How to manage the fee collector addresses of your Orbit chain

PUBLIC PREVIEW, MAINNET READY Orbit chains are nowMainnet ready ! Note that Orbit is still apublic preview capability - the Orbit product and its supporting documentation may change significantly as we capture feedback from readers like you.

To provide feedback, click theRequest an update button at the top of this documentjoin the Arbitrum Discord , or reach out to our team directly by completingthis form . As part of the activity of an Orbit chain, different fees are collected with every transaction. These fees are collected as a single amount (the transaction fees) but are internally split into different components depending on their purpose. Each component can also be transferred to a different fee collector address that can be configured on your chain.

This guide describes the different fees that are collected, and explains how to specify the fee collector address on your chain for each fee type.

What fees are collected on an Orbit chain?

There are four fee types that are collected on every transaction of an Orbit chain:

- Orbit base fee
- : fees paid for executing the transaction on the chain based on the minimum base price configured.
- Orbit surplus fee
- : if the chain is congested (i.e., the base price paid for the transaction is higher than the minimum base price), these fees account for executing the transaction on the chain based on any gas price paid above the minimum base price configured.
- Parent chain base fee
- : relative fees paid for posting the transaction on the parent chain. This amount is calculated based on the transaction's estimated size and the current view of the parent chain's base fee.
- Parent chain surplus fee
- : if configured, these are extra fees rewarded to the batch poster.

You can find more detailed information about these fee types in these pages:

- L2 fees
- for the Orbit base fee and surplus fee
- L1 fees
- for the Parent chain base fee and surplus fee

How to configure the fee collector addresses?

Let's now look at how to configure the collector addresses for each fee type.

Orbit base fee

Orbit base fees are paid to theinfraFeeAccount configured in your chain. You can retrieve the current configured address by calling the methodgetInfraFeeAccount() of theArbOwnerPublic precompile. For example:

cast call --rpc-url ORBIT_CHAIN_RPC 0x000000000000000000000000000000000000006B "getInfraFeeAccount() (address)" Note: TheArbOwner precompile also has agetInfraFeeAccount() method that can be used, but only by the owner of the chain.

Alternatively, you can use the Orbit SDK to retrieve the current address configured asinfraFeeAccount , by calling theArbOwner precompile:

const orbitChainClient =

createPublicClient ({ chain :

< OrbitChainDefinition

, transport :

http ( ) , }) . extend (arbOwnerPublicActions) ;

const infraFeeAccount =

await orbitChainClient . arbOwnerReadContract ({ functionName :

'getInfraFeeAccount' , }) ; To set a newinfraFeeAccount , use the methodsetInfraFeeAccount(address) of theArbOwner precompile. For example:

cast send --rpc-url ORBIT_CHAIN_RPC --private-key OWNER_PRIVATE_KEY 0x0000000000000000000000000000000000000070 "setInfraFeeAccount(address) ()"

NEW_INFRAFEEACCOUNT_ADDRESS Or using the Orbit SDK:

const owner =

privateKeyToAccount (< OwnerPrivateKey

) ; const orbitChainClient =

createPublicClient ({ chain :

< OrbitChainDefinition

, transport :

http ( ) , }) . extend (arbOwnerPublicActions) ;

const transactionRequest =

await orbitChainClient . arbOwnerPrepareTransactionRequest ({ functionName :

'setInfraFeeAccount' , args :

[< NewInfraFeeAccountAddress

] , upgradeExecutor :

false , account : owner . address , }) ;

await orbitChainClient . sendRawTransaction ({ serializedTransaction :

await owner . signTransaction (transactionRequest) , }) ;

Orbit surplus fee

Orbit surplus fees are paid to thenetworkