StargateComposed.sol

Leverage Stargate to perform additional smart contract logic on the destination chain! To test Stargate Composability you can mint yourself testnet tokens using the addresses found on theTest Faucet page. This contract is an example of the power of composability. A contract on the source chain composes an AMM and Stargate, and a destination contract which has access to an AMM implementssgReceive() receives the tokens and a payload to perform additional logic.

swapNativeForNative()

The StargateComposed.sol contract can swap native on the source chain for native on the destination chain. To swap tokens to another chain using Stargate this contract must use theStargateComposer.sol to interact with Stargate. This code snippet shows how the contract uses it to swap native on the source chain for native on the destination chain.

Warning: When composing, Do Not swap() real funds to a contract address that does not implement sgReceive() or you*will* lose those funds. ```

Copy ///@paramdstChainId The message ordering nonce ///@parambridgeToken The remote Bridge address ///@paramsrcPoolId The token contract on the local chain ///@paramdstPoolId The qty of local _token contract tokens ///@paramnativeAmountIn The amount of native token coming in on source ///@paramto The address to send the destination tokens to ///@paramamountOutMin The minimum amount of stargatePoolId token to get out of amm router ///@paramamountOutMinSg The minimum amount of stargatePoolId token to get out on destination chain ///@paramamountOutMinDest The minimum amount of native token to receive on destination ///@paramdeadline The overall deadline ///@paramdestStargateComposed The destination contract address that must implement sgReceive() functionswapNativeForNative(uint16dstChainId, addressbridgeToken, uint16srcPoolId, uint16dstPoolId, uintnativeAmountIn, addressto, uintamountOutMin, uintamountOutMinSg, uintamountOutMinDest, uintdeadline, addressdestStargateComposed)

```

Use the AMM Router

Using the amm router, swap native into the Stargate pool token, sending the output token to this contract.

```

Copy // special token value that indicates the sgReceive() should swap OUT native asset address publicOUT_TO_NATIVE=0x0000000000000000000000000000000000000000;

uint bridgeAmount; // using the amm router, swap native into the Stargate pool token, sending the output token to this contract { // create path[] for amm swap address[] memory path=newaddress; path[0]=IUniswapV2Router02(ammRouter).WETH();// native IN requires that we specify the WETH in path[0] path[1]=bridgeToken;// the bridge token,

uint[] memory amounts=IUniswapV2Router02(ammRouter).swapExactETHForTokens{value:nativeAmountIn}(amountOutMin, path, address(this), deadline);

bridgeAmount=amounts[1]; require(bridgeAmount>0,'error: ammRouter gave us 0 tokens to swap() with stargate');

// this contract needs to approve the stargateComposer to spend its path[1] token! IERC20(bridgeToken).approve(address(stargateComposer),bridgeAmount); }

// encode payload data to send to destination contract, which it will handle with sgReceive() bytes memory data; { data=abi.encode(OUT_TO_NATIVE,deadline,amountOutMinDest,to); }

```

StargateComposer.swap()

Call StargateComposer swap() to send the tokens to the destination chain.

```

Copy IStargateRouter(stargateComposer).swap{value:msg.value.sub(nativeAmountIn)}(dstChainId, // the destination chain id srcPoolId, // the source Stargate poolId dstPoolId, // the destination Stargate poolId payable(msg.sender), // refund adddress. if msg.sender pays too much gas, return extra eth bridgeAmount, // total tokens to send to destination chain amountOutMinSg, // minimum IStargateRouter.lzTxObj(500000, 0, "0x"), // 500,000 for the sgReceive() abi.encodePacked(destStargateComposed), // destination address, the sgReceive() implementer data // bytes payload);

```
```

sgReceive()

StargateComposed.sol implementsIStargateReceiver so it can implement thesgReceive function to receive the tokens and payload.

```

Copy ///@param_chainId The remote chainId sending the tokens ///@param_srcAddress The address of the msg.sender who initiated the swap ///@param_nonce The message ordering nonce ///@param_token The token contract on the local chain ///@paramamountLD The qty of local _token contract tokens ///@param_payload The bytes containing the _tokenOut, _deadline, _amountOutMin, _toAddr functionsgReceive( uint16_chainId, bytesmemory_srcAddress, uint_nonce, address_token, uintamountLD, bytesmemorypayload )

```

Unpack payload, approve, get pre balance of toAddress

Unpack the payload to get_tokenOut ,_deadline ,_amountOutMin ,_toAddr . Approve the amm router so it can swap our tokens. Get the pre balance of the_toAddr to emit an event of exact amount sent.

```

Copy (address_tokenOut,uint_deadline,uint_amountOutMin,address_toAddr)=abi.decode(payload, (address,uint,uint,address));

IERC20(_token).approve(address(ammRouter),amountLD);

uint_toBalancePreTransferOut=address(_toAddr).balance;

```

If _tokenOut is the Zero Address they will get native token

Use the amm router to swap the incoming bridge token into native token

```

Copy if(_tokenOut==address(0x0)){ address[]memorypath=newaddress; path[0]=_token; path[1]=IUniswapV2Router02(ammRouter).WETH();

tryIUniswapV2Router02(ammRouter).swapExactTokensForETH( amountLD,// the stable received from stargate at the destination _amountOutMin,// slippage param, min amount native token out path,// path[0]: stabletoken address, path[1]: WETH from sushi router _toAddr,// the address to send the *out* native to _deadline// the unix timestamp deadline ) { emitReceivedOnDestination( OUT_TO_NATIVE, address(_toAddr).balance.sub(_toBalancePreTransferOut) ); }catch{ IERC20(_token).transfer(_toAddr,amountLD); emitReceivedOnDestination(_token,amountLD); } }

```

Else they will get ERC20 token

Use the amm router to swap the incoming bridge token into an ERC20 token

```

Copy else{ uint_toAddrTokenBalancePre=IERC20(_tokenOut).balanceOf(_toAddr); address[]memorypath=newaddress; path[0]=_token; path[1]=_tokenOut; tryIUniswapV2Router02(ammRouter).swapExactTokensForTokens( amountLD,// the stable received from stargate at the destination _amountOutMin,// slippage param, min amount native token out path,// path[0]: stabletoken address, path[1]: WETH from sushi router _toAddr,// the address to send the *out* tokens to _deadline// the unix timestamp deadline ) { emitReceivedOnDestination(_tokenOut,IERC20(_tokenOut).balanceOf(_toAddr).sub(_toAddrTokenBalancePre)); }catch{ IERC20(_token).transfer(_toAddr,amountLD); emitReceivedOnDestination(_token,amountLD); } }

```

Previous StargateComposer.sol Next Stargate Router Methods Last updated6 months ago On this page * swapNativeForNative() * Use the AMM Router * StargateComposer.swap() * sgReceive() * Unpack payload, approve, get pre balance of toAddress * If _tokenOut is the Zero Address they will get native token * Else they will get ERC20 token