Migration from V2 to V3

Find below important migration information for users of the Across API and the smart contracts (e.g. those who call the Across SpokePools directly to deposit or fill bridge transfers and those who track SpokePool events).

Introduction

Across V2 is migrating in-place to V3 and will support cross-chain token swaps and a more streamlined contract interface. This guide is intended to help developers who integrate with Across to prepare their codebases for the upgrade. Please note that this page is a work in progress, and more sections will be added to help with different types of integrations.

Summary of Breaking Changes

* Recipient contracts implementing * handleAcrossMessage * must now replace that with a * handleAcrossV3Message * call. * Details * . * Across V2 events including * FundsDeposited * will no longer be emitted. These will be replaced with V3 events. * Details * . * Relayers must now fill all deposits with a new function, * fillV3Relay() * . Details * . * Depositors who wish to speed up their deposits must now use * speedUpV3Deposit() * instead of * requestSpeedUpDeposit() * . * Details * .

Across API migration guide

Overview

Across v3 redesigns how fees are handled when creating deposits. In a nutshell, the calculation of fees will be simplified and replaced by inputAmount / outputAmount arguments. This will impact the response data of the API and also how to call the deposit function of a SpokePool contract. Note that these changes won't be breaking short-term but are actionable mid- to long-term.

Non-breaking changes

For a seamless upgrade, there will be no breaking changes for the existing v2 interfaces.

[Updated]

relayFeePct will now include lpFeePct In Across v2, the request GET /suggested-fees returned type

FeesResponse

=

{ // ... other fields capitalFeePct :

string ; capitalFeeTotal :

string ; relayGasFeePct :

string ; relayGasFeeTotal :

string ; relayFeePct :

string ;

// capitalFeePct + gasFeePct relayFeeTotal :

string ;

// capitalFeeTotal + gasFeeTotal lpFeePct :

string ; } It was then expected to call the method deposit of a SpokePool like const tx =

await spokePool . deposit (// ... other args feesResponse . relayFeePct // capitalFeePct + gasFeePct) The lpFeePct was not required as an argument and automatically derived based on the quoteTimestamp . But if you wanted to show the total bridge fee to the user, then you would have to sum them up like const totalBridgeFeePct = relayFeePct + lpFeePct In Across v3, we need to pass the lpFeePct as part of the total fee when calling the deposit function. In order to be backwards-compatible, the API now returns for GET /suggested-fees type FeesResponse = { // ... other fields capitalFeePct: string; capitalFeeTotal: string; relayGasFeePct: string; relayGasFeeTotal: string; - relayFeePct: string; // capitalFeePct + gasFeePct + relayFeePct: string; // capitalFeePct + gasFeePct + lpFeePct - relayFeeTotal: string; // capitalFeeTotal + gasFeeTotal + relayFeeTotal: string; // capitalFeeTotal + gasFeeTotal + lpFeeTotal - lpFeePct: string; + lpFeePct: "0"; } There are no

changes to the interface and therefore no changes are required for how you call the deposit function or calculate the total bridge fee. If you want to show the correct fee breakdown though, some changes are needed (see here).

[Added] New v3

fees structure If you want to show the correct detailed fee breakdown, you can use the newly added v3 properties of the GET /suggested-fees response data type FeesResponse = { // ... other fields + totalRelayFee: { // relayerCapitalFee + relayerGasFee + lpFee + pct: string; + total: string; + }; + relayerCapitalFee: { + pct: string; + total: string; + }; + relayerGasFee: { + pct: string; + total: string; + }; + lpFee: { + pct: string; + total: string; + }; } Using the values of the new fees struct, you need to call the deposit function like const tx =

await spokePool . deposit (// ... other args feesResponse . totalRelayFee . pct)

Breaking changes (mid- to long-term)

Even though there are no actionable changes short-term, some change will be actionable mid- to long-term.

