User sponsored

This tutorial represents the API integration flow where smart contract wallet has the native tokens to sponsor the transaction.

Pre-requisites:

- Biconomy bundler url (refer to the docs
- to get the same)
- Partial user operation, where* sender is the smart account Address
-
    - nonce can be calculated using the smart account contract methods.
-
    - initCode will be 0x, if the account is already deployed or can be fetched using contract methods.
-
    - paymasterAndData will be 0x, given the smart account has native tokens to pay for the gasFees.
-
    - callData is the abi encoded form of transaction.
-
    - It also requires putting a semi-valid/ dummy signature (e.g. a signature in the right length).

```
type

UserOperation

=

{ sender :

string ; nonce :

string ; initCode :

string ; callData :

string ; callGasLimit :

string ; verificationGasLimit :

string ; preVerificationGas :

string ; maxFeePerGas :

string ; maxPriorityFeePerGas :

string ; paymasterAndData :

string ; signature :

string ; }

let partialUserOp : Partial < UserOperation

=

{ sender :

'0x4dF23B78543F5c2F9CBCDF09956288B3e97bb9a4' , nonce :

'0x08' , initCode :

"0x" , paymasterAndData :

"0x" , callData :

"0x0000189a000000000000000000000000322af0da66d00be980c7aa006377fcaaeee3bdfd000000000000000000000000000000000000000000000000002386f26fc10000000000000000000000000000000
, signature :

"0x0000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000000001c5b32F37F5beA87BDD5374eB2aC54eA8e000000000000000000000000000
, }
```

1. Calculate Gas estimations:

Based on the response, update the userOp gas limits and values.

```
async

function

getGasEstimations

( partialUserOp : Partial < UserOperation

        )

:

Promise < UserOperation

{ const url = "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44"

const data = { "method" :

"eth_estimateUserOperationGas" , "params" :

[ partialUserOp , "0x5ff137d4b0fdcd49dca30c7cf57e578a026d2789" ] , "id" : Date . now ( ) , "jsonrpc" :

"2.0" }

const response =

await axios . post ( url , data ) ; const

{ callGasLimit , verificationGasLimit , preVerificationGas , maxPriorityFeePerGas , maxFeePerGas }

= response . data . result return

{

... partialUserOp , callGasLimit : callGasLimit . toString ( ) , verificationGasLimit : verificationGasLimit . toString ( ) , preVerificationGas : preVerificationGas . toString ( ) , maxPriorityFeePerGas ,
maxFeePerGas }

as UserOperation ; }
```

If you are facing any AA** error, please refer to common errors . Also, make sure your smart account wallet has sufficient funds to sponsor the transaction even to estimate gas.

2. Sign userOperation

To sign the userOp, calculate the userOpHash and then sign it using the same signer, account was created. Follow this tutorial to learn about signing the userOp. Below is an example with ECDSA module .

```
async
function
signUserOp
( userOp : UserOperation )
{ const userOpHash =
getUserOpHash ( userOp ) ;
// It's defined in the full code in the end.
const moduleSig =
await signer . signMessage ( ethers . utils . arrayify ( userOpHash ) ) ; const signatureWithModuleAddress = ethers . utils . defaultAbiCoder . encode ( [ "bytes" ,
"address" ] , [ moduleSig ,
"0x0000001c5b32F37F5beA87BDD5374eB2aC54eA8e" ] , ) ; return
{
... userOp , signature : signatureWithModuleAddress } ;
}
```

3. Send UserOperation:

eth_sendUserOperation sends a user operation to the given network.

```
async
function
sendUserOp ( userOp : UserOperation )
{ const url = "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44"
const requestData =
{ jsonrpc :
'2.0' , method :
'eth_sendUserOperation' , id : Date . now ( ) , params :
[ userOp , "0x5ff137d4b0fdcd49dca30c7cf57e578a026d2789" ] , } ; const response =
await axios . post ( url , requestData ) ; return response . data . result ; }
```

4. Fetch user operation receipt

This API returns null until the transaction is mined, you will either need to poll or set a timeout. For reference checkout the sdk code . You can also get transaction hash from the response

```
async
function
getUserOpReceipt ( userOpHash :
string )
{ const url = "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44" const requestData =
{ jsonrpc :
'2.0' , method :
'eth_getUserOperationReceipt' , id : Date . now ( ) , params :
[ userOpHash ] , } ;
const
{ data }
=
await axios . post ( url , requestData ) ; return data . result ; }
```

If you are facing errors while integration, do checkout the common errors .

