Bridge tokens via Arbitrum's standard ERC20 gateway

PUBLIC PREVIEW DOCUMENT This document is currently inpublic preview and may change significantly as feedback is captured from readers like you. Click theRequest an update button at the top of this document orjoin the Arbitrum Discord to share your feedback. In this how-to you'll learn how to bridge your own token between Ethereum (Layer 1 or L1) and Arbitrum (Layer 2 or L2), usingArbitrum's standard ERC20 gateway . For alternative ways of bridging tokens, don't forget to check out thisoverview .

Familiarity withArbitrum's token bridge system , smart contracts, and blockchain development is expected. If you're new to blockchain development, consider reviewing ourQuickstart: Build a dApp with Arbitrum (Solidity, Hardhat)before proceeding. We will useArbitrum's SDK throughout this how-to, although no prior knowledge is required.

We will go through all steps involved in the process. However, if you want to jump straight to the code, we have created this script in our tutorials repository that encapsulates the entire process.

Step 1: Create a token and deploy it on L1

We'll begin the process by creating and deploying on L1 a sample token to bridge. If you already have a token contract on L1, you don't need to perform this step.

We first create a standard ERC20 contract using OpenZeppelin's implementation. We make only 1 adjustment to that implementation, for simplicity, although it is not required: we specify aninitialSupply to be pre-minted and sent to the deployer address upon creation.

// SPDX-License-Identifier: MIT pragma

solidity

^ 0.8.0 ;

import

"@openzeppelin/contracts/token/ERC20/ERC20.sol" ;

contract

DappToken

is ERC20 { /* * @dev See {ERC20-constructor}. * * An initial supply amount is passed, which is preminted to the deployer. constructor (uint256 _initialSupply)

ERC20 ("Dapp Token" ,

"DAPP")

{ _mint (msg . sender , _initialSupply *

10

**

decimals ( )) ; } } We now deploy that token to L1.

const

{ ethers }

=

require ('hardhat') ; const

{ providers ,

Wallet

}

=

require ('ethers') ; require ('dotenv') . config ( ) ; const walletPrivateKey = process . env . DEVNET_PRIVKEY ; const l1Provider =

new

providers . JsonRpcProvider (process . env . L1RPC) ; const l1Wallet =

new

Wallet (walletPrivateKey , l1Provider) ;

/* * *For the purpose of our tests, here we deploy an standard ERC20 token (DappToken) to L1 * It sends its deployer (us) the initial supply of 1000* / const

main

=

async

( )

=>

{ console . log ('Deploying the test DappToken to L1:') ; const

L1DappToken

=

await

(await ethers . getContractFactory ('DappToken')) . connect (l1Wallet) ; const l1DappToken =

await

L1DappToken . deploy (1000) ;

await l1DappToken . deployed ( ) ; console . log (DappToken is deployed to L1 at { l1DappToken . address }) ;

/* * *Get the deployer token balance*/ const tokenBalance =

await l1DappToken . balanceOf (l1Wallet . address) ; console . log (nitial token balance of deployer: { tokenBalance }) ; } ;

main ( ) . then (( )

=> process . exit (0)) . catch ((error)

=>

{ console . error (error) ; process . exit (1) ; }) ;

Step 2: Identify the bridge contracts to call (concepts summary)

As stated in thetoken bridge conceptual page , when using Arbitrum's standard ERC20 gateway, you don't need to do any pre-configuration process. Your token will be "bridgeable" out of the box.

As explained in the conceptual page, there are 2 contracts that we need to be aware of when bridging tokens:

- Router contract
- : this is the contract that we'll interact with. It keeps a mapping of the gateway contracts assigned to each token, fallbacking to a default gateway for standard ERC20 tokens.
- Gateway contract
- : this is the contract that escrows or burns the tokens in the layer of origin, and sends the message over to the counterpart layer to mint or release the tokens there.

For simplicity, in this how-to we'll focus on the first case: bridging from Ethereum (L1) to Arbitrum (L2).

We'll explain below what specific contracts and methods need to be called to bridge your token, but you can abstract this whole process of finding the right addresses by using Arbitrum's SDK. You can use thedeposit function of theERC20Bridger class to bridge your tokens, which will use the appropriate router contract based on the network you're connected to, and will relay the request to the appropriate gateway contract. You can also use the functiongetL1GatewayAddress to get the address of the gateway contract that's going to be used. But don't worry about any of this yet, we'll use those functions in the next steps.

Now, here's an explanation of the contracts and methods that need to be called to manually bridge your token:

- When bridging from Ethereum (L1) to Arbitrum (L2), you'll need to interact with theL1GatewayRouter
- contract, by calling theoutboundTransferCustomRefund
- method. This router contract will relay your request to the appropriate gateway contract, in this case, theL1ERC20Gateway
- contract. To get the address of the gateway contract that's going to be used, you can call thegetGateway
- function in theL1GatewayRouter
- contract.
- When bridging from Arbitrum (L2) to Ethereum (L1), you'll need to interact with theL2GatewayRouter
- contract, by calling theoutBoundTransfer
- method. This router contract will relay your request to the appropriate gateway contract, in this case, theL2ERC20Gateway
- contract. To get the address of the gateway contract that's going to be used, you can call thegetGateway
- function in theL2GatewayRouter
- contract.

You can find the addresses of the contracts involved in the process inthis page .

Step 3: Approve token allowance for the gateway contract

The gateway contract will be the one that will transfer the tokens to be bridged over. So the next step is to allow the gateway contract to do so.

We typically do that by using theapprove method of the token, but you can use Arbitrum's SDK to abstract this process, by calling the methodapproveToken of theERC20Bridger class, which will call the approve method of the token passed by parameter, and set the allowance to the appropriate gateway contract.

/* * *Use l2Network to create an Arbitrum SDK Erc20Bridger instance * We'll use Erc20Bridger for its convenience methods around transferring token to L2* / const l2Network =

await

getL2Network (l2Provider) ; const erc20Bridge =

new

Erc20Bridger (l2Network) ;

/* * *The Standard Gateway contract will ultimately be making the token transfer call; thus, that's the contract we need to approve. * erc20Bridger.approveToken handles this approval * Arguments required are: * (1) l1Signer: The L1 address transferring token to L2 * (2) erc20L1Address: L1 address of the ERC20 token to be depositted to L2* / console . log ('Approving:') ; const l1Erc20Address = l1DappToken . address ; const approveTx =

await erc20Bridger . approveToken ({ l1Signer : l1Wallet , erc20L1Address : l1Erc20Address , }) ;

const approveRec =

await approveTx . wait ( ) ; console . log (You successfully allowed the Arbitrum Bridge to spend DappToken { approveRec . transactionHash } ,) ; As mentioned before, you can also call theapprove method of the token and send as a parameter the address of the gateway contract, which you can find by calling the methodgetGateway function in the router contract.

Step 4: Start the bridging process through the router contract

After allowing the gateway contract to transfer the tokens, we can now start the bridging process.

You can use Arbitrum's SDK to abstract this process, by calling the methoddeposit of theERC20Bridger class, which will estimate the gas parameters (_maxGas,_gasPriceBid and maxSubmissionCost, explained below) and call theoutboundTransferCustomRefund method of the router contract. You will only need to specify the following parameters:

- amount
- : Amount of tokens to bridge
- erc20L1Address
- : L1 address of the ERC20 token being bridged
- l1Signer
- : Signer object of the account transferring the tokens, connected to the L1 network
- l2Provider
- : Provider connected to the L2 network

/* * *Deposit DappToken to L2 using erc20Bridger. This will escrow funds in the Gateway contract on L1, and send a message to mint tokens on L2. * The erc20Bridge.deposit method handles computing the necessary fees for automatic-execution of retryable tickets — maxSubmission cost & l2 gas price * gas — and will automatically forward the fees to L2 as callvalue * Also note that since this is the first DappToken deposit onto L2, a standard Arb ERC20 contract will automatically be deployed. * Arguments required are: * (1) amount: The amount of tokens to be transferred to L2 * (2) erc20L1Address: L1 address of the ERC20 token to be depositted to L2 * (2) l1Signer: The L1 address transferring token to L2 * (3) l2Provider: An l2 provider* / const depositTx =

await erc20Bridger . deposit ({ amount : tokenDepositAmount , erc20L1Address : l1Erc20Address , l1Signer : l1Wallet , l2Provider : l2Provider , }) ; As mentioned before, you can also call the methodoutboundTransferCustomRefund manually in the router contract and specify the following parameters:

- address_token
- : L1 address of the ERC20 token being bridged
- address_refundTo
- : Account to be credited with the excess gas refund in L2
- address_to
- : Account to be credited with the tokens in L2
- uint256_amount
- : Amount of tokens to bridge
- uint256_maxGas
- : Max gas deducted from user's L2 balance to cover the execution in L2
- uint256_gasPriceBid
- : Gas price for the execution in L2
- bytes_data
- : 2 pieces of data encoded:* uint256maxSubmissionCost
  - : Max gas deducted from user's L2 balance to cover base submission fee
  - bytesextraData
  - : "0x"

Step 5: Wait for execution on L2

After calling the deposit method (or theoutboundTransferCustomRefund , if you're choosing the manual way), you'll have to wait a bit until the message is executed on L2. We will verify the status of the underlying retryable ticket created to bridge the tokens. Check this page, to know more aboutL1-to-L2 messages, also known as retryables.

You can programmatically wait for the execution of the transaction on L2 using Arbitrum's SDK. You should first wait for the execution of the submission transaction (the one sent to the router contract) and then the execution of the L2 transaction.

/* * *Now we wait for L1 and L2 side of transactions to be confirmed* const depositRec =

await depositTx . wait ( ) ; const l2Result =

await depositRec . waitForL2 (l2Provider) ;

/* * *The complete boolean tells us if the l1 to l2 message was successful* l2Result . complete ?

console . log (L2 message successful: status: { L1ToL2MessageStatus [l2Result . status] }) :

console . log (L2 message failed: status { L1ToL2MessageStatus [l2Result . status] }) ; If you're going the manual way, you can verify if the message has been executed on L2 through theRetryables Dashboard . Just paste the hash of transaction submitted to the router contract and the tool will tell you whether it's been redeemed or not.

Step 6: Check the new token contract created on L2

Finally, let's find the token contract that has been created on L2.

Using Arbitrum's SDK, you can call methodgetL2ERC20Address of theERC20Bridger class, which will return the address of the token contract in L2 that corresponds to the L1 token contract sent as parameter.

/* * *Check if our l2Wallet DappToken balance has been updated correctly * To do so, we use erc20Bridge to get the l2Token address and contract* / const l2TokenAddress =

await erc20Bridger . getL2ERC20Address (l1Erc20Address , l1Provider) ; const l2Token = erc20Bridger . getL2TokenContract (l2Provider , l2TokenAddress) ; To do this operation manually, you can call

methodcalculateL2TokenAddress of the router contract.

If you visit that address inArbiscan , you'll notice that it is a copy of the contractStandardArbERC20 . This is the standard contract that is automatically created the first time a token that doesn't exist in Arbitrum is bridged.The token bridge conceptual page has more information about this contract.

Conclusion

After finishing this process, you'll now have a counterpart token contract automatically created on L2. You can bridge tokens between L1 and L2 using the original token contract on L1 and the standard created contract on L2, along with the router and gateway contracts from each layer.

Resources

1. Concept page: Token Bridge
2. Arbitrum SDK
3. Token bridge contract addresses Edit this page Last updatedonMar 26, 2024 Previous Get started with token bridging Next Bridge tokens via Arbitrum's generic-custom gateway