**Introduction to Paymaster API**

The basics of Stackup's Paymaster API

[Suggest Edits](https://docs.stackup.sh/edit/paymaster-api)

Account abstraction with ERC-4337 enable special entities called Paymasters. These are third-parties that can sponsor the gas for an account if their conditions are met. This enables account holders to pay for transaction fees in many different ways without resulting to custodial solutions.

**This page covers v0.6+ paymasters. For documentation about v0.4 paymasters, see**[**Legacy RPC Methods**](https://docs.stackup.sh/docs/paymaster-api-legacy-rpc-methods)**.**

**Stackup Paymasters**

Stackup's paymasters allow users to sponsor transactions or accept ERC-20 tokens for gas payment. The full RPC methods can be found in [Paymaster RPC Methods](https://docs.stackup.sh/docs/paymaster-api-rpc-methods). Stackup charges a few of 5% of the gas price to sponsor transactions.

**Sponsored Transactions**

Using the payg paymaster type, you can sponsor transactions directly for users. Stackup will fully cover gas for valid UserOperations and the value will be billed to your monthly invoice with Stackup.

**ERC-20 Tokens**

The erc20token paymaster type allows you to use ERC-20 tokens for gas payments. Stackup will fully cover gas for valid UserOperations and the Stackup paymaster will withdraw the appropriate ERC-20 token amount at the end of the UserOperation.

ERC-20 withdrawal occurs during the postOp method. Stackup has various protections to ensure that the Smart Account's ERC-20 token balance does not change between the paymaster agreeing to sponsor the transaction and withdrawing the ERC-20 token. **In the rare case a Smart Account bypasses these checks to exploit the paymaster, Stackup will treat it as a sponsored transaction and add the unpaid gas to your monthly invoice.**

**Using userop.js**

The easiest way to use paymasters is with the userop.js library to call the [Paymaster RPC Methods](https://docs.stackup.sh/docs/paymaster-api-rpc-methods). As a UserOperation is built using buildOp, the verifyingPaymaster middleware can be used to retrieve the paymasterAndData and required gas limits. See the [verifyingPaymaster](https://docs.stackup.sh/docs/useropjs-presets" \l "verifyingpaymaster" \t "_self) function for more details and the [examples repository](https://github.com/stackup-wallet/erc-4337-examples) for an implementation.

**Use a Paymaster**

Open Recipe

**Endpoints**

| **Node Version** | **Endpoint** |
| --- | --- |
| **v0.6.x** | https://api.stackup.sh/v1/paymaster/API\_KEY |
| **v0.4.x (legacy PAYG)** | https://app.stackup.sh/api/v2/paymaster/payg/API\_KEY |

**Enable Paymasters**

Paymasters are included by default in the Developer and Enterprise Plans. This can be enabled on the billing page in the [Stackup dashboard](https://app.stackup.sh/dashboard" \t "_self).

Updated 9 days ago

**Paymaster RPC Methods**

JSON-RPC API reference for Stackup's Verifying Paymaster service.

[Suggest Edits](https://docs.stackup.sh/edit/paymaster-api-rpc-methods)

All of Stackup's Paymaster API endpoints are listed here. pm is the Paymaster namespace used by Stackup and implements the proposed interface outlined [here](https://hackmd.io/@stackup/H1oIvV-qi).

**pm\_sponsorUserOperation**

This methods sends a User Operation to a Paymaster for off-chain verification. If approved, it will return the paymasterAndData and updated gas values which can be appended to the User Operation before signing.

If the Paymaster rejects the User Operation it should not return a result but a standard JSON-RPC error with the reason.

**Parameters (in order)**

* **UserOperation**: This is a User Operation with a valid dummy signature.
* **entryPoint**: The EntryPoint address that the User Operation is intended for.
* **context**: This argument contains information about the specific Paymaster implementation you are using.

Request

{

"jsonrpc": "2.0",

"id": 1,

"method": "pm\_sponsorUserOperation",

"params": [

{

sender, // address

nonce, // uint256

initCode, // bytes

callData, // bytes

callGasLimit, // uint256

verificationGasLimit, // uint256

preVerificationGas, // uint256

maxFeePerGas, // uint256

maxPriorityFeePerGas, // uint256

paymasterAndData, // bytes

signature, // Can be a valid dummy value

},

"0x0576a174D229E3cFA37253523E645A78A0C91B57",

{ /\* PM specific data here... \*/ }

]

}

Response (success)

{

"jsonrpc": "2.0",

"id": 1,

"result": {

"paymasterAndData": "0x1234...5678",

"preVerificationGas": "0x...",

"verificationGasLimit": "0x...",

"callGasLimit": "0x...",

}

}

Response (error)

{

"jsonrpc": "2.0",

"id": 1,

"error": {

"message": "Error reason here.",

"code": -32601

}

}

**Sponsored transactions**

Stackup's pay-as-you-go paymaster (PAYG) allows you to sponsor User Operations for users directly. Your Stackup account will be billed at the end of the month. To use this paymaster, simply add { "type": "payg" } to the context field when making a request:

ContextExample Request

{

"type": "payg"

}

**ERC-20 transactions**

Similarly, you can use ERC-20 tokens for gas payment. For now Stackup accepts USDC for mainnet transactions and test ERC-20 tokens for testnet transactions.

To use this paymaster, simply add {"type": "erc20token", "token": "0x000..."} to the context field when making a request.

ContextExample Request

{

"type": "erc20token",

"token": "USDC\_ADDRESS" | "TEST\_TOKEN\_ADDRESS"

}

The testnet ERC-20 token address is [0x3870419Ba2BBf0127060bCB37f69A1b1C090992B](https://blockscan.com/address/0x3870419Ba2BBf0127060bCB37f69A1b1C090992B). You may mint up to 1 ETH of test tokens at a time using the block scanner interface.

**pm\_accounts**

This method allows clients to get all the Paymaster addresses associated with an EntryPoint that’s owned by this service. The first address in the returned array is the preferred paymaster contract.

This is useful for use cases where a client application is required to know the paymaster address to create certain transactions like an ERC-20 approve.

RequestResponse

{

"jsonrpc": "2.0",

"id": 1,

"method": "pm\_accounts",

"params": [

entryPoint // string

]

}

Updated 3 months ago