Across Bridge Integration

Product Description

The Across Bridge is Across' first and most familiar product: an end-user application to move value between chains. It combines a reference implementation of Across' quoting engine, relayer network, and Across' modular intents settlement layer, Across Settlement . Across Bridge is an end-user product but can be integrated into other protocols to provide quoting and bridging directly in a 3rd party application. Bungee (Socket), Jumper (Li.Fi), Rango , and many others do this today.

Integrating Across Bridge into Your Application

This guide contains instructions and examples for calling the smart contract functions that would allow third party projects to transfer assets between Across supported chains. If you have further questions or suggestions for this guide, please send a message to the

developer-questions

channel in the Across Discord .

Initiating a Deposit (User Intent)

Deposits are initiated by interacting with contracts called SpokePools . There is one SpokePool deployed on each chain supported by Across. Each SpokePool has minor modifications to work with each chain, but maintains the same core interface and implementation. For example, on Ethereum the SpokePool contract is named Ethereum_SpokePool.sol , and on Optimism the contract is named Optimism_SpokePool.sol .

Getting a Quote

The process for initiating a deposit begins with determining the fee that needs to be paid to the relayer. To do this, you can use the suggested-fees endpoint: https://across.to/api/suggested-fees with the following query parameters: * originChainId: chainId where the user's deposit is originating * destinationChainId: chainId where the user intends to receive their funds * token: the address of the token that the user is depositing on the origin chain * amount: the raw amount the user is transferring. By raw amount, this means it should be represented exactly as it is in Solidity, meaning 1 USDC would be 1e6 or 1 ETH would be 1e18. Example for a user transferring 1000 USDC from Ethereum to Optimism: curl "https://across.to/api/suggested-fees?token=0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48&destinationChainId=10&amount=100000" There are two elements of the response that you'll need to create the deposit: * totalRelayFee.total * : this is the amount of the deposit that the user will need to pay to have it relayed. It is intended to capture all fees, including gas fees, relayer fees, and system fees. * timestamp * : this is the quote timestamp. Specifying this timestamp onchain ensures that the system fees don't shift under the user while the intent is in-flight. You can find details on the Across API here .

Checking Limits

You'll want to check the Across' limits to determine how long a relay is expected to take and to ensure that Across can process a deposit of this size. To do this, you can use the suggested-fees endpoint: https://across.to/api/limits endpoint with the following query parameters: * originChainId: chainId where the user's deposit is originating * destinationChainId: chainId where the user intends to receive their funds * token: the address of the token that the user is depositing on the origin chain Example for a user transferring USDC from Ethereum to Optimism: curl "https://across.to/api/limits?token=0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48&destinationChainId=10&originChainId=1" The response has three elements: 1. 1. 2. maxDepositInstant 3. : check this value first. if the user's amount is less than or equal to this amount, there is known to be enough relayer liquidity on the destination chain to fill them immediately. 4. 2. 5. maxDepositShortDelay 6. : if the user's deposit amount is larger than maxDepositInstant, check this value. If the user's amount is less than or equal to this amount, there is known to be enough relayer liquidity that can be moved to the destination chain within 30 minutes, so you should expect them to be filled within that time frame. 7. 3. 8. maxDeposit 9. : if the user's deposit amount is larger than maxDepositShortDelay, check this value. If the user's amount is less than or equal to this amount, there is enough liquidity in Across to fill them via a slow fill, which could take up to 3 hours. If the user's deposit amount is larger than this, Across cannot fulfil the user's intent. You can find details on the Across API here .

Calling depositV3

Before making the call, you'll need the SpokePool address. This can be retrieved from the suggested-fees response for convenience, but we suggest manually verifying these and hardcoding them per chain in your application for security. You can find the addresses here . Once you have the SpokePool address, you'll need to approve the SpokePool to spend tokens from the user's EOA or the contract that will be calling the SpokePool . The approval amount must be >= the amount value. If sending ETH, no approval is necessary. The deposit call can come from an intermediary contract or directly from the user's EOA. This is the function that needs to be called: function

depositV3 (address depositor , address recipient , address inputToken , address outputToken , uint256 inputAmount , uint256 outputAmount , uint256 destinationChainId , address exclusiveRelayer , uint32 quoteTimestamp , uint32 fillDeadline , uint32 exclusivityDeadline , bytes

calldata message)

external ; Here is an example of how the parameters could be populated in Javascript: spokePool . depositV3 (userAddress ,

// User's address on the origin chain. userAddress ,

// recipient. Whatever address the user wants to recieve the funds on the destination. usdcAddress ,

// inputToken. This is the usdc address on the originChain "0x0000000000000000000000000000000000000000" ,

// outputToken: 0 address means the output token and input token are the same. Today, no relayers support swapping so the relay will not be filled if this is set to anything other than 0x0. amount ,

// inputAmount amount . sub (BigNumber . from (totalRelayFee . total)),

// outputAmount: this is the amount - relay fees. totalRelayFee.total is the value returned by the suggested-fees API. destinationChainId ,

// destinationChainId "0x0000000000000000000000000000000000000000" ,

// exclusiveRelayer: set to 0x0 for typical integrations. timestamp ,

// quoteTimestamp: this should be set to the timestamp returned by the API. Math . round (Date . now ()

/

1000)

+

21600 ,

// fillDeadline: We reccomend a fill deadline of 6 hours out. The contract will reject this if it is beyond 8 hours from now. 0 ,

// exclusivityDeadline: since there's no exclusive relayer, set this to 0. "0x" ,

// message: empty message since this is just a simple transfer.) and in solidity: spokePool . depositV3 (userAddress ,

// User's address on the origin chain. userAddress ,

// recipient. Whatever address the user wants to recieve the funds on the destination. usdcAddress ,

// inputToken. This is the usdc address on the originChain address (0),

// outputToken: 0 address means the output token and input token are the same. Today, no relayers support swapping so the relay will not be filled if this is set to anything other than 0x0. amount ,

// inputAmount amount - totalRelayFee ,

// outputAmount: this is the amount - relay fees. totalRelayFee is the value returned by the suggested-fees API. destinationChainId ,

// destinationChainId address (0),

// exclusiveRelayer: set to 0x0 for typical integrations. timestamp ,

// timestamp: this should be the timestamp returned by the API. Otherwise, set to block.timestamp. block . timestamp +

21600 ,

// fillDeadline: We reccomend a fill deadline of 6 hours out. The contract will reject this if it is beyond 8 hours from now. 0 ,

// exclusivityDeadline: since there's no exclusive relayer, set this to 0. "" ,

// message: empty message since this is just a simple transfer.);

Speeding up Deposits

Relayer fees are a function of the spread between the outputAmount and inputAmount parameters of a depositV3 . If this spread is too low, it may not be profitable for relayers to fill. While a deposit is not filled, the outputAmount can be increased by calling speedUpDepositV3 , which requires a signature from the depositor address. Example coming soon!

Tracking Events

Developer Notes Concepts -Previous Canonical Asset Maximalism Next- Integration Guides Across+ Integration Last modified17d ago On this page Product Description Integrating Across Bridge into Your Application Initiating a Deposit (User Intent) Speeding up Deposits