introduction-to-sdk-ecosystem-providers)

Configure SDK Providers

Seamlessly connecting every ecosystem for your needs

Introduction to SDK Ecosystem Providers

The LI.FI SDK supports different blockchain ecosystems, allowing you to integrate with EVM and Solana networks, with more ecosystems on the way. Internally, providers act as abstractions for each ecosystem, handling crucial tasks such as address resolution, balance checking, and transaction handling during route/quote execution.

These ecosystem providers are designed with modularity in mind and are fully tree-shakable, ensuring that they do not add unnecessary weight to your bundle if not used.

The SDK offers two providers,EVM andSolana , both with similar configuration options respective to their ecosystems.

```

Copy import{ EVM,Solana }from'@lifi/sdk'

```

The setup for both providers focuses on utilizing a wallet client, wallet adapter, or a similar wallet interface concept depending on the ecosystem-specific libraries and standards. This unified approach simplifies managing wallets and transactions across EVM-compatible and Solana chains.

Different types of wallets/accounts

To execute quotes/routes via a specific provider, that provider must be capable of signing transactions. SDK providers support signing transactions over the following types of wallets/accounts:

- Local Accounts
- (e.g. private key/mnemonic wallets).
- Local accounts are wallets managed using private keys or mnemonic phrases. This setup is often used in backend services or scenarios where automated signing and transaction management are required.
- JSON-RPC Accounts
- (e.g. Browser Extension Wallets, WalletConnect, etc.).
- Using JSON-RPC accounts involves connecting through a Web3 provider, e.g.window.ethereum
- , and managing the user's account within the browser or mobile context. This setup is popular among dApps UIs and is often used together with libraries likeWagmi
- or@solana/web3.js
- .

These account types and interaction methods allow developers to choose the most suitable approach for integrating the SDK with their applications.

Setup EVM Provider

The EVM provider execution logic is built based on theViem library, using some of its types and terminology.

Options available for configuring the EVM provider:

- getWalletClient
- : A function that returns aWalletClient

- instance.
- switchChain
- : A hook for switching between different networks.

Local Accounts

When using local accounts, developers need a predefined list of chains they plan to interact with in order to switch chains during transaction execution. These chains can be either from theviem/chains package or fetched from LI.FI API and adopted to viem'sChain type.

Here's a basic example using chains fromviem/chains :

```

Copy import{ createConfig,EVM }from'@lifi/sdk' importtype{ Chain }from'viem' import{ createWalletClient,http }from'viem' import{ privateKeyToAccount }from'viem/accounts' import{ arbitrum,mainnet,optimism,polygon,scroll }from'viem/chains'

constaccount=privateKeyToAccount('PRIVATE_KEY')

constchains=[arbitrum,mainnet,optimism,polygon,scroll]

constclient=createWalletClient({ account, chain:mainnet, transport:http(), })

createConfig({ integrator:'Your dApp/company name', providers:[ EVM({ getWalletClient:async()=>client, switchChain:async(chainId)=> // Switch chain by creating a new wallet client createWalletClient({ account, chain:chains.find((chain)=>chain.id==chainId)asChain, transport:http(), }), }), ], })

```

JSON-RPC Accounts

The best way to interact withJSON-RPC accounts and passWalletClient to theEVM provider is to use theWagmi library. Developers can configure Wagmi chains either by using chains from theviem/chains package or fetching chains from LI.FI API and adapting them to Viem'sChain type.

Below is a simplified example of how to set up the EVM provider using chains from the LI.FI API in conjunction with Wagmi and React.

We provide auseSyncWagmiConfig hook that synchronizes fetched chains with the Wagmi configuration and updates connectors. Please note that we do not initialize the Wagmi configuration with connectors. Additionally, we setreconnectOnMount tofalse since thereconnect action will be called withinuseSyncWagmiConfig hook after the chains are synchronized with the configuration and connectors.

```

Copy import{ ChainType,EVM,config,createConfig,getChains }from'@lifi/sdk'; import{ useSyncWagmiConfig }from'@lifi/wallet-management'; import{ useQuery }from'@tanstack/react-query'; import{ getWalletClient,switchChain }from'@wagmi/core'; import{typeFC,typePropsWithChildren }from'react'; import{ createClient,http }from'viem'; import{ mainnet }from'viem/chains'; importtype{ Config,CreateConnectorFn }from'wagmi'; import{ WagmiProvider,createConfigascreateWagmiConfig }from'wagmi'; import{ injected }from'wagmi/connectors';

// List of Wagmi connectors constconnectors:CreateConnectorFn[]=[injected()];

// Create Wagmi config with default chain and without connectors constwagmiConfig:Config=createWagmiConfig({ chains: [mainnet], client({ chain }) { returncreateClient({ chain,transport:http() }); }, });

// Create SDK config using Wagmi actions and configuration createConfig({ integrator:'Your dApp/company name', providers:[ EVM({ getWalletClient:()=>getWalletClient(wagmiConfig), switchChain:async(chainId)=>{ constchain=awaitswitchChain(wagmiConfig,{ chainId }); returngetWalletClient(wagmiConfig,{ chainId:chain.id }); }, }), ], // We disable chain preloading and will update chain configuration in runtime preloadChains:false, });

exportconstCustomWagmiProvider:FC=({ children })=>{ // Load EVM chains from LI.FI API using getChains action from LI.FI SDK const{ data:chains}=useQuery({ queryKey:['chains']asconst, queryFn:async()=>{ constchains=awaitgetChains({ chainTypes:[ChainType.EVM], }); // Update chain configuration for LI.FI SDK config.setChains(chains); returnchains; }, });

// Synchronize fetched chains with Wagmi config and update connectors useSyncWagmiConfig(wagmiConfig,connectors,chains);

return( {children} ); };

```

Update provider configuration

Additionally, providers allow for dynamic updates to its initial configuration via thesetOptions function.

Here's an example of how to modify the initial configuration forEVM provider:

```

Copy import{ createConfig,EVM }from'@lifi/sdk' import{ createWalletClient,http }from'viem' import{ privateKeyToAccount }from'viem/accounts' import{ arbitrum,mainnet }from'viem/chains'

constaccount=privateKeyToAccount('PRIVATE_KEY')

constmainnetClient=createWalletClient({ account, chain:mainnet, transport:http(), })

constevmProvider=EVM({ getWalletClient:async()=>mainnetClient, })

createConfig({ integrator:'Your dApp/company name', providers:[evmProvider], })

constoptimismClient=createWalletClient({ account, chain:arbitrum, transport:http(), })

evmProvider.setOptions({ getWalletClient:async()=>optimismClient, })

```

Support for Ethers.js and other alternatives

Developers can still use Ethers.js or any other alternative Web3 library in their project andconvert Signer /Provider objects to Viem'sWalletClient before passing it to the EVM provider configuration.

Setup Solana Provider

The Solana provider execution logic is built based on the@solana/web3.js and@solana/wallet-adapter-base libraries, using some of its types and terminology.

Options available for configuring the EVM provider:

- getWalletAdapter
- : A function that returns a WalletAdapter instance.

Local Wallet Adapter

Standard Solana libraries do not offer a built-in method for creating a wallet adapter directly from a private key. To address this limitation, we provide theKeypairWalletAdapter . This custom adapter enables users to create a wallet adapter from a private key.

It is worth noting that theKeypairWalletAdapter is designed specifically for backend or testing purposes and should not be used in user-facing code to prevent the risk of exposing your private key.

```

Copy import{ createConfig,KeypairWalletAdapter,Solana }from'@lifi/sdk'

constwalletAdapter=newKeypairWalletAdapter('PRIVATE_KEY')

createConfig({ integrator:'Your dApp/company name', providers:[ Solana({ getWalletAdapter:async()=>walletAdapter, }), ], })

```

JSON-RPC Wallet Adapter

To interact with JSON-RPC accounts and passWalletAdapter to the Solana provider, we recommend using the@solana/wallet-adapter-base and@solana/wallet-adapter-react libraries. UnlikeWagmi , Solana configuration for React does not have global configurations. Therefore, we need to use React hooks to update the SDK configuration at runtime.

Below is a simplified example of how to set up the Solana provider.

SDKProviders.tsx SolanaProvider.tsx ```

Copy import{ Solana,config,createConfig }from'@lifi/sdk'; importtype{ SignerWalletAdapter }from'@solana/wallet-adapter-base'; import{ useWallet }from'@solana/wallet-adapter-react'; import{ useEffect }from'react';

createConfig({ integrator:'Your dApp/company name', });

exportconstSDKProviders=()=>{ const{wallet}=useWallet();

useEffect(()=>{ // Configure SDK Providers config.setProviders([Solana({ asyncgetWalletAdapter() { returnwallet?.adapterasSignerWalletAdapter; }, }),]); },[wallet?.adapter]);

returnnull; };

Copy importtype{ Adapter }from'@solana/wallet-adapter-base'; import{ WalletAdapterNetwork }from'@solana/wallet-adapter-base'; import{ ConnectionProvider, WalletProvider, }from'@solana/wallet-adapter-react'; import{ clusterApiUrl }from'@solana/web3.js'; import{typeFC,typePropsWithChildren }from'react'; import{ SDKProviders }from'./SDKProviders.js';

constendpoint=clusterApiUrl(WalletAdapterNetwork.Mainnet); /* *Wallets that implement either of these standards will be available automatically. * * - Solana Mobile Stack Mobile Wallet Adapter Protocol * (https://github.com/solana-mobile/mobile-wallet-adapter) * - Solana Wallet Standard * (https://github.com/solana-labs/wallet-standard) * * If you wish to support a wallet that supports neither of those standards, * instantiate its legacy wallet adapter here. Common legacy adapters can be found * in the npm package @solana/wallet-adapter-wallets.* / constwallets:Adapter[]=[];

exportconstSVMBaseProvider:FC=({ children })=>{ return({/ *Configure Solana SDK provider* /} {children}); };

``` Request Routes/Quotes Last updated4 months ago
```