Introduction

The Arbitrum SDK is a powerful TypeScript library that streamlines interactions with Arbitrum networks. It offers robust tools for bridging tokens and passing messages between networks through an intuitive interface to the underlying smart contracts.

Key Features

- Token Bridging: Effortlessly bridge tokens between Ethereum and Arbitrum.
- Message Passing: Seamlessly pass messages across networks.
- Contracts Interface: Leverage a strongly-typed interface for interacting with smart contracts.

Below is an overview of the Arbitrum SDK functionality. See the tutorials for more examples.

Getting Started

Install dependencies

- npm
- yarn
- pnpm

npm

install @arbitrum/sdk yarn

add @arbitrum/sdk pnpm

install @arbitrum/sdk

Using the Arbitrum SDK

Bridging assets

Arbitrum SDK can be used to bridge assets to or from an Arbitrum Network. The following asset bridgers are currently available:

- EthBridger
- Erc20Bridger

All asset bridgers have the following methods which accept different parameters depending on the asset bridger type:

- deposit
-
    - moves assets from the Parent to the Child chain
- withdraw
-
    - moves assets from the Child to the Parent chain

Example ETH Deposit to Arbitrum One

import

{ getArbitrumNetwork , EthBridger }

from

'@arbitrum/sdk'

// get the @arbitrum/sdk ArbitrumNetwork object using the chain id of the Arbitrum One chain const childNetwork =

await

getArbitrumNetwork (42161) const ethBridger =

new

EthBridger (childNetwork)

const ethDepositTxResponse =

await ethBridger . deposit ({ amount : utils . parseEther ('23') , parentSigner ,

// an ethers v5 signer connected to mainnet ethereum childProvider ,

// an ethers v5 provider connected to Arbitrum One })

const ethDepositTxReceipt =

await ethDepositTxResponse . wait ( )Learn more in the Eth Deposit tutorial

Example ETH Withdrawal from Arbitrum One

import

{ getArbitrumNetwork , EthBridger }

from

'@arbitrum/sdk'

// get the @arbitrum/sdk ArbitrumNetwork object using the chain id of the Arbitrum One chain const childNetwork =

await

getArbitrumNetwork (42161) const ethBridger =

new

EthBridger (childNetwork)

const withdrawTx =

await ethBridger . withdraw ({ amount : utils . parseEther ('23') , childSigner ,

// an ethers v5 signer connected to Arbitrum One destinationAddress : childWallet . address , }) const withdrawRec =

await withdrawTx . wait ( )Learn more in the Eth Withdraw tutorial

Networks

Arbitrum SDK comes pre-configured for Mainnet and Sepolia, and their Arbitrum counterparts. Any other networks that are not pre-configuredmust be registered before being used.

Configuring Network

To interact with a customArbitrumNetwork , you can register it using theregisterCustomArbitrumNetwork function.

import

{ registerCustomArbitrumNetwork }

from

'@arbitrum/sdk'

registerCustomArbitrumNetwork ({ chainID :

123456 , name :

'Custom Arbitrum Network' , })

Cross chain messages

When assets are moved by the Parent and Child cross chain messages are sent. The lifecycles of these messages are encapsulated in the classesParentToChildMessage andChildToParentMessage . These objects are commonly created from the receipts of transactions that send cross chain messages. A cross chain message will eventually result in a transaction being executed on the destination chain, and these message classes provide the ability to wait for that finalizing transaction to occur.

Redeem a Parent-to-Child Message

import

{ ParentTransactionReceipt , ParentToChildMessageStatus , }

from

'@arbitrum/sdk'

const parentTxnReceipt =

new

ParentTransactionReceipt (txnReceipt // ethers-js TransactionReceipt of an ethereum tx that triggered a Parent-to-Child message (say depositing a token via a bridge))

const parentToChildMessage =

(await parentTxnReceipt . getParentToChildMessages (childSigner // connected ethers-js Wallet)) [0]

const res =

await parentToChildMessage . waitForStatus ( )

if

(res . status === ParentToChildMessageStatus . Child)

{ // Message wasn't auto-redeemed; redeem it now: const response =

await parentToChildMessage . redeem ( ) const receipt =

await response . wait ( ) }

else

if

(res . status === ParentToChildMessageStatus . REDEEMED)

{ // Message successfully redeemed }Learn more in the Redeem Failed Retryable Tickets tutorial

Inbox Tools

As part of normal operation, the Arbitrum sequencer will send messages into the rollup chain. However, if the sequencer is unavailable and not posting batches, the inbox tools can be used to force the inclusion of transactions into the Arbitrum network.

Here's how you can use the inbox tools to withdraw ether from Arbitrum One without waiting for the sequencer:

const childNetwork =

await

getArbitrumNetwork (await childWallet . getChainId ( ))

const inboxSdk =

new

InboxTools (parentWallet , childNetwork) const arbSys = ArbSys__factory . connect (ARB_SYS_ADDRESS , childProvider) const arbSysIface = arbSys . interface const childCalldata = arbSysIface . encodeFunctionData ('withdrawEth' ,

[parentWallet . address ,])

const txChildRequest =

{ data : childCalldata , to :

ARB_SYS_ADDRESS , value :

1 , }

const childSignedTx =

await inboxSdk . signChildTx (txChildRequest , childWallet) const childTxhash = ethers . utils . parseTransaction (childSignedTx) . hash const resultsParent =

await inboxSdk . sendChildSignedTx (childSignedTx)

const inboxRec =

await resultsParent . wait ( )Learn more in the Delayed Inbox tutorial.

Utils

- EventFetcher
  - A utility to provide typing for the fetching of events
- MultiCaller
  - A utility for executing multiple calls as part of a single RPC request. This can be useful for reducing round trips.
- constants
  - A list of useful Arbitrum related constants

Development

Run Integration tests

1. Copy the.env-sample
2. file to.env
3. and update the values with your own.
4. First, make sure you have aNitro test node
5. running. Follow the instructionshere
6. .
7. After the node has started up (that could take up to 20-30 mins), runyarn gen:network
8. .
9. Once done, finally runyarn test:integration
10. to run the integration tests.

Defaults toArbitrum Sepolia , for custom network use--network flag.

Arbitrum Sepolia expects env varARB_KEY to be prefunded with at least 0.02 ETH, and env varINFURA_KEY to be set. (seeintegration_test/config.ts) Edit this page Previous Troubleshooting Next Migrating from v3 to v4