execute-route)

Execute Routes/Quotes

We allow you to execute any on-chain or cross-chain swap and bridging transfer and a combination of both. The LI.FI SDK offers functionality to execute routes and quotes. In this guide, you'll learn how to utilize the SDK's features to handle complex cross-chain transfers, manage execution settings, and control the transaction flow.

Execute route

Let's say you have obtained the route. Refer toRequest Routes/Quotes for more details.

Please make sure you've configured SDK with EVM/Solana providers. Refer toConfigure SDK Providers for more details.

Now, to execute the route, we can use theexecuteRoute function. Here is a simplified example of how to use it:

```

Copy import{ executeRoute,getRoutes }from'@lifi/sdk'

constresult=awaitgetRoutes({ fromChainId:42161,// Arbitrum toChainId:10,// Optimism fromTokenAddress:'0xaf88d065e77c8cC2239327C5EDb3A432268e5831',// USDC on Arbitrum toTokenAddress:'0xDA10009cBd5D07dd0CeCc66161FC93D7c9000da1',// DAI on Optimism fromAmount:'10000000',// 10 USDC // The address from which the tokens are being transferred. fromAddress:'0x552008c0f6870c2f77e5cC1d2eb9bdff03e30Ea0', })

constroute=result.routes[0]

constexecutedRoute=awaitexecuteRoute(route,{ // Gets called once the route object gets new updates updateRouteHook(route) { console.log(route) }, })

```

TheexecuteRoute function internally manages allowance and balance checks, chain switching, transaction data retrieval, transactions submission, and transactions status tracking.

- Parameters:
  - route
  - (Route
  -): The route to be executed.
  - executionOptions
  - (ExecutionOptions
  - , optional): An object containing settings and callbacks for execution.

- Returns:
-
  - Promise
-
  - : Resolves when execution is done or halted and rejects when it is failed.

Execution Options

All execution options are optional, but we recommend reviewing their descriptions to determine which ones may be beneficial for your use case.

Certain options, such asacceptExchangeRateUpdateHook , can be crucial for successfully completing a transfer if the exchange rate changes during the process.

updateRouteHook

The function is called when the route object changes during execution. This function allows you to handle route updates, track execution status, transaction hashes, etc. SeeMonitor route execution section for more details.

- Parameters
- :
-
  - updatedRoute
-
  - (RouteExtended
-
  -): The updated route object.

updateTransactionRequestHook

The function is intended for advanced usage, and it allows you to modify swap/bridge transaction requests or token approval requests before they are sent, e.g., updating gas information.

- Parameters
- :
-
  - updatedTxRequest
-
  - (TransactionRequestParameters
-
  -): The transaction request parameters need to be updated.
- Returns
- :Promise
- : The modified transaction parameters.

acceptExchangeRateUpdateHook

This function is called whenever the exchange rate changes during a swap or bridge operation. It provides you with the old and new amount values. To continue the execution, you should returntrue . If this hook is not provided or if you returnfalse , the SDK will throw an error. This hook is an ideal place to prompt your users to accept the new exchange rate.

- Parameters
- :
-
  - toToken
-
  - (Token
-
  -): The destination token.
-
  - oldToAmount
-
  - (string
-
  -): The previous amount of the target token.
-
  - newToAmount
-
  - (string

-
  -): The new amount of the target token.
- Returns
- :Promise
- : Whether the update is accepted.
- Throws:
- TransactionError: Exchange rate has changed!

switchChainHook

A hook that is triggered when a chain switch is required. This function allows the user to handle the chain switch process. When specified it will replace theswitchChain hook from EVM Provider. SeeSetup EVM Provider .

- Parameters
- :
-
  - chainId
-
  - (number
-
  -): The ID of the chain to which to switch.
- Returns
- :Promise
- : The new wallet client after switching chains.

executeInBackground

A boolean flag indicating whether the route execution should continue in the background without requiring user interaction. SeeUpdate route execution andResume route execution sections for details on how to utilize this option.

- Type
- :boolean
- Default
- :false

infiniteApproval

A boolean flag indicating whether to attempt setting an infinite token approval, allowing the smart contract to spend tokens without limits. When using this option with wallet extensions, users can still modify the spending cap during the prompt, with infinite approval serving as the initial suggestion.

