Paymaster Methods

Imports needed for these methods:

import

{ IHybridPaymaster , PaymasterFeeQuote , PaymasterMode , SponsorUserOperationDto , createPaymaster , }

from

"@biconomy/account" ;

const biconomyPaymaster = smartAccount . paymaster as IHybridPaymaster < SponsorUserOperationDto

;

// Or...

const biconomyPaymaster =

await

createPaymaster ({ paymasterUrl }) ;

// Found at https://dashboard.biconomy.io After setting up your smart account as mentioned above you can start using the Paymaster.

getPaymasterFeeQuotesOrData

()

This method is particularly useful forERC20 mode and is used to get the fee quotes information for the ERC20 tokens.

Usage

const feeQuotesResponse =

await biconomyPaymaster . getPaymasterFeeQuotesOrData (userOp , { mode : PaymasterMode . ERC20 , tokenList :

["0xdA5289FCAAF71d52A80A254dA614A192B693e975"] , preferredToken :

"0xdA5289FCAAF71d52A80A254dA614A192B693e975" , }) ; As per the code

- We set the mode to ERC20
- In the token list we can specify a list of addresses of the ERC20 tokens we want to have our users pay in.
- We can also decide to choose a preferred token for the response to include.

Parameters

- userOp(Partial
- , required): A UserOperation object representing the user's request that needs to be processed.
- paymasterServiceData
- (FeeQuotesOrDataDto
- , required): The paymaster service data containing token information and sponsorship details. You can send just the preferred token or an array of token addresses.
- type
- FeeQuotesOrDataDto
- =
- {
- mode
- ?
- :
- PaymasterMode
- ;
- // enum values ERC20 and SPONSORED
- expiryDuration
- ?
- :
- number

- ;
- // Specifies the duration, in seconds, for which the user intends the paymasterAndData to remain valid, minimum duration is 300 secs.
- calculateGasLimits
- ?
- :
- boolean
- ;
- tokenList
- ?
- :
- string
- [
-]
- ;
- // If you pass tokenList as an empty array. and it
- // would return fee quotes for all tokens supported by the Biconomy paymaster
- preferredToken
- ?
- :
- string
- ;
- // If you want to pass only one token
- webhookData
- ?
- :
- Record
- <
- string
- ,
- any
-

- ;
- smartAccountInfo
- ?
- :
- SmartAccountData
- ;
- // use this to calulate the gas for smart account v1, otherwise by default it will return for latest smart account version
- }
- ;

Returns

- response
- (Promise
-): It returns a promise that resolves to the following object containing a feeQuotes Array. A paymaster Fee quote will have additional information about the fee being paid in the specified ERC20 token, how long this fee is valid, what type of premium is going to be paid, as well as general token information and the amount in USD. Here is the full typing for a single fee quote:
- type
- FeeQuotesOrDataResponse
- =
- {
- feeQuotes
- ?
- :
- PaymasterFeeQuote
- [
-]
- ;
- tokenPaymasterAddress
- ?
- :
- string
- ;
- paymasterAndData

- ?
- :
- string
- ;
- preVerificationGas
- ?
- :
- BigNumberish
- ;
- verificationGasLimit
- ?
- :
- BigNumberish
- ;
- callGasLimit
- ?
- :
- BigNumberish
- ;
- }
- ;
- type
- PaymasterFeeQuote
- =
- {
- symbol
- :
- string
- ;
- // ERC20 token symbol
- tokenAddress
- :
- string
- ;
- // ERC20 token address
- decimal
- :
- number
- ;
- // number of decimal places used to represent the token
- logoUrl
- ?
- :
- string
- ;
- // logo url of the ERC20 token
- maxGasFee
- :
- number
- ;
- maxGasFeeUSD
- ?
- :
- number
- ;
- usdPayment
- ?
- :
- number
- ;
- // the fee converted into the USD
- premiumPercentage
- :
- number
- ;
- // premium in percentage that biconomy charges for the txn, generally ranges between 7-12
- validUntil
- ?

- :
- number
- ;
- // number (epoch time in milliseconds, till these fee quotes are valid)
- }
- ;

getPaymasterAndData

()

ThegetPaymasterAndData method is used to get the information on how the transaction fees will be covered for the transactions. The returnedPaymasterAndDataResponse includes a signed string (paymasterAndData) that signifies the paymaster's commitment to covering the transaction fee.

