Across+ Integration

Product Description

Across+ is a bridge abstraction framework. A developer can use Across+ in their dApp to bundle bridging and protocol actions in the same transaction, abstracting it away from the user. This can be used to promote user onboarding and defragment user liquidity across chains. Instead of users independently bridging assets to chains to use applications, Across+ enables your application to meet the user where they already are. At a high level, Across+ works through the following process: * End-user (or intermediate contract) includes a message field in the deposit transaction. * To be repaid for the fill, the relayer is required to include the same message field in their fill. * When this message field is non-empty and the recipient is a contract, the * SpokePool * calls a special handler function on the recipient contract with the message (and a few other fields). * This function can do anything, meaning application-specific actions can be executed atomically.At

Integrating Across+ into Your Application

This guide contains instructions and examples for calling the smart contract functions and constructing the message and creating the destination handler contract. If you have further questions or suggestions for this guide, please send a message to the

developer-questions

channel in the Across Discord .

Creating an Across+ Transaction

In this example we'll be walking through how to use Across+ to perform an AAVE deposit on behalf of the user on the destination chain.

Crafting the Message

Across+ requires that you send some nonempty message to your contract on the other side. This message allows you to pass arbitrary information to your recipient contract and it ensures that Across understands that you intend to trigger the handler function on the recipient (instead of just transferring tokens). A message is required if you want the handler to be called. In this example, our message will be just the user's address since that's all our contract would need to know to generate an AAVE deposit. Here's an example for generating this in typescript: // Ethers V6 import ethers from

"ethers" ; function

generateMessage (userAddress :

string)

{ const abiCoder = ethers . AbiCoder . defaultAbiCoder (); return abiCoder . encode (["address"],

[userAddress]); } Example in solidity: function

generateMessage (address userAddress)

returns

(bytes

memory)

{ return abi . encode (userAddress); }

Generating the Deposit

The deposit creation process is nearly identical to the process described for initiating a deposit (Initiating a Deposit (User Intent)). However, there are two tweaks to that process to include a message. 1. 1. 2. When getting a quote (3Getting a Quote 4.), two additional query parameters need to be added. 5. 1. 1. 6. 2. recipient 7. 3. : the recipient for the deposit. For Across+ transactions, this is not the end-user. It is the contract that implements the handler you would like to call (more on that later). Usually, this is a contract you have created to decode and handle the message. 8. 4. 2. 9. 5. message 10. 6. : the message you crafted above. 11. 2. 12. When calling deposit (13. Calling depositV3 14.), you'll need to make a slight tweak to the parameters. 15. 1. 1. 16. 2. The recipient should be set to your handler contract. 17. 3. 2. 18. 4. The message field should be set to the message you generated above instead of 19. 5. 0x 20. 6. .

Building a Handler Contract

You will need to implement a function matching the following interface in your handler contract to receive the message: function

handleV3AcrossMessage (address tokenSent , uint256 amount , address relayer , bytes

memory message)

external ; For this example, we're depositing the funds the user sent into AAVE on the user's behalf. Here's how that full contract implementation might look. Note: this contract has not been vetted whatsoever, so use this sample code at your own risk. import

{ IERC20 }

from

"openzeppelin-contracts/contracts/token/ERC20/IERC20.sol" ; import

{ SafeERC20 }

from

"openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol" ; interface

AavePool

{ function

deposit (address asset , uint256 amount , address onBehalfOf , uint16 referralCode)

external ; } contract

AaveDepositor

{ using

SafeERC20

for IERC20 ;

error Unauthorized (); address

public immutable aavePool ; uint16

public immutable referralCode ; address

public immutable acrossSpokePool ;

constructor (address _aavePool ,

uint16 _referralCode ,

address _acrossSpokePool)

{ aavePool = _aavePool ; referralCode = _referralCode ; acrossSpokePool = _acrossSpokePool ; }

function

handleV3AcrossMessage (address tokenSent , uint256 amount , address relayer ,

// relayer is unused bytes

memory message)

external

{ // Verify that this call came from the Across SpokePool. if

(msg . sender != acrossSpokePool)

revert

Unauthorized ();

// Decodes the user address from the message. address user = abi . decode (message ,

(address));

// Approve and deposit the funds into AAVE on behalf of the user. IERC20 (tokenSent). safeIncreaseAllowance (aavePool , amount); aavePool . deposit (tokenSent , amount , user , referralCode); } } One note on this implementation: this contract only uses the funds that are sent to it. That means that the message is assumed to only have authority over those funds and no outside funds. This is important because relayers can send invalid relays. They will not be repaid the funds sent in, but if an invalid message could unlock other funds, then a relayer could use this maliciously. You can find this interface definition in the codebase here .

