Solidity Code to Execute Swap on L1

To execute the swaps on L1, go back to theUniswapPortal.sol wecreated earlier inl1-contracts .

Under the struct, paste this code that will manage the public flow:

```
solidity_uniswap_swap_public /* * @notice Exit with funds from L2, perform swap on L1 and deposit output asset to L2 again publicly * @dev msg.value indicates fee to submit message to inbox. Currently, anyone can call this method on your behalf. * They could call it with 0 fee causing the sequencer to never include in the rollup. * In this case, you will have to cancel the message and then make the deposit later * @param _inputTokenPortal - The ethereum address of the input token portal * @param _inAmount - The amount of assets to swap (same amount as withdrawn from L2) * @param _uniswapFeeTier - The fee tier for the swap on UniswapV3 * @param _outputTokenPortal - The ethereum address of the output token portal * @param _amountOutMinimum - The minimum amount of output assets to receive from the swap (slippage protection) * @param _aztecRecipient - The aztec address to receive the output assets * @param _secretHashForL1ToL2Message - The hash of the secret consumable message. The hash should be 254 bits (so it can fit in a Field element) * @param _deadlineForL1ToL2Message - deadline for when the L1 to L2 message (to mint output assets in L2) must be consumed by * @param _canceller - The ethereum address that can cancel the deposit * @param _withCaller - When true, using msg.sender as the caller, otherwise address(0) * @return The entryKey of the deposit transaction in the Inbox / function

swapPublic ( address _inputTokenPortal , uint256 _inAmount , uint24 _uniswapFeeTier , address _outputTokenPortal , uint256 _amountOutMinimum , bytes32 _aztecRecipient , bytes32 _secretHashForL1ToL2Message , uint32 _deadlineForL1ToL2Message , address _canceller , bool _withCaller )

public

payable

returns

( bytes32 )

{ LocalSwapVars memory vars ;

vars . inputAsset =

TokenPortal ( _inputTokenPortal ) . underlying ( ) ; vars . outputAsset =

TokenPortal ( _outputTokenPortal ) . underlying ( ) ;

// Withdraw the input asset from the portal TokenPortal ( _inputTokenPortal ) . withdraw ( address ( this ) , _inAmount ,

true ) ; { // prevent stack too deep errors vars . contentHash = Hash . sha256ToField ( abi . encodeWithSignature ( "swap_public(address,uint256,uint24,address,uint256,bytes32,bytes32,uint32,address,address)" , _inputTokenPortal , _inAmount , _uniswapFeeTier , _outputTokenPortal , _amountOutMinimum , _aztecRecipient , _secretHashForL1ToL2Message , _deadlineForL1ToL2Message , _canceller , _withCaller ? msg . sender :

address ( 0 ) ) ) ; }

// Consume the message from the outbox registry . getOutbox ( ) . consume ( DataStructures . L2ToL1Msg ( { sender : DataStructures . L2Actor ( l2UniswapAddress ,

1 ) , recipient : DataStructures . L1Actor ( address ( this ) , block . chainid ) , content : vars . contentHash } ) ) ;

// Perform the swap ISwapRouter . ExactInputSingleParams memory swapParams ; { swapParams = ISwapRouter . ExactInputSingleParams ( { tokenIn :

address ( vars . inputAsset ) , tokenOut :

address ( vars . outputAsset ) , fee : _uniswapFeeTier , recipient :

address ( this ) , deadline : block . timestamp , amountIn : _inAmount , amountOutMinimum : _amountOutMinimum , sqrtPriceLimitX96 :

0 } ) ; } // Note, safeApprove was deprecated from Oz vars . inputAsset . approve ( address ( ROUTER ) , _inAmount ) ; uint256 amountOut = ROUTER . exactInputSingle ( swapParams ) ;

// approve the output token portal to take funds from this contract // Note, safeApprove was deprecated from Oz vars . outputAsset . approve ( address ( _outputTokenPortal ) , amountOut ) ;
```

// Deposit the output asset to the L2 via its portral return

TokenPortal (_outputTokenPortal) . depositToAztecPublic { value : msg . value } (_aztecRecipient , amountOut , _canceller , _deadlineForL1ToL2Message , _secretHashForL1ToL2Message) ; } Source code: l1-contracts/test/portals/UniswapPortal.sol#L40-L132 What’s happening here?

1. It fetches the input and output tokens we are swapping. The Uniswap portal only needs to know the portal addresses of the input and output as they store the underlying ERC20 token address.
2. Consumes thewithdraw
3. message to get input tokens on L1 to itself. This is needed to execute the swap.
4. Before it actually can swap, it checks if the provided swap parameters were what the user actually wanted by creating a message content hash (similar to what we did in the L2 contract) to ensure the right parameters are used.
5. Executes the swap and receives the output funds to itself.
6. The deadline by which the funds should be swapped isblock.timestamp
7. i.e. this block itself. This makes things atomic on the L1 side.
8. The portal must deposit the output funds back to L2 using the output token’s portal. For this we first approve the token portal to move Uniswap funds, and then call the portal’sdepositToAztecPublic()
9. method to transfer funds to the portal and create a L1 → l2 message to mint the right amount of output tokens on L2.
10. To incentivize the sequencer to pick up this message, we pass a fee to the deposit message.

This concludes the public flow.

In the next step, we will code a private flow in the Aztec.nr contractEdit this page

Previous Swapping Publicly Next Swapping Privately