How to create and use a SimpleAccount with permissionless.js

SimpleAccount is the original reference sample implementation of an ERC-4337 made by the Eth-Infinitism team. Despite being a reference implementation, it is widely used in production. It allows for a single EOA signer to sign user operations for the account. This guide will show you how to create and use a SimpleAccount with permissionless.js.

Steps

Create the clients

First we must create the public, bundler, and (optionally) paymaster clients that will be used to interact with the SimpleAccount.

```

exportconstpublicClient=createPublicClient({ transport:http("https://CHAIN.infura.io/v3/API_KEY"), });

exportconstpaymasterClient=createPimlicoPaymasterClient({ entryPoint, transport:http( "https://api.pimlico.io/v2/CHAIN/rpc?apikey=API_KEY", ), });

```

Create the SimpleAccount

For a full list of options for creating a SimpleAccount, take a look at the reference documentation page forsignerToSimpleSmartAccount . You can create a SimpleAccount with the canonical module addresses by specifying the factory address the account will be deployed from. You can also pass anaddress to use an already created SimpleAccount.

```

constsimpleAccount=awaitprivateKeyToSimpleSmartAccount(publicClient, { privateKey:"0xPRIVATE_KEY", factoryAddress:"0x9406Cc6185a346906296840746125a0E44976454", entryPoint:"0x5FF137D4b0FDCD49DcA30c7CF57E578a026d2789",// global entrypoint address:"0x..."// optional, only if you are using an already created account });

```

Create the smart account client

The smart account client is a permissionless.js client that is meant to serve as an almost drop-in replacement for viem'swalletClient .

```

constsmartAccountClient=createSmartAccountClient({ account: simpleAccount, entryPoint:"0x5FF137D4b0FDCD49DcA30c7CF57E578a026d2789", chain: sepolia, bundlerTransport:http( "https://api.pimlico.io/v1/CHAIN/rpc?apikey=API_KEY", ), middleware: { sponsorUserOperation: paymasterClient.sponsorUserOperation,// optional } });

```

Fetch the gas prices (optional)

If you're using Pimlico as your bundler, fetch the required gas price to use beforehand and pass it in as themaxFeePerGas andmaxPriorityFeePerGas parameters. Other providers might have different requirements for fetching the gas price.

```

exportconstpimlicoBundlerClient=createPimlicoBundlerClient({ transport:http( "https://api.pimlico.io/v1/CHAIN/rpc?apikey=API_KEY", ), entryPoint:"0x5FF137D4b0FDCD49DcA30c7CF57E578a026d2789", });

constsmartAccountClient=createSmartAccountClient({ account: simpleAccount, entryPoint:"0x5FF137D4b0FDCD49DcA30c7CF57E578a026d2789", chain: sepolia, bundlerTransport:http( "https://api.pimlico.io/v1/CHAIN/rpc?apikey=API_KEY", ), middleware: { gasPrice:async()=> (awaitpimlicoBundlerClient.getUserOperationGasPrice()).fast,// use pimlico bundler to get gas prices sponsorUserOperation: paymasterClient.sponsorUserOperation,// optional } });
```

```

## Send a transaction

Transactions using permissionless.js simply wrap around user operations. This means you can switch to permissionless.js from your existing viem EOA codebase with minimal-to-no changes.

```

consttxHash=awaitsmartAccountClient.sendTransaction({ to:"0xd8da6bf26964af9d7eed9e03e53415d37aa96045", value:parseEther("0.1"), });

```

This also means you can also use viem Contract instances to transact without any modifications.

```

constnftContract=getContract({ address:'0xFBA3912Ca04dd458c843e2EE08967fC04f3579c2', abi: nftAbi, publicClient, walletClient: smartAccountClient, });

consttxHash=awaitnftContract.write.mint()

```
```