View Complete Code

```
import
{ ethers }
from
"ethers" ; import axios from
'axios' ; import
{
string ,
string
}
from
"ethers" ;
let provider =
new
ethers . providers . JsonRpcProvider ( "https://rpc.ankr.com/polygon_mumbai"
) ; let signer =
new
```

```
ethers . Wallet ( "private key" , provider ) ;
type
UserOperation
=
{ sender :
string ; nonce :
string ; initCode :
string ; callData :
string ; callGasLimit :
string ; verificationGasLimit :
string ; preVerificationGas :
string ; maxFeePerGas :
string ; maxPriorityFeePerGas :
string ; paymasterAndData :
string ; signature :
string ; }
async
function
getGasEstimations
( partialUserOp : Partial < UserOperation
        )
:
Promise < UserOperation
{ const url = "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44"
const data = { "method" :
"eth_estimateUserOperationGas" , "params" :
[ partialUserOp , "0x5ff137d4b0fdcd49dca30c7cf57e578a026d2789" ] , "id" : Date . now ( ) , "jsonrpc" :
"2.0" }
const response =
await axios . post ( url , data ) ; const
{ callGasLimit , verificationGasLimit , preVerificationGas , maxPriorityFeePerGas , maxFeePerGas }
= response . data . result return
{
... partialUserOp , callGasLimit : callGasLimit . toString ( ) , verificationGasLimit : verificationGasLimit . toString ( ) , preVerificationGas : preVerificationGas . toString ( ) , maxPriorityFeePerGas ,
maxFeePerGas }
as UserOperation ;
} function
getUserOpHash ( useOpMinusSignature : UserOperation ) :
string
{ const packedData = ethers . utils . defaultAbiCoder . encode ( [ "address" , "uint256" , "bytes32" , "bytes32" , "uint256" , "uint256" , "uint256" , "uint256" , "uint256" , "bytes32" , ] , [
useOpMinusSignature . sender , useOpMinusSignature . nonce , ethers . utils . keccak256 ( useOpMinusSignature . initCode ) , ethers . utils . keccak256 ( useOpMinusSignature . callData ) ,
useOpMinusSignature . callGasLimit , useOpMinusSignature . verificationGasLimit , useOpMinusSignature . preVerificationGas , useOpMinusSignature . maxFeePerGas , useOpMinusSignature .
maxPriorityFeePerGas , ethers . utils . keccak256 ( useOpMinusSignature . paymasterAndData ) , ] ) ;
const enc = ethers . utils . defaultAbiCoder . encode ( [ "bytes32" ,
"address" ,
"uint256" ] , [ ethers . utils . keccak256 ( packedData ) ,
"0x5ff137d4b0fdcd49dca30c7cf57e578a026d2789" ,
80001 ] ) ;
const userOpHash = ethers . utils . keccak256 ( enc ) ; return userOpHash ; }
async
function
signUserOp
( userOp : UserOperation ) :
Promise < UserOperation
{ const userOpHash =
getUserOpHash ( userOp ) ;
const moduleSig =
await signer . signMessage ( ethers . utils . arrayify ( userOpHash ) ) ; const signatureWithModuleAddress = ethers . utils . defaultAbiCoder . encode ( [ "bytes" ,
"address" ] , [ moduleSig ,
"0x0000001c5b32F37F5beA87BDD5374eB2aC54eA8e" ] , ) ; return
{
... userOp , signature : signatureWithModuleAddress } ; }
```

```
async

function

sendUserOp ( userOp : UserOperation )

{ const url = "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44" const requestData =

{ jsonrpc :

'2.0' , method :

'eth_sendUserOperation' , id : Date . now ( ) , params :

[ userOp , "0x5ff137d4b0fdcd49dca30c7cf57e578a026d2789" ] , } ; const response =

await axios . post ( url , requestData ) ; return response . data . result ; }

async

function

getUserOpReceipt ( userOpHash :

string )

{ const url = "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44" const requestData =

{ jsonrpc :

'2.0' , method :

'eth_getUserOperationReceipt' , id : Date . now ( ) , params :

[ userOpHash ] , } ;

const

{ data }

=

await axios . post ( url , requestData ) ; return data . result ; }

async

function

executePartialUserOp ( )

{ try

{ let partialUserOp : Partial < UserOperation

=

{ sender :

'0x4dF23B78543F5c2F9CBCDF09956288B3e97bb9a4' , nonce :

'36' , initCode :

"0x" , paymasterAndData :

"0x" , callData :

"0x0000189a000000000000000000000000322af0da66d00be980c7aa006377fcaaeee3bdfd000000000000000000000000000000000000000000000000002386f26fc1000000000000000000000000000000
, signature :

"0x00000000000000000000000000000000000000000000000000000000000000400000000000000000000000001c5b32F37F5beA87BDD5374eB2aC54eA8e0000000000000000000000000000
, }

// Step 1 Gas estimation let userOp =

await

getGasEstimations ( partialUserOp )

// Step 2 sign user op userOp =

await

signUserOp ( userOp )

// // Step 3: send user operation const userOpHash =

await

sendUserOp ( userOp ) ; console . log ( "userOpHash" , userOpHash ) // // Step 4: Get UserOpReceipt const receipt =

await

getUserOpReceipt ( userOpHash ) ;

} catch

( error )

{ console . error ( error ) } }

executePartialUserOp ( ) ;
```

Previous API integration Next Paymaster sponsored