- Type
- :boolean
- Default
- :false

Manage route execution

After starting route execution, there might be use cases when you need to adjust execution settings, stop execution and come back later, or move execution to the background. We provide several functions to achieve that.

Update route execution

TheupdateRouteExecution function is used to update the settings of an ongoing route execution.

One common use case is to push the execution to the background, for example, when a user navigates away from the execution page in your dApp. When this function is called, the execution will continue until it requires user interaction (e.g., signing a transaction or switching the chain). At that point, the execution will halt, and theexecuteRoute promise will be resolved.

To move the execution back to the foreground and make it active again, you can callresumeRoute with the same route object. The execution will then resume from where it was halted.

```

Copy import{ updateRouteExecution }from'@lifi/sdk'

updateRouteExecution(route,{ executeInBackground:true});

```

- Parameters:
- route
- (Route
-): The active route to be updated.
- executionOptions
- (ExecutionOptions
- ,required
-): An object containing settings and callbacks for execution.

Resume route execution

TheresumeRoute function is used to resume a halted, aborted, or failed route execution from the point where it stopped. It is crucial to callresumeRoute with the latest active route object returned from theexecuteRoute function or the most recent version of the updated route object from theupdateRouteHook .

Common Use Cases

- Move Execution to Foreground
- : When a user navigates back to the execution page in your dApp, you can call this function to move the execution back to the foreground. The execution will resume from where it was halted.
- Page Refresh
- : If the user refreshes the page in the middle of the execution process, calling this function will attempt to resume the execution.
- User Interaction Errors
- : If the user rejects a chain switch, declines to sign a transaction, or encounters any other error, you can call this function to attempt to resume the execution.

```

Copy import{ resumeRoute }from'@lifi/sdk'

constroute=awaitresumeRoute(route,{ executeInBackground:false});

```

- Parameters:
- route
- (Route
-): The route to be resumed to execution.
- executionOptions
- (ExecutionOptions
- , optional): An object containing settings and callbacks for execution.
- Returns:
- Promise
- : Resolves when execution is done or halted and rejects when it is failed.

Stop route execution

ThestopRouteExecution function is used to stop the ongoing execution of an active route. It stops any remaining user interaction within the ongoing execution and removes the route from the execution queue. However, if a transaction has already been signed and sent by the user, it will be executed on-chain.

```

Copy import{ stopRouteExecution }from'@lifi/sdk'
```

conststoppedRoute=stopRouteExecution(route);

```

- Parameters:
- route
- (Route
- ): The route that is currently being executed and needs to be stopped.
- Returns:
- Route
- : The route object that was stopped.

Monitor route execution

Monitoring route execution is important and we provide tools for tracking progress, receiving data updates, accessing transaction hashes, and explorer links.

Brief description of steps

Aroute object includes multiplestep objects, each representing a set of transactions that should be completed in the specified order. Each step can include multiple transactions that require a signature, such as an allowance transaction followed by the main swap or bridge transaction. Read moreLI.FI Terminology .

Understanding theexecution object

Eachstep within aroute has anexecution object. This object contains all the necessary information to track the execution progress of that step. Theexecution object has aprocess array where each entry represents a sequential stage in the execution. The latest process entry contains the most recent information about the execution stage.

Process array

Theprocess array within theexecution object details each step's progression. Eachprocess object has a type and status and might also include a transaction hash and a link to a blockchain explorer after the user signs the transaction.

Tracking progress

To monitor the execution progress, you leverage theupdateRouteHook callback and iterate through the route steps, checking theirexecution objects. Look at theprocess array to get the latest information about the execution stage. The most recent entry in theprocess array will contain the latest transaction hash, status, and other relevant details.

Example to access transaction hashes

```

Copy constgetTransactionLinks=(route:RouteExtended)=>{ route.steps.forEach((step,index)=>{ step.execution?.process.forEach((process)=>{ if(process.txHash) { console.log(Transaction Hash for Step{index+1}, Process{process.type}:, process.txHash) } }) }) }

constexecutedRoute=awaitexecuteRoute(route,{ updateRouteHook(route) { getTransactionLinks(route) }, })

```

Get active routes

To get routes that are currently being executed (active), you can usegetActiveRoutes andgetActiveRoute functions.

