2

callWithSyncFeeERC2771

Transactions with on-chain payments and ERC2771 authentication support If you plan to use ERC-2771 with amulticall method or any other method usingdelegateCall() Please read carefully the sectionAvoid ERC-2771-risks If you are using@gelatonetwork/relay-sdk v3 or contracts from the package@gelatonetwork/relay-context v2 please follow thismigration guide to migrate to the new versions. After reading this page:

- You'll know how to use thecallWithSyncFeeERC2771
- 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

ThecallWithSyncFeeERC2771 method uses thesyncFee payment method withERC-2771 support.

Paying for Transactions

When usingcallWithSyncFeeERC2771 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 theGelatoRelayContextERC2771
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 theGelatoRelayFeeCollectorERC2771
10. contract. With this approach, Gelato decodes only 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.

This modular design ensures a smooth integration with Gelato's fee handling mechanisms, providing a flexible and user-friendly approach to managing transaction fees within your dApps.

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 theGelatoRelayContextERC2771 contract provides a convenient way to set themaxFee easily.

If you are utilizing theGelatoRelayFeeCollectorERC2771 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 Methods

callWithSyncFeeERC2771

This method initiates the signing of ERC2771 requests with the providedBrowserProvider orWallet . Once the signature is obtained, the request is forwarded to Gelato.

```

Copy constcallWithSyncFeeERC2771=async( request:CallWithSyncFeeERC2771Request|CallWithSyncFeeConcurrentERC2771Request, signerOrProvider:ethers.BrowserProvider|ethers.Signer, options?:RelayRequestOptions, apiKey?:string ):Promise

```

Arguments

- request
- : Thebody
- of the request intended for sending.
- signerOrProvider
- : a valid provider connected to RPC or a signer.
- options
- : an object for specifyingoptional parameters
- .
- apiKey
- : an optional API key that links your request to your Gelato Relay account. As this call 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.
-

Response

```

Copy typeRelayResponse={ taskId:string; };

```

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

getSignatureDataERC2771

This method starts the signing process for ERC2771 requests using the givenBrowserProvider orSigner . After capturing the signature, it returns both the signature and the message. This collected data can then be used with thecallWithSyncFeeERC2771WithSignature method to send the request to Gelato.

```

Copy getSignatureDataERC2771=( request:CallWithERC2771Request|CallWithConcurrentERC2771Request, signerOrProvider:ethers.BrowserProvider|ethers.Signer, type:ERC2771Type ):Promise

```

Arguments

- request
- : Thebody
- of the request intended for sending.
- signerOrProvider
- : a valid provider connected to RPC or a signer.
- type
- :CallWithSyncFee
- for asequential
- flow orConcurrentCallWithSyncFee
- for aconcurrent
- flow.
-

Response

```

Copy typeSignatureData=ConcurrentSignatureData |SequentialSignatureData;

typeConcurrentSignatureData={ struct:CallWithConcurrentERC2771Struct; signature:string; };

typeSequentialSignatureData={ struct:CallWithERC2771Struct; signature:string; };

```

- struct
- : EIP-712 message data.
- signature
- : EIP-712 signature.
-

getDataToSignERC2771

This method provides the message data intended for external signing along with the EIP-712 typed data. After obtaining the signature, the request can be dispatched using thecallWithSyncFeeERC2771WithSignature method.

```

Copy getDataToSignERC2771=( request:CallWithERC2771Request|CallWithConcurrentERC2771Request,
type:ERC2771Type, signerOrProvider?:ethers.BrowserProvider|ethers.Signer, ):Promise

```

Arguments

- request
- : Thebody
- of the request intended for sending.
- type
- :CallWithSyncFee
- for asequential
- flow or ConcurrentCallWithSyncFee for aconcurrent
- flow.
- signerOrProvider
- (optional): A provider needed in a sequential flow to obtain the nonce from the smart contract. If you're providing the nonce within your request or if you're using the concurrent flow, this parameter isn't necessary.
-

Response

```

Copy typePayloadToSign=ConcurrentPayloadToSign|SequentialPayloadToSign;

typeConcurrentPayloadToSign={ struct:CallWithConcurrentERC2771Struct;
typedData:CallWithSyncFeeConcurrentERC2771PayloadToSign; };

typeSequentialPayloadToSign={ struct:CallWithERC2771Struct; typedData:CallWithSyncFeeERC2771PayloadToSign; };

```

- struct
- : EIP-712 message data.
- typedData
- : EIP-712 typed data.
-

callWithSyncFeeERC2771WithSignature

This method sends pre-signed requests to Gelato.

```

Copy constcallWithSyncFeeERC2771WithSignature=async(
struct:CallWithERC2771Struct|CallWithConcurrentERC2771Struct; syncFeeParams:BaseCallWithSyncFeeParams;
```

signature:string; options?:RelayRequestOptions; apiKey?:string):Promise

```

Arguments

- struct
- : EIP-712 message data returned from the signing methods.
- syncFeeParams:
- thefeetoken
- andisRelayContext
- params.
- signature:
- EIP-712 signature returned after signing the request.
- options
- : an object for specifyingoptional parameters
- .
- apiKey
- : an optional API key that links your request to your Gelato Relay account. As this call 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.
-

Response

```

Copy typeRelayResponse={ taskId:string; };

```

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

Optional Parameters

SeeOptional Parameters .

Sending a Request

As of today, we support two distinct ways of sendingcallWithSyncFeeERC2771 requests:

1. Sequentially
2. : This approach ensures that each request is ordered and validated against thenonce
3. stored on-chain. You have two options in this method:
4.
    - Fetch the currentnonce
5.
    - value from the smart contract yourself and include it with your request.
6.
    - Allow the relay-sdk to fetch thenonce
7.
    - value for you when handling your relay request.
8. *
9. Concurrently
10. : This method enables you to send multiple transactions simultaneously. Replay protection is achieved using a hash-basedsalt
11. mechanism. Again, you have two options:
12.
    - Provide your ownsalt
13.
    - value.
14.
    - Allow the relay-sdk to generate a uniquesalt
15.
    - value for you when processing your relay request.
16. *
17.
```

By defaultcallWithSyncFeeERC2771 requests are using the sequential method.

Concurrent ERC2771 support has been introduced in the relay-sdk version5.1.0 . Please make sure that your package is up-to-date to start using it.

Request Body

```

Copy typeSequentialERC2771Request={ chainId:BigNumberish; target:string; data:BytesLike; user:string; userDeadline?:BigNumberish; feeToken:string; isRelayContext?:boolean; isConcurrent?:false; userNonce?:BigNumberish; }; typeConcurrentERC2771Request={ chainId:BigNumberish; target:string; data:BytesLike; user:string; userDeadline?:BigNumberish; feeToken:string; isRelayContext?:boolean; isConcurrent:true; userSalt?:string; };

```

Common Parameters

- 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.
- user
- : the address of the user's EOA.
- userDeadline
- :optional,
- the amount of time in seconds that a user is willing for the relay call to be active in the relay backend before it is dismissed.
-
    - This way the user knows that if the transaction is not sent within a certain timeframe, it will expire. Without this, an adversary could pick up the transaction in the mempool and send it later. This could transfer money, or change state at a point in time which would be highly undesirable to the user.
- *
- feeToken
- : the address of the token that is to be used for payment. Please visitSyncFee Payment Tokens
- for the full list of supported payment tokens per network.
- 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. In this case your target contract should inherit from theGelatoRelayContextERC2771
-
    - contract.
-
    - If set tofalse
-
    - , Gelato Relay will only append thefeeCollector
-
    - address to the calldata. In this case your target contract should inherit from theGelatoRelayFeeCollectorERC2771
-
    - contract.
- *
-

Parameters For Sequential Requests:

- isConcurrent

- :false
- (default)
- ,optional,
- represents that the users' requests are validated based on a nonce, which enforces them to be processed sequentially.
- userNonce
- :optional
- , this nonce, akin to Ethereum nonces, is stored in a local mapping on the relay contracts. It serves to enforce the nonce ordering of relay calls if the user requires sequential processing. If this parameter is omitted, the relay-sdk will automatically query the current value on-chain.
-

Parameters For Concurrent Requests:

- isConcurrent
- :true
- , indicates that the users' requests are validated based on a unique salt, allowing them to be processed concurrently. Replay protection is still ensured by permitting each salt value to be used only once.
- userSalt
- :optional
- , this is a bytes32 hash that is used for replay protection. If the salt is not provided then relay-sdk would generate a unique value based on a random seed and a timestamp.
-

Example Code (usingGelatoRelayContextERC2771)

1. Deploy aGelatoRelayContextERC2771 compatible contract

```

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

import{ GelatoRelayContextERC2771 }from"@gelatonetwork/relay-context/contracts/GelatoRelayContextERC2771.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. function _getMsgSender(): decodes and returns the user's address from the // calldata, which can be used to refer to user safely instead of msg.sender // (which is Gelato Relay in this case). // 7. _getMsgData(): returns the original msg.data without appended information // 8. onlyGelatoRelay modifier: allows only Gelato Relay's smart contract // to call the function contractCounterRelayContextERC2771isGelatoRelayContextERC2771{ usingAddressforaddresspayable;

mapping(address=>uint256)publiccontextCounter;

// emitting an event for testing purposes eventIncrementCounter(addressmsgSender);

// increment is the target function to call. // This function increments a counter variable which is // mapped to every _getMsgSender(), the address of the user. // This way each user off-chain has their own counter // variable on-chain. functionincrement()externalonlyGelatoRelayERC2771{ // 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();

// Incrementing the counter mapped to the _getMsgSender() contextCounter[_getMsgSender()]++;

emitIncrementCounter(_getMsgSender()); }

// 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. // The maxFee will be passed as an argument to the contract call. // 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)externalonlyGelatoRelayERC2771{

// 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);

// Incrementing the counter mapped to the _getMsgSender() contextCounter[_getMsgSender()]++;
```

emitIncrementCounter(counter); } }

```

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

```

Copy import{ GelatoRelay,CallWithSyncFeeERC2771Request }from"@gelatonetwork/relay-sdk"; constrelay=newGelatoRelay();

```

1. Send the payload to Gelato

```

Copy // target contract address constcounter="";

// using a human-readable ABI for generating the payload constabi=["function increment()"];

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

// connect to the blockchain via a front-end provider constprovider=newethers.BrowserProvider(window.ethereum); constsigner=awaitprovider.getSigner(); constuser=awaitsigner.getAddress();

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

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

// populate the relay SDK request body constrequest:CallWithSyncFeeERC2771Request={ chainId: (awaitprovider.getNetwork()).chainId, target:counter, data:data, user:user, feeToken:feeToken, isRelayContext:true, };

// send relayRequest to Gelato Relay API constrelayResponse=awaitrelay.callWithSyncFeeERC2771(request,provider);

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

// retrieve the estimate fee from Gelato Fee Oracle constfee=awaitrelay.getEstimatedFee((awaitprovider.getNetwork()).chainId, feeToken, gasLimit, false,)

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

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

// populate the relay SDK request body constrequestMaxFee:CallWithSyncFeeERC2771Request={ chainId: (awaitprovider.getNetwork()).chainId, target:counter, data:dataMaxFee, user:user, feeToken:feeToken, isRelayContext:true, };

// send relayRequest to Gelato Relay API constrelayResponseMAxFee=awaitrelay.callWithSyncFeeERC2771(requestMaxFee,provider);

```

Previous sponsoredCallERC2771 Next Relay Context Contracts ERC2771 Last updated1 day ago On this page *Overview * Paying for Transactions * Setting maxFee for Your Transaction * SDK Methods * callWithSyncFeeERC2771 * getSignatureDataERC2771 * getDataToSignERC2771 * callWithSyncFeeERC2771WithSignature * Optional Parameters * Sending a Request * Example Code (using GelatoRelayContextERC2771) * 1. Deploy a GelatoRelayContextERC2771 compatible contract * 2. Import GelatoRelaySDK into your front-end .js project* 3. Send the payload to Gelato
```