Bridging Your Custom ERC-20 Token Using the Standard Bridge

This tutorial is for developers who want to bridge a new ERC-20 token to an OP Stack chain. If you want to bridge existing tokens, you can skip to the tutorial on Bridging ERC-20 tokens with the Optimism SDK. In this tutorial you'll learn how to bridge a custom ERC-20 token from Ethereum to an OP Stack chain using the Standard Bridge system. This tutorial is meant for developers who already have an existing ERC-20 token on Ethereum and want to create a bridged representation of that token on OP Mainnet.

This tutorial explains how you can create a custom token that conforms to the IOptimismMintableERC20 (opens in a new tab) interface so that it can be used with the Standard Bridge system. A custom token allows you to do things like trigger extra logic whenever a token is deposited. If you don't need extra functionality like this, consider following the tutorial on Bridging Your Standard ERC-20 Token Using the Standard Bridge instead.

The Standard Bridge does not support fee on transfer tokens (opens in a new tab) or rebasing tokens (opens in a new tab) because they can cause bridge accounting errors.

About OptimismMintableERC20s

The Standard Bridge system requires that L2 representations of L1 tokens implement the IOptimismMintableERC20 (opens in a new tab) interface. This interface is a superset of the standard ERC-20 interface and includes functions that allow the bridge to properly verify deposits/withdrawals and mint/burn tokens as needed. Your L2 token contract must implement this interface in order to be bridged using the Standard Bridge system. This tutorial will show you how to create a custom token that implements this interface.

Dependencies

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

Get ETH on Sepolia and OP Sepolia

This tutorial explains how to create a bridged ERC-20 token on OP Sepolia. You will need to get some ETH on both of these testnets.

You can use this faucet (opens in a new tab) to get ETH on Sepolia. You can use the Superchain Faucet (opens in a new tab) to get ETH on OP Sepolia.

Add OP Sepolia to Your Wallet

This tutorial uses Remix (opens in a new tab) to deploy contracts. You will need to add the OP Sepolia network to your wallet in order to follow this tutorial. You can use this website (opens in a new tab) to connect your wallet to OP Sepolia.

Get an L1 ERC-20 Token Address

You will need an L1 ERC-20 token for this tutorial. If you already have an L1 ERC-20 token deployed on Sepolia, you can skip this step. Otherwise, you can use the testing token located at 0x5589BB8228C07c4e15558875fAf2B859f678d129 (opens in a new tab) that includes a faucet() function that can be used to mint tokens.

Create an L2 ERC-20 Token

Once you have an L1 ERC-20 token, you can create a corresponding L2 ERC-20 token on OP Sepolia. This tutorial will use Remix (opens in a new tab) so you can easily deploy a token without a framework like Hardhat (opens in a new tab) or Foundry (opens in a new tab) . You can follow the same general process within your favorite framework if you prefer.

In this section, you'll be creating an ERC-20 token that can be deposited but cannot be withdrawn. This is just one example of the endless ways in which you could customize your L2 token.

Open Remix

Navigate to Remix (opens in a new tab) in your browser.

Create a new file

Click the ("Create new file") button to create a new empty Solidity file. You can name this file whatever you

Copy the example contract

Copy the following example contract into your new file:

```
// SPDX-License-Identifier: MIT pragma

solidity ^0.8.0;

import { ERC20 } from

"@openzeppelin/contracts/token/ERC20/ERC20.sol" ; import { IERC165 } from

"@openzeppelin/contracts/utils/introspection/IERC165.sol" ; import { IOptimismMintableERC20 } from

"https://github.com/ethereum-optimism/optimism/blob/v1.1.4/packages/contracts-
bedrock/src/universal/IOptimismMintableERC20.sol" ;

contract

MyCustomL2Token

is

IOptimismMintableERC20 , ERC20 { /// @notice Address of the corresponding version of this token on the remote chain.
address

public

immutable REMOTE_TOKEN;

/// @notice Address of the StandardBridge on this network. address

public

immutable BRIDGE;

/// @notice Emitted whenever tokens are minted for an account. /// @param account Address of the account tokens are
being minted for. /// @param amount Amount of tokens minted. event

Mint ( address

indexed account, uint256 amount);

/// @notice Emitted whenever tokens are burned from an account. /// @param account Address of the account tokens are
being burned from. /// @param amount Amount of tokens burned. event

Burn ( address

indexed account, uint256 amount);

/// @notice A modifier that only allows the bridge to call. modifier

onlyBridge () { require (msg.sender == BRIDGE ,

"MyCustomL2Token: only bridge can mint and burn" ); _; }

/// @param _bridge Address of the L2 standard bridge. /// @param _remoteToken Address of the corresponding L1 token.
/// @param _name ERC20 name. /// @param _symbol ERC20 symbol. constructor ( address

_bridge , address

_remoteToken , string

memory

_name , string

memory

_symbol ) ERC20 (_name, _symbol) { REMOTE_TOKEN = _remoteToken; BRIDGE = _bridge; }
```

