2

callWithSyncFee

Permissionless transactions with on-chain payments After reading this page:

- You'll know how to use thecallWithSyncFee
- SDK method, using thesyncFee
- payment method.
- You'll see some code which will help you send a relay request within minutes.
- You'll learn how to pay for transactions using the provided values forfee
- ,feeToken
- andfeeCollector
- . * Please proceed to ourSecurity Considerations page and read it thoroughly before advancing with your implementation. It is crucial to understand all potential security risks and measures to mitigate them.

Overview

ThecallWithSyncFee method uses thesyncFee payment method.

Paying for Transactions

When usingcallWithSyncFee relay method the target contract assumes responsibility for transferring the fee to Gelato's fee collector during transaction execution. For this, the target contract needs to know:

- fee
- : the transfer amount
- feeToken
- : the token to be transferred
- feeCollector
- : the destination address for the fee
-

Fortunately, Gelato provides some useful tools within theRelay Context Contracts :

1. By inheriting theGelatoRelayContext
2. contract in your target contract, you have the ability to transfer the fee through one of two straightforward methods:_transferRelayFee()
3. or_transferRelayFeeCapped(uint256 maxFee)
4. . In either case, the inherited contract takes care of decoding thefee
5. ,feeToken
6. , andfeeCollector
7. behind the scenes.
8. The Gelato Relay backend simplifies the process by automatically calculating the fee for you, using Gelato's Fee Oracle to perform the calculations in the background.
9. Alternatively, you may choose to inherit theGelatoRelayFeeCollector
10. contract. With this approach, Gelato only decodes thefeeCollector
11. . You must provide thefee
12. andfeeToken
13. on-chain, either by hardcoding them (which is not recommended) or embedding them within the payload to be executed. The suggested way to handle this is to calculate the fee withGelato's Fee Oracle
14. .
15.

SettingmaxFee for Your Transaction

Setting a maximum fee, ormaxFee , for your transactions is strongly advised. This practice enables you to ensure that transaction costs remain below a specific limit. The method_transferRelayFeeCapped(uint256 maxFee) in theGelatoRelayContext contract provides a convenient way to set themaxFee easily.

If you are utilizing theGelatoRelayFeeCollector contract, the recommended way to pass themaxFee is by calculating the fee withGelato's Fee Oracle , which is accessible in therelay-sdk . ThegetEstimatedFee() method is provided to facilitate this calculation.

SDK method: callWithSyncFee

```

Copy constcallWithSyncFee=async( request:CallWithSyncFeeRequest, options?:RelayRequestOptions, apiKey?:string ):Promise
```

```

Arguments:

- request
- : this is therequest body
- used to send a request.
- options
- :RelayRequestOptions
- is an optional object.
- apiKey
- : this is an optional API key that links your request to your Gelato Relay account. As this pertains to thesyncFee
- payment method, transaction costs won't be deducted from your 1Balance account. By using the API key, you can benefit from increased rate limits of your Gelato Relay account.
-

Return Object: RelayResponse

```

Copy typeRelayResponse={ taskId:string; };

```

- taskId
- : your unique relay task ID which can be used fortracking your request
- .
-

Optional Parameters

SeeOptional Parameters .

Sending a Request

Request Body

```

Copy constrequest={ chainId:BigNumberish; target: string; data: BytesLike; isRelayContext?:boolean; feeToken: string; };

```

- chainId
- : the chain ID of the chain where thetarget
- smart contract is deployed.
- target
- : the address of the target smart contract.
- data
- : encoded payload data (usually a function selector plus the required arguments) used to call the requiredtarget
- address.
- isRelayContext
- : an optional boolean (default:true
- ) denoting what data you would prefer appended to the end of the calldata.
-
    - If set totrue
-
    - (default), Gelato Relay will append thefeeCollector
-
    - address, thefeeToken
-
    - address, and the uint256fee
-
    - to the calldata. This requires the target contract to inherit theGelatoRelayContext
-
    - contract.
-
    - If set tofalse
-
```

    - , Gelato Relay will only append thefeeCollector
-
    - address to the calldata. In this case the contract to be inherit by the target contract is the[GelatoRelayFeeCollector](#)
-
    - .
- *
- feeToken
- : the address of the token that is to be used for payment. Please visit[SyncFee Payment Tokens](#)
- for the full list of supported payment tokens per network.
-

