Setup & Installation

This tutorial builds on top of the project created in the previous tutorial. It can exist on its own, but for our sake it is much easier to utilize the existing L1 contracts that we already have in place.

If you don't have this, you can find the code for it in our dev-rels repo .

Uniswap contract

To interact with Uniswap we need to add its interface. In the root repo we created in the token bridge tutorial , run this:

cd packages/l1-contracts mkdir external &&

cd external touch ISwapRouter.sol InsideISwapRouter.sol paste this:

iswaprouter // SPDX-License-Identifier: GPL-2.0-or-later pragma

solidity

= 0.7.5 ; pragma abicoder v2 ;

/// @title Router token swapping functionality /// @notice Functions for swapping tokens via Uniswap V3 interface

ISwapRouter

{ struct

ExactInputSingleParams

{ address tokenIn ; address tokenOut ; uint24 fee ; address recipient ; uint256 deadline ; uint256 amountIn ; uint256 amountOutMinimum ; uint160 sqrtPriceLimitX96 ; }

/// @notice Swaps amountIn of one token for as much as possible of another token /// @param params The parameters necessary for the swap, encoded as ExactInputSingleParams in calldata /// @return amountOut The amount of the received token function

exactInputSingle (ExactInputSingleParams calldata params) external payable returns

(uint256 amountOut) ;

struct

ExactInputParams

{ bytes path ; address recipient ; uint256 deadline ; uint256 amountIn ; uint256 amountOutMinimum ; }

/// @notice Swaps amountIn of one token for as much as possible of another along the specified path /// @param params The parameters necessary for the multi-hop swap, encoded as ExactInputParams in calldata /// @return amountOut The amount of the received token function

exactInput (ExactInputParams calldata params) external payable returns

(uint256 amountOut) ;

struct

ExactOutputSingleParams

{ address tokenIn ; address tokenOut ; uint24 fee ; address recipient ; uint256 deadline ; uint256 amountOut ; uint256 amountInMaximum ; uint160 sqrtPriceLimitX96 ; }

/// @notice Swaps as little as possible of one token foramountOut of another token /// @param params The parameters necessary for the swap, encoded as ExactOutputSingleParams in calldata /// @return amountIn The amount of the input token function

exactOutputSingle (ExactOutputSingleParams calldata params) external payable returns

(uint256 amountIn) ;

struct

ExactOutputParams

{ bytes path ; address recipient ; uint256 deadline ; uint256 amountOut ; uint256 amountInMaximum ; }

/// @notice Swaps as little as possible of one token foramountOut of another along the specified path (reversed) /// @param params The parameters necessary for the multi-hop swap, encoded as ExactOutputParams in calldata /// @return amountIn The amount of the input token function

exactOutput (ExactOutputParams calldata params) external payable returns

(uint256 amountIn) ; }Source code: l1-contracts/test/external/ISwapRouter.sol#L1-L79 This is an interface for the Uniswap V3 Router, providing token swapping functionality. The contract defines methods for token swaps, both between two tokens or via a multi-hop path. Our portal will interact with the Uniswap V3 Router via this interface to perform token swaps on L1. We'll see more about this in the next step.

Create another Aztec project

Inaztec-contracts create a new Aztec contract project.

cd aztec-contracts && aztec-nargo new --contract uniswap Now youraztec-contracts will look like this:

aztec-contracts └── token_bridge ├── Nargo.toml ├── src ├── main.nr └── uniswap ├── Nargo.toml ├── src ├── main.nr Insideuniswap/Nargo.toml paste this in[dependencies] :

[dependencies] aztec = { git= "https://github.com/AztecProtocol/aztec-packages/" , tag= "aztec-packages-v0.28.1" , directory= "noir-projects/aztec-nr/aztec"

} authwit = { git= "https://github.com/AztecProtocol/aztec-packages/" , tag= "aztec-packages-v0.28.1" , directory= "noir-projects/aztec-nr/authwit" }

L2 contracts

Themain.nr will utilize a few helper functions that are outside the scope of this tutorial. Insideuniswap/src create two new files:

cd uniswap/src &&

touch util.nr &&

touch interfaces.nr Insideinterfaces.nr paste this:

interfaces use

dep :: aztec :: prelude :: { FunctionSelector ,

AztecAddress ,

EthAddress ,

PrivateContext } ; use

dep :: aztec :: context :: PublicContext ;

struct

Token

{ address :

AztecAddress , }

impl

Token

{ pub

fn

at (address :

AztecAddress)

->

Self

{ Self

{ address } }

pub

fn

transfer_public (self :

Self , context :

& mut

PublicContext , from :

AztecAddress , to :

AztecAddress , amount :

Field , nonce :

Field)

{ let _ = context . call_public_function (self . address , FunctionSelector :: from_signature ("transfer_public((Field),(Field),Field,Field)") , [from . to_field ( ) , to . to_field ( ) , amount , nonce]) ; }

pub

fn

unshield (self :

Self , context :

& mut

PrivateContext , from :

AztecAddress , to :

AztecAddress , amount :

Field , nonce :

Field)

{ let _ = context . call_private_function (self . address , FunctionSelector :: from_signature ("unshield((Field),(Field),Field,Field)") , [from . to_field ( ) , to . to_field ( ) , amount , nonce]) ; } }

struct

TokenBridge

{ address :

AztecAddress , }

impl

TokenBridge

{ pub

fn

at (address :

AztecAddress)

->

Self

{ Self

{ address } }

pub

fn

token (self :

Self , context :

& mut

PublicContext)

->

AztecAddress

{ let return_values = context . call_public_function (self . address , FunctionSelector :: from_signature ("get_token()") , [ ]) ; AztecAddress :: from_field (return_values [0]) }

pub

fn

exit_to_l1_public (self :

Self , context :

& mut

PublicContext , recipient :

EthAddress , amount :

Field , callerOnL1 :

EthAddress , nonce :

Field)

{ let _ = context . call_public_function (self . address , FunctionSelector :: from_signature ("exit_to_l1_public((Field),Field,(Field),Field)") , [recipient . to_field ( ) , amount , callerOnL1 . to_field ( ) , nonce]) ; } } Source code: noir-projects/noir-contracts/contracts/uniswap_contract/src/interfaces.nr#L1-L78 This creates interfaces for theToken contract andTokenBridge contract

- Token
- is a reference implementation for a token on Aztec. Here we just need two methods transfer_public
- andunshield()
- .
- TheTokenBridge
- facilitates interactions with ourbridge contract
- . Here we just need itsexit_to_l1_public

Run Aztec sandbox

You will need a running sandbox.

bash -i < (curl -s install.aztec.network) Then

aztec-sandbox In the next step, we will write the L1 Uniswap Portal.Edit this page

Previous Build an Aztec Connect-style Uniswap Next Uniswap Portal on L1