/// @custom:legacy /// @notice Legacy getter for REMOTE_TOKEN. function

remoteToken () public

view

returns (address) { return REMOTE_TOKEN; }

/// @custom:legacy /// @notice Legacy getter for BRIDGE. function

bridge () public

view

returns (address) { return BRIDGE; }

/// @notice ERC165 interface check function. /// @param _interfaceId Interface ID to check. /// @return Whether or not the interface is supported by this contract. function

supportsInterface (bytes4

_interfaceId) external

pure

virtual

returns (bool) { bytes4 iface1 = type(IERC165).interfaceId; // Interface corresponding to the updated OptimismMintableERC20 (this contract). bytes4 iface2 = type(IOptimismMintableERC20).interfaceId; return _interfaceId == iface1 || _interfaceId == iface2; }

/// @notice Allows the StandardBridge on this network to mint tokens. /// @param _to Address to mint tokens to. /// @param _amount Amount of tokens to mint. function

mint (address

_to , uint256

_amount) external virtual override (IOptimismMintableERC20) onlyBridge { _mint (_to , _amount); emit

Mint (_to , _amount); }

/// @notice Prevents tokens from being withdrawn to L1. function

burn (address , uint256) external virtual override (IOptimismMintableERC20) onlyBridge { revert ("MyCustomL2Token cannot be withdrawn"); } }

Review the example contract

Take a moment to review the example contract. It's almost the same as the standardOptimismMintableERC20 (opens in a new tab) contract except that the_burn function has been made to always revert. Since the bridge needs to burn tokens when users want to withdraw them to L1, this means that users will not be able to withdraw tokens from this contract.

/// @notice Prevents tokens from being withdrawn to L1. function

burn (address , uint256) external virtual override (IOptimismMintableERC20) onlyBridge { revert ("MyCustomL2Token cannot be withdrawn"); }

Compile the contract

Save the file to automatically compile the contract. If you've disabled auto-compile, you'll need to manually compile the contract by clicking the "Solidity Compiler" tab (this looks like the letter "S") and press the blue "Compile" button.

Deploy the contract

Open the deployment tab (this looks like an Ethereum logo with an arrow pointing left). Make sure that your environment is set to "Injected Provider", your wallet is connected to OP Sepolia, and Remix has access to your wallet. Then, select theMyCustomL2Token contract from the deployment dropdown and deploy it with the following parameters:

_BRIDGE: "0x4200000000000000000000000000000000000007" _REMOTETOKEN: "" _NAME: "My Custom L2 Token" _SYMBOL: "MCL2T"

Bridge Some Tokens

Now that you have an L2 ERC-20 token, you can bridge some tokens from L1 to L2. Check out the tutorial on Bridging ERC-20 tokens with the Optimism SDK to learn how to bridge your L1 ERC-20 to L2s using the Optimism SDK. Remember that the withdrawal step willnot work for the token you just created! This is exactly what this tutorial was meant to demonstrate.

Add to the Superchain Token List

The Superchain Token List(opens in a new tab) is a common list of tokens deployed on chains within the Optimism Superchain. This list is used by services like the Optimism Bridge UI(opens in a new tab). If you want your OP Mainnet token to be included in this list, take a look at the review process and merge criteria(opens in a new tab).

Bridging ERC-20 Tokens With the Optimism SDK Bridging Your Standard ERC-20 Token to OP Mainnet