Example Code GelatoRelayContext

1. Deploy a GelatoRelayContext compatible contract

```

Copy // SPDX-License-Identifier: MIT pragmasolidity0.8.17;

import{ GelatoRelayContext }from"@gelatonetwork/relay-context/contracts/GelatoRelayContext.sol"; import{Address}from"@openzeppelin/contracts/utils/Address.sol";

// Inheriting GelatoRelayContext gives access to: // 1. _getFeeCollector(): returns the address of Gelato's feeCollector // 2. _getFeeToken(): returns the address of the fee token // 3. _getFee(): returns the fee to pay // 4. _transferRelayFee(): transfers the required fee to Gelato's feeCollector.abi // 5. _transferRelayFeeCapped(uint256 maxFee): transfers the fee to Gelato // only if fee < maxFee // 6. _getMsgData(): returns the original msg.data without appended information // 7. onlyGelatoRelay modifier: allows only Gelato Relay's smart contract // to call the function contractCounterRelayContextisGelatoRelayContext{ usingAddressforaddresspayable;

uint256publiccounter;

eventIncrementCounter(uint256newCounterValue);

// increment is the target function to call. // This function increments a counter variable by 1 // IMPORTANT: withcallWithSyncFee you need to implement // your own smart contract security measures, as this // function can be called by any third party and not only by // Gelato Relay. If not done properly, funds kept in this // smart contract can be stolen. functionincrement()externalonlyGelatoRelay{ // Remember to autheticate your call since you are not using ERC-2771 // _yourAuthenticationLogic()

// Payment to Gelato //NOTE: be very careful here! // if you do not use the onlyGelatoRelay modifier, // anyone could encode themselves as the fee collector // in the low-level data and drain tokens from this contract. _transferRelayFee();

counter++;

emitIncrementCounter(counter); }

// incrementFeeCapped is the target function to call. // This function uses_transferRelayFeeCapped method to ensure // better control of gas fees. If gas fees are above the maxFee value // the transaction will not be executed. // This function increments a counter variable by 1 // IMPORTANT: with callWithSyncFee you need to implement // your own smart contract security measures, as this // function can be called by any third party and not only by // Gelato Relay. If not done properly, funds kept in this // smart contract can be stolen. functionincrementFeeCapped(uint256maxFee)externalonlyGelatoRelay{ // Remember to autheticate your call since you are not using ERC-2771 // _yourAuthenticationLogic()

// Payment to Gelato //NOTE: be very careful here! // if you do not use the onlyGelatoRelay modifier, // anyone could encode themselves as the fee collector // in the low-level data and drain tokens from this contract.

_transferRelayFeeCapped(maxFee);

counter++;

emitIncrementCounter(counter); } }

```

1. Import GelatoRelaySDK into your front-end .js project

```

Copy import{ GelatoRelay,CallWithSyncFeeRequest }from"@gelatonetwork/relay-sdk"; constrelay=newGelatoRelay();

```

1. Send the payload to Gelato

```

Copy // set up target address and function signature abi constcounter="'; constabi=["function increment()"];

// generate payload using front-end provider such as MetaMask
constprovider=newethers.BrowserProvider(window.ethereum); constsigner=provider.getSigner();

// address of the token to pay fees constfeeToken="0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE";

// instantiate the target contract object constcontract=newethers.Contract(counterAddress,abi,signer);

// example callig the increment() method const{data}=awaitcontract.populateTransaction.increment();

// populate the relay SDK request body constrequest:CallWithSyncFeeRequest={ chainId:
(awaitprovider.getNetwork()).chainId, target:counter, data:data, feeToken:feeToken, isRelayContext:true, };

// send relayRequest to Gelato Relay API constrelayResponse=awaitrelay.callWithSyncFee(request);

// ---------- // the following is an alternative example using Gelato Fee Oracle, // setting
maxFee, and calling the incrementFeeCapped(maxFee) method

// retrieve the estimate fee from the Gelato

constfee=awaitrelay.getEstimatedFee( (awaitprovider.getNetwork()).chainId, feeToken, gasLimit, false, )

constmaxFee=fee*2// you can use 2x or 3x to set your maxFee

// example calling the incrementFeeCapped(maxFee) method
const{dataMaxFee}=awaitcontract.incrementFeeCapped.populateTransaction(maxFee);

// populate the relay SDK request body constrequestMaxFee:CallWithSyncFeeRequest={ chainId:
(awaitprovider.getNetwork()).chainId, target:counter, data:dataMaxFee, feeToken:feeToken, isRelayContext:true, };

// send relayRequest to Gelato Relay API constrelayResponse=awaitrelay.callWithSyncFee(requestMaxFee);

```

Previous sponsoredCall Next Relay Context Contracts Last updated2 days ago On this page *Overview * Paying for Transactions * Setting maxFee for Your Transaction * SDK method: callWithSyncFee * Sending a Request * Request Body * Example Code GelatoRelayContext * 1. Deploy a GelatoRelayContext compatible contract * 2. Import GelatoRelaySDK into your front-end .js project * 3. Send the payload to Gelato