Parameters

- userOperation (UserOperation
- , required): A UserOperation object representing the user's request that needs to be processed.
- paymasterServiceData (SponsorUserOperationDto
- , Optional): An optional parameter that allows you to provide additional data specific to the paymaster service.
- type
- SponsorUserOperationDto
- =
- {
- mode
- :
- PaymasterMode
- ;
- calculateGasLimits
- ?
- :
- boolean
- ;
- expiryDuration
- ?
- :
- number
- ;
- webhookData
- ?
- :
- {
- [
- key
- :
- string
-]
- :
- any
- ;
- }
- ;
- smartAccountInfo
- ?
- :
- SmartAccountData
- ;
- feeTokenAddress
- ?
- :
- string
- ;
- }
- ;

Returns

- paymasterAndDataResponse(Promise
-): It returns a promise that resolves to a Paymaster and Data Response object. The data in this response can be used to update the userOp to include sponsorship data or data for paying gas in ERC20 tokens.type
- PaymasterAndDataResponse
- =
- {
- paymasterAndData
- :
- string
- ;
- preVerificationGas
- ?
- :
- BigNumberish
- ;
- verificationGasLimit
- ?
- :
- BigNumberish
- ;
- callGasLimit
- ?
- :
- BigNumberish
- ;
- }
- ;

Usage Mode:SPONSORED

There are two modes for using this function:SPONSORED andERC20 .

Here it is meant to act as Sponsorship/Verifying paymaster hence we send mode: PaymasterMode.SPONSORED which is required

let paymasterServiceData : SponsorUserOperationDto =

{ mode : PaymasterMode . SPONSORED , // optional params... calculateGasLimits :

true , } ; const paymasterAndDataResponse =

await biconomyPaymaster . getPaymasterAndData (userOp , paymasterServiceData) ;

userOp . paymasterAndData = paymasterAndDataResponse . paymasterAndData ; As per the code we get the Paymaster Data Response and update the userOp, to specify that this will be a sponsored transaction by the Biconomy Paymaster.

Usage Mode:ERC20

ERC20 mode enables you to pay the gas fees of your users in exchange for ERC-20 tokens. When switching mode to ERC20 there are additional steps that need to be considered.

warning Important: When usingToken Paymaster with ERC20 tokens, always ensure to calculate thefeeQuote correctly. This is crucial to avoid transaction reverts due to insufficient token balance after execution. ThefeeQuote should consider both the transaction cost and any othertoken movements within the same operation.

Example: If a user is transacting withUSDC , and the feeQuote is2 USDC , the DApp must ensure that the user's balance post-callData execution is sufficient to cover this fee. Incorrect fee calculations can lead to transaction failures and a degraded user experience. 1. Get paymaster fee quote : We need to get the swap quotes of the ERC20 tokens that the user is paying for the native token of the network we are on.

const feeQuotesResponse =

await biconomyPaymaster . getPaymasterFeeQuotesOrData (userOp , { mode : PaymasterMode . ERC20 , tokenList :

[ ] , preferredToken :

"" , }) ; 2. Update userOp : After getting this information we need to build our updated userOp with the preferredfeeQuote from thefeeQuotes array.

finalUserOp

await smartAccount . buildTokenPaymasterUserOp (userOp ,

{ feeQuote : selectedFeeQuote , spender : spender , maxApproval :

false , }) ; 3. Create paymaster data : Now we create ourpaymasterServiceData again and call thegetPaymasterAndData method

let paymasterServiceData =

{ mode : PaymasterMode . ERC20 , feeTokenAddress : selectedFeeQuote . tokenAddress , calculateGasLimits :

true ,

// Always recommended and especially when using token paymaster } ; const paymasterAndDataWithLimits =

await biconomyPaymaster . getPaymasterAndData (finalUserOp , paymasterServiceData) ; finalUserOp . paymasterAndData = paymasterAndDataWithLimits . paymasterAndData ; tip A common error you might encounter here is the AA34 error.

Code snippet to fix AA34 try

{ const paymasterAndDataResponse =

await biconomyPaymaster . getPaymasterAndData (partialUserOp , paymasterServiceData) ; partialUserOp . paymasterAndData = paymasterAndDataResponse . paymasterAndData ;

if

(paymasterAndDataResponse . callGasLimit && paymasterAndDataResponse . verificationGasLimit && paymasterAndDataResponse . preVerificationGas)

{ // Returned gas limits must be replaced in your op as you update paymasterAndData. // Because these are the limits paymaster service signed on to generate paymasterAndData // If you receive AA34 error check here..

partialUserOp . callGasLimit = paymasterAndDataResponse . callGasLimit ; partialUserOp . verificationGasLimit = paymasterAndDataResponse . verificationGasLimit ; partialUserOp . preVerificationGas = paymasterAndDataResponse . preVerificationGas ; } }

catch

(e)

{ console . log ("error received " , e) ; }

Previous Integration Next APIs