```

Copy import{ getActiveRoute,getActiveRoutes,RouteExtended }from'@lifi/sdk'

constactiveRoutes:RouteExtended[]=getActiveRoutes();

constrouteId=activeRoutes[0].routeId;

constactiveRoute=getActiveRoute(routeId);

```
```

Execute quote

To execute a quote using theexecuteRoute , you need to convert it to a route object first. We provideconvertQuoteToRoute helper function to transform quote objects to route objects. This applies to both standard and contract call quotes.

```

Copy import{ convertQuoteToRoute,executeRoute,getQuote }from'@lifi/sdk';

constquoteRequest:QuoteRequest={ fromChain:42161,// Arbitrum toChain:10,// Optimism fromToken:'0xaf88d065e77c8cC2239327C5EDb3A432268e5831',// USDC on Arbitrum toToken:'0xDA10009cBd5D07dd0CeCc66161FC93D7c9000da1',// DAI on Optimism fromAmount:'10000000',// 10 USDC // The address from which the tokens are being transferred. fromAddress:'0x552008c0f6870c2f77e5cC1d2eb9bdff03e30Ea0', };

constquote=awaitgetQuote(quoteRequest);

constroute=convertQuoteToRoute(quote)

constexecutedRoute=awaitexecuteRoute(route,{ // Gets called once the route object gets new updates updateRouteHook(route) { console.log(route) }, })

```

Manual route execution

In addition to using theexecuteRoute function, you can execute routes and quotes manually. This approach requires developers to handle the logic for obtaining transaction data, switching chains, sending transactions, and tracking transaction status independently.

Initially, when route objects are requested, they do not include transaction data. This is because multiple route options are provided, and generating transaction data for all options would substantially delay the response. Each route consists of multiple steps, and once a user selects a route, transaction data for each step should be requested individually using thegetStepTransaction function (see example below). Each step should be executed sequentially, as each step depends on the outcome of the previous one.

On the other hand, quote objects are returned with transaction data included, so thegetStepTransaction call is not necessary, and they can be executed immediately.

After sending a transaction using the obtained transaction data, you can track the status of the transaction using thegetStatus function. This function helps you monitor the progress and completion of each transaction. Read moreStatus of a Transaction .

Here's a simplified example. For the sake of simplicity, this example omits balance checks, transaction replacements, error handling, chain switching, etc. However, in a real implementation, you should include these additional functionalities to have a robust solution and ensure reliability.

```

Copy import{ getStepTransaction,getStatus }from'@lifi/sdk';

// Simplified example function to execute each step of the route sequentially asyncfunctionexecuteRouteSteps(route) { for(conststepofroute.steps) { // Request transaction data for the current step conststep=awaitgetStepTransaction(step);

// Send the transaction (e.g. using Viem) consttransactionHash=awaitsendTransaction(step.transactionRequest);

// Monitor the status of the transaction letstatus; do{ constresult=awaitgetStatus({ txHash:transactionHash, fromChain:step.action.fromChainId, toChain:step.action.toChainId, bridge:step.tool, }) status=result.status

console.log(Transaction status for{transactionHash}:,status);

// Wait for a short period before checking the status again awaitnewPromise(resolve=>setTimeout(resolve,5000));

}while(status!=='DONE'&&status!=='FAILED');

if(status===='FAILED') { console.error(Transaction{transactionHash}failed); return; } }

console.log('All steps executed successfully'); }

```

getStepTransaction

- Parameters:
- step
- (LiFiStep
-): The step object for which we need to get transaction data.
- options
- (RequestOptions
- , optional): An object containing request options, such asAbortSignal
- , which can be used to cancel the request if necessary.
- Returns:
- Promise
- : A promise that resolves to the step object containing the transaction data.

getStatus

- Parameters:
- params
- (GetStatusRequest
-): The parameters for checking the status include the transaction hash, source and destination chain IDs, and the DEX or bridge name.
- options
- (RequestOptions
- , optional): An object containing request options, such asAbortSignal
- , which can be used to cancel the request if necessary.
- Returns:
- Promise
- : A promise that resolves to a status response containing all relevant information about the transfer.