Conclusion

Now that you have a process for constructing a message, creating a deposit transaction, and you have a handler contract built and deployed on the destination chain, all you need to do is send the deposit to have the handler get executed on the destination.

WrapChoice Example

Here's another contract example using a slightly different message format to allow users to choose whether they want to receive WETH or ETH. In this example, a bool is passed along with the address of the user to allow the message to define the unwrapping behavior on the destination. import

{ IERC20 }

from

"openzeppelin-contracts/contracts/token/ERC20/IERC20.sol" ; import

{ SafeERC20 }

from

"openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol" ; interface

WETHInterface

{ function

withdraw (uint wad)

external ; } contract

WrapChoice

{ using

SafeERC20

for IERC20 ; error Unauthorized (); error NotWETH (); error FailedETHTransfer () address

public immutable weth ; address

public immutable acrossSpokePool ;

constructor (address _weth ,

address _acrossSpokePool)

{ weth = _weth ; acrossSpokePool = _acrossSpokePool ; }

function

handleV3AcrossMessage (address tokenSent , uint256 amount , address relayer ,

// relayer is unused bytes

memory message)

external

{ if

(msg . sender != acrossSpokePool)

revert

Unauthorized (); if

(tokenSent != weth)

revert

NotWETH ();

(address

payable user ,

bool sendWETH)

= abi . decode (message ,

(address ,

bool));

if

(sendWETH)

{ // Transfer WETH directly to user address. IERC20 (weth). safeTransfer (user , amount); }

else

{ weth . withdraw (amount); // Transfer the funds to the user via low-level call. (bool success ,)

= user . call { value : amount }(""); if

(! success)

revert

FailedETHTransfer (); } }

// Required to receive ETH from the WETH contract. receive ()

external

payable

{} }

Summarized Requirements

* The deposit * message * is not empty * The * recipient * address is a contract on the * destinationChainId * that implements a public * handleV3AcrossMessage(address,uint256,address,bytes) * function. See * Reverting Transactions * for considerations. * Construct your * message * Use the Across API to get an estimate of the * relayerFeePct * you should set for your message and recipient combination * Call * depositV3() * passing in your message * Once the relayer calls * fillV3Relay() * on the destination, your recipient's * handleAcrossMessage * will be executed * The additional gas cost to execute the above function is compensated for in the deposit's * relayerFeePct * .

Security & Safety Considerations

* It is recommended that recipient contracts require that * handleV3AcrossMessage() * is only callable by the Across SpokePool contract on the same chain. * Avoid making unvalidated assumptions about the * message * data supplied to * handleV3AcrossMessage() * . Across+ does not guarantee message integrity, only that a relayer who spoofs a message will not be repaid by Across. If integrity is required, integrators should consider including a depositor signature in the message for additional verification. Message data should otherwise be treated as spoofable and untrusted for use beyond directing the funds passed along with it. * Avoid embedding assumptions about the transfer token or transfer amount into their messages. Instead use the * tokenSent * and * amount * variables supplied with to the * handleV3AcrossMessage() * function. These fields are enforced by the SpokePool contract and so can be assumed to be correct by the recipient contract. * In the event that a deposit expires, it will be refunded to the depositor address. Ensure that the depositor address on the origin SpokePool is capable of receiving refunds. * The relayer(s) able to complete a fill can be restricted by storing an approved set of addresses in a mapping and validating the * handleV3AcrossMessage() * relayer * parameter. This

implies a trust relationship with one or more relayers.

Reverting Transactions

* If the * message * specifies a transaction that could revert when handled on the destination, the user will be refunded on the origin chain in the bundle following the expiry of the deposit. * * Note: Expiry occurs when the destination SpokePool timestamp exceeds the deposit fillDeadline timestamp. * If this is not desirable behavior it is recommended to include logic in the handler contract to simply transfer the funds to the user in the case of a reverting transaction. Integration Guides - Previous Across Bridge Integration Next- Integration Guides Across Settlement Integration Last modified17d ago On this page Product Description Integrating Across+ into Your Application Creating an Across+ Transaction WrapChoice Example Summarized Requirements Security & Safety Considerations Reverting Transactions