[Deprecated] Flattened fees

Pct / Total fields will be removed As described here, the v3 fees have a different structure now. Eventually the redundant fields capitalFeePct , capitalFeeTotal , relayGasFeePct , relayGasFeeTotal and lpFeePct will be removed type FeesResponse = { // ... other fields - capitalFeePct: string; - capitalFeeTotal: string; - relayGasFeePct: string; - relayGasFeeTotal: string; - relayFeePct: string; // capitalFeePct + gasFeePct - relayFeeTotal: string; // capitalFeeTotal + gasFeeTotal - lpFeePct: string; + totalRelayFee: { // relayerCapitalFee + relayerGasFee + lpFee + pct: string; + total: string; + }; + relayerCapitalFee: { + pct: string; + total: string; + }; + relayerGasFee: { + pct: string; + total: string; + }; + lpFee: { + pct: string; + total: string; + }; }

Changes to Across Smart Contracts

Breaking Change: Across+ (Composable Bridging)

All users of Across+ (Composable Bridging) are required to upgrade their messaging implementations to use handleV3AcrossMessage(). Across v3 updates the callback interface for receiving messages in a recipient contract. The new function prototype is as follows: function handleV3AcrossMessage(address tokenSent, uint256 amount, address relayer, bytes memory message) external; The key differences to Across v2 are: * handleAcrossMessage() -> handleV3AcrossMessage * Function parameter bool fillCompleted has been removed. For a smooth transition to Across v3, integrators are recommended to implement concurrent support for handleAcrossMessage() and handleV3AcrossMessage(). handleAcrossMessage() can be removed at a later date. The motivation for updating the function prototype is that partial fills are no longer possible in Across v3, so messaging recipients no longer require special logic to account for them.

Events

If you depend on querying SpokePool events to track the status of bridge transfers, then this section is designed to support you.

Breaking Change: Deprecating Across V2 Events

In Across V2, a bridge consists of a FundsDeposited event (emitted by a deposit() call) on the origin chain and a FilledRelay event on the destination chain. They must match on all common parameters and the FilledRelay#realizedLpFeePct must be equal to the LP fee at the FundsDeposited#quoteTimestamp based on the computation rules described in the UMIP . Moreover, the FilledRelay#destinationToken also needs to match the FundsDeposited#originToken based on the matching rules described in the UMIP. In Across V3, the deposit() call now emits a V3FundsDeposited event, which must match with a V3FilledRelay event as explained below . This means that FundsDeposited events (V2 events) will no longer be emitted. event

FundsDeposited (uint256 amount , uint256 originChainId , uint256

indexed destinationChainId , int64 relayerFeePct , uint32

indexed depositId , uint32 quoteTimestamp , address originToken , address recipient , address

indexed depositor , bytes message) event

FilledRelay (uint256 amount , uint256 totalFilledAmount , uint256 fillAmount , uint256 repaymentChainId , uint256

indexed originChainId , uint256 destinationChainId , int64 relayerFeePct , int64 realizedLpFeePct , uint32

indexed depositId , address destinationToken , address relayer , address

indexed depositor , address recipient , bytes message , RelayExecutionInfo updatableRelayData) struct

RelayExecutionInfo

{ address recipient ; bytes message ; int64 relayerFeePct ; bool isSlowRelay ; int256 payoutAdjustmentPct ; }

Across V3 Events

Across V3 bridge transfers consist of a V3FundsDeposited event emitted on the origin chain and a FilledV3Relay event emitted on the destination chain. V3FundsDeposited and FilledV3Relay must match on all common parameters except for outputToken . This can be set to 0x0 (the zero address) at deposit time to signal to relayers that they should use the "equivalent" token on the destinationChainId to fill the deposit, therefore if V3FundsDeposited#outputToken == 0x0 then FilledV3Relay#outputToken must be equal to the "equivalent" token on the destination chain. The Across UMIP should be updated to explain what it means to be "equivalent" but this essentially means the output and input token map to the same token on Ethereum, for example they could be the different USDC addresses on different L2s. Note that the amount transferred to the recipient is relayExecutionInfo.updatedOutputAmount // Fill type is emitted in the FilledRelay event to assist Dataworker with determining which types of // fills to refund (e.g. only fast fills) and whether a fast fill created a sow fill excess. enum

