance to get the transaction receipt for the L1 transaction that executes and completes the withdrawal. The result of this function is an object with amessageStatus property and atransactionReceipt property that is identical to the result of theethers functiongetTransactionReceipt .

const

withdrawalReceipt

=

await

messenger .getMessageReceipt (withdrawal) console .log (withdrawalReceipt)

Parse withdrawal logs

Like any other transaction receipt, the withdrawal transaction receipt contains logs that can be parsed to get additional information about the withdrawal. You can iterate through these logs and parse them just as you would parse any other receipt. For instance, you may wish to search forBurn events to determine the amount of tokens that were burned as a result of the withdrawal.

Get the withdrawal transaction

Once you have the transaction receipt, you can directly query for the actual L1 transaction that executed the withdrawal.

const

withdrawalTx

=

await

l1Provider .getTransaction (withdrawalReceipt . transactionReceipt .transactionHash) console .log (withdrawalTx)

Viewing Deposits and Withdrawals by Address Estimating Transaction Costs