Relay Context Contracts

Getting your smart contracts compatible with Gelato Relay's callWithSyncFee After reading this page:

- You'll learn how to use our helper functions to get useful information from directly within your target contract.
- These allow you access to thefeeCollector
- ,feeToken
- address , andfee
- amount from within your target contract. * When usingcallWithSyncFee , you need to pay Gelato's fee collecting contract when your target function is called, otherwise your relay request will not execute. To carry out this payment, your target contract needs to know the address of the fee collector so that you can transfer funds during the call. Furthermore, you need to know in what token to pay the fee in, and how much to pay the fee collector.

Gelato Relay appends this useful information to the end of thecalldata , when using thecallWithSyncFee SDK method. Gelato Relay's Context contracts give you helper functions which you can use via inheritance in your target contract allowing you to decode information from the relaycalldata , giving you access to:

- uint256_getFee()
- : a value denoting how much fee to pay.
- address_getFeeToken()
- : the address of the token the fee is paid in.
- address_getFeeCollector()
- : the address to which to send your payment.
-

NOTE :

- If you need target function needs to know all three variables from the relay calldata, see
- GelatoRelayContext
- .
- If you only need the
- feeCollector
- address (i.e. you already encode the
- fee
- and
- feeToken
- inside your own function parameters), see
- GelatoRelayFeeCollector
- .
-

Getting Started

Installing relay-context

relay-context is an extremely simple way to create a Gelato Relay compatible smart contract, with just one import.

?

Terminal

Note : please make sure to use version v3.0.0 and above.

```
Copy npm install --save-dev @gelatonetwork/relay-context
```

or

```
Copy yarn add -D @gelatonetwork/relay-context
```

Smart Contract

```

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

```

forGelatoRelayContext .

OR:

```

Copy import{ GelatoRelayFeeCollector }from"@gelatonetwork/relay-context/contracts/GelatoRelayFeeCollector.sol";

```

forGelatoRelayFeeCollector .

GelatoRelayContext

GelatoRelayContext allows your smart contract to retrieve the following information from the relay calldata :

1. Gelato's fee collector address
2. , a contract specifically deployed for collecting relay fees securely. This allows a smart contract to transfer fees directly if you are using thesyncFee
3. payment method.
4. Thefee token address
5. specifying which address the fee will be paid in, which Gelato resolved from the original relay-SDK request body.
6. Thefee
7. itself, which includes the gas cost + Gelato'sfee
8. on top.
9.

Below is an example:

```

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

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

// Inheriting GelatoRelayContext gives access to: // 1. onlyGelatoRelay modifier // 2. payment methods, i.e. _transferRelayFee // 3. _getFeeCollector(), _getFeeToken(), _getFee() contractCounterisGelatoRelayContext{ uint256publiccounter;

eventIncrementCounter();

// increment is the target function to call // this function increments a counter variable after payment to Gelato 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(); } }

```

Line 12 inherits theGelatoRelayContext contract, giving access to these features:

Verifying the caller:

- onlyGelatoRelay
- : amodifier
- which will only allow Gelato Relay to call this function.
- _isGelatoRelay(address _forwarder)
- : a function which returns true if the address matches Gelato Relay's address.
-

Decoding thecalldata :

- _getFeeCollector()
- : a function to retrieve the fee collector address.
- _getFee()
- : a function to retrieve the fee that Gelato will charge.
- _getFeeToken()
- : a function to retrieve the address of the token used for fee payment.
-

Transferring Fees to Gelato:

As you are using the[callWithSyncFee](#) SDK method, you can use the below helper functions to pay directly to Gelato:

- _transferRelayFee()
- : a function which transfers the fee amount to Gelato, with no cap.
- _transferRelayFeeCapped(uint256 _maxFee)
- : a function which transfers the fee amount to Gelato which a set cap from the argumentmaxFee
- inwei
- . This helps limit fees on function calls in case of gas volatility or just for general budgeting.
-

GelatoRelayFeeCollector

Why are there two different contracts that I can inherit?

You can choose to inherit either[GelatoRelayContext](#) or[GelatoRelayFeeCollector](#) .GelatoRelayContext gives you access to all three pieces of information:feeCollector ,feeToken , andfee , whereasGelatoRelayFeeCollector

assumes only thefeeCollector address is appended to the calldata.

Which contract should I inherit?

In the majority of scenarios, inheriting fromGelatoRelayContext is recommended. This approach provides the most convenient way to handle fees, as it only requires you to call either the_transferRelayFee() or_transferRelayFeeCapped(uint256 _maxFee) method. All other processes are managed seamlessly behind the scenes.

If the fee is known in advance - for instance, if you have already queried our[fee oracle](#) for the fee amount and a user has given their approval on this amount and the token to be used for payment via a front-end interface - you would only need to inform your smart contract where to direct this fee. In this case, you would require only thefeeCollector address. For this purpose, please inherit fromGelatoRelayFeeCollector . Refer to the following details.

Recommendation: maximum fee signing

The[fee oracle](#) allows you to query and display a fee to your users before sending their transaction via Gelato Relay. Therefore, you could also give them the option to sign off on a certain fee. In this case, you might want to pass the fee you receive from the oracle directly to your target function as an argument.

This makes sense, but be wary that due to gas price volatility, a smoother UX might be to query the fee oracle and calculate amaximum fee by adding some overhead, and displaying thismaximum fee to the user. This way, if gas prices do fluctuate more than normal, you can be certain that your user's transaction is executed. You can choose to set a very large overhead, or a smaller one, depending on your own trade-off between execution likelihood and cost to the user. This way, you can also integrate a fee check from inside target function to avoid any overpayments.

GelatoRelayFeeCollector Integration

GelatoRelayFeeCollector

allows your smart contract to retrieve the following information from the relaycalldata :

- Gelato's fee collector address
- , a contract specifically deployed for collecting relay fees securely. This allows a smart contract to transfer fees directly if you are using thesyncFee
- payment method.
-

Below is an example:

```

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

import{IERC20}from"@openzeppelin/contracts/token/ERC20/IERC20.sol";
```

```
import{Address}from"@openzeppelin/contracts/utils/Address.sol"; import{ SafeERC20
}from"@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol"; import{ GelatoRelayFeeCollector
}from"@gelatonetwork/relay-context/contracts/GelatoRelayFeeCollector.sol";

contractCounterisGelatoRelayFeeCollector{ usingSafeERC20forIERC20; uint256publiccounter;

eventIncrementCounter();

// increment is the target function to call // this function increments a counter variable after payment to Gelato
functionincrement()externalonlyGelatoRelay{ // Remember to autheticate your call since you are not using ERC-2771 //
_yourAuthenticationLogic()

// Retreiving the feeCollector address, using the nativeToken addressfeeCollector=_getFeeCollector();
addressnativeToken="0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE"; // Hardcoding the fee to 100 wei -NOTE:
this is just for example // If you do not pay enough to feeCollector, // your relay request will not go through // In reality, you
should pass in user signaturesTODO uint256fee=100;

// 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.
transfer(nativeToken,feeCollector,fee);

counter++;

emitIncrementCounter(); }

functiontransfer( address_token, address_to, uint256_amount )internal{ if(_amount==0)return; _token==NATIVE_TOKEN ?
Address.sendValue(payable(_to),_amount) :IERC20(_token).safeTransfer(_to,_amount); } }
```

Line 13 inherits theGelatoRelayFeeCollector

contract, giving access to these features:

Verifying the caller:

- onlyGelatoRelay
- : amodifier
- which will only allow Gelato Relay to call this function.
- _isGelatoRelay(address _forwarder)
- : a function which returns true if the address matches Gelato Relay's address.
-

Decoding the calldata:

- _getFeeCollector()
- : a function to retrieve the fee collector address.
-

Previous callWithSyncFee Next Gelato's Fee Oracle Last updated6 months ago On this page *Getting Started * Installing relay-context * GelatoRelayContext * GelatoRelayFeeCollector * Why are there two different contracts that I can inherit?* Which contract should I inherit?* GelatoRelayFeeCollector Integration