FillType

{ FastFill , // Fast fills are normal fills that do not replace a slow fill request. ReplacedSlowFill , // Replaced slow fills are fast fills that replace a slow fill request. This type is used by the Dataworker // to know when to send excess funds from the SpokePool to the HubPool because they can no longer be used // for a slow fill execution. SlowFill // Slow fills are requested via requestSlowFill and executed by executeSlowRelayLeaf after a bundle containing // the slow fill is validated. } struct

V3RelayExecutionEventInfo

{ address updatedRecipient ; bytes updatedMessage ; uint256 updatedOutputAmount ; FillType fillType ; } event

V3FundsDeposited (address inputToken , address outputToken , uint256 inputAmount , uint256 outputAmount , uint256

indexed destinationChainId , uint32

indexed depositId , uint32 quoteTimestamp , uint32 fillDeadline , uint32 exclusivityDeadline , address

indexed depositor , address recipient , address exclusiveRelayer , bytes message); event

FilledV3Relay (address inputToken , address outputToken , uint256 inputAmount , uint256 outputAmount , uint256 repaymentChainId , uint256

indexed originChainId , uint32

indexed depositId , uint32 fillDeadline , uint32 exclusivityDeadline , address exclusiveRelayer , address

indexed relayer , address depositor , address recipient , bytes message , V3RelayExecutionEventInfo relayExecutionInfo);

Important changes for Relayers

This is an abridged summary of important breaking changes for Relayers to be aware of. All deposits emitted in V3 contracts will be V3FundsDeposited events which can only be filled via SpokePool#filledV3Relay. The important changes here are: * realizedLpFeePct * The relayer no longer needs to set the * realizedLpFeePct * as a parameter when calling the fill function. LP fees will still be charged to relayers at refund time (i.e. they will receive the inputAmount minus the LP fees) and these fees will still be computed using the formula in the UMIP based on the * deposit.quoteTimestamp. * This is a nice quality of life improvement for relayers who will still need to compute the LP fee for a deposit but can use a much wider margin of error, as this fee will only be important for computing their profitability and they will not be at risk of sending an invalid fill if this fee is off by a small amount. * outputToken: * See the section above about how * outputToken * must be set if FundsDeposited#outputToken == 0x0. This should be set to the "equivalent" token for the destination chain in this case. * No partial fills: it is not possible to send partial fills in V3. * Slow fills: Slow fills will be requested via a new function * requestSlowFill * . Slow fills can only be requested on deposits where the * inputToken * and * outputToken * are "equivalent". Slow fills will pay out the * inputAmount * realizedLpFeePct * , which will be set as the slow fill's * updatedOutputAmount /* *@notice Fulfill request to bridge cross chain by sending specified output tokens to the recipient. * @dev The fee paid to relayers and the system should be captured in the spread between output * amount and input amount when adjusted to be denominated in the input token. A relayer on the destination * chain will send outputAmount of outputTokens to the recipient and receive inputTokens on a repayment * chain of their choice. Therefore, the fee should account for destination fee transaction costs, the * relayer's opportunity cost of capital while they wait to be refunded following an optimistic challenge * window in the HubPool, and a system fee charged to relayers. * @dev The hash of the relayData will be used to uniquely identify the deposit to fill, so * modifying any params in it will result in a different hash and a different deposit. The hash will comprise * all parameters passed to depositV3() on the origin chain along with that chain's*

*chainId(). This chain's * chainId() must therefore match the destinationChainId passed into depositV3. * Relayers are only refunded for filling deposits with deposit hashes that map exactly to the one emitted by the * origin SpokePool therefore the relayer should not modify any params in relayData. * @dev Cannot fill more than once. Partial fills are not supported. * @param relayData struct containing all the data needed to identify the deposit to be filled. Should match * all the same-named parameters emitted in the origin chain V3FundsDeposited event. * - depositor: The account credited with the deposit who can request to "speed up" this deposit by modifying * the output amount, recipient, and message. * - recipient The account receiving funds on this chain. Can be an EOA or a contract. If * the output token is the wrapped native token for the chain, then the recipient will receive native token if * an EOA or wrapped native token if a contract. * - inputToken: The token pulled from the caller's account to initiate the deposit. The equivalent of this * token on the repayment chain will be sent as a refund to the caller. * - outputToken The token that the caller will send to the recipient on the destination chain. Must be an * ERC20. * - inputAmount: This amount, less a system fee, will be sent to the caller on their repayment chain of choice as a refund * following an optimistic challenge window in the HubPool. * - outputAmount: The amount of output tokens that the caller will send to the recipient. * - originChainId: The origin chain identifier. * - exclusiveRelayer The relayer that will be exclusively allowed to fill this deposit before the * exclusivity deadline timestamp. * - fillDeadline The deadline for the caller to fill the deposit. After this timestamp, * the fill will revert on the destination chain. * - exclusivityDeadline: The deadline for the exclusive relayer to fill the deposit. After this * timestamp, anyone can fill this deposit. * - message The message to send to the recipient if the recipient is a contract that implements a * handleV3AcrossMessage() public function * @param repaymentChainId Chain of SpokePool where relayer wants to be refunded after the challenge window has * passed. Will receive inputAmount of the equivalent token to inputToken on the repayment chain.* / function

fillV3Relay (V3RelayData calldata relayData ,

uint256 repaymentChainId) // This struct represents the data to fully specify a**unique** relay submitted on this chain. // This data is hashed with the chainId() and saved by the SpokePool to prevent collisions and protect against // replay attacks on other chains. If any portion of this data differs, the relay is considered to be // completely distinct. struct

V3RelayData

{ // The address that made the deposit on the origin chain. address depositor ; // The recipient address on the destination chain. address recipient ; // This is the exclusive relayer who can fill the deposit before the exclusivity deadline. address exclusiveRelayer ; // Token that is deposited on origin chain by depositor. address inputToken ; // Token that is received on destination chain by recipient. address outputToken ; // The amount of input token deposited by depositor. uint256 inputAmount ; // The amount of output token to be received by recipient. uint256 outputAmount ; // Origin chain id. uint256 originChainId ; // The id uniquely identifying this deposit on the origin chain. uint32 depositId ; // The timestamp on the destination chain after which this deposit can no longer be filled. uint32 fillDeadline ; // The timestamp on the destination chain after which any relayer can fill the deposit. uint32 exclusivityDeadline ; // Data that is forwarded to the recipient. bytes message ; } struct

V3SlowFill

{ V3RelayData relayData ; uint256 chainId ; uint256 updatedOutputAmount ; }

Important changes for dApp Developers

The deposit interface will not change. The function will however emit a new event V3FundsDeposited as outlined in Across V3 Events . The fillDeadline will be set to MAX_UINT , meaning that the deposit will never expire and be refunded to the origin chain. The outputToken will be set to 0x0 to signal to depositors that the "equivalent" output token should be replaced at fill time. The exclusivityDeadline and exclusiveRelayer will be set to 0 and 0x0 respectively suggesting that there is no exclusivity period. Finally, the outputAmount will be equal to inputAmount * (1 - relayerFeePct) . 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

payable ; // Old function: function

deposit (address recipient , address originToken , uint256 amount , uint256 destinationChainId , int64 relayerFeePct , uint32 quoteTimestamp , bytes

memory message , uint256 maxCount)

external

payable ; To speed up a deposit, a new function will need to be used. Previously this was named speedUpDeposit and the new function to speed up a V3FundsDeposited event will be speedUpV3Deposit. function

speedUpV3Deposit (address depositor , uint32 depositId , uint256 updatedOutputAmount , address updatedRecipient , bytes

calldata updatedMessage , bytes

calldata depositorSignature)

external ; // Old function: function

speedUpDeposit (address depositor , int64 updatedRelayerFeePct , uint32 depositId , address updatedRecipient , bytes

memory updatedMessage , bytes

memory depositorSignature)

external ; Introduction -Previous Developer Notes Next- Concepts What are Cross-chain Intents? Last modified4d ago On this page Introduction Summary of Breaking Changes Across API migration guide Overview Non-breaking changes Breaking changes (mid- to long-term) Changes to Across Smart Contracts Breaking Change: Across+ (Composable Bridging) Events Important changes for Relayers Important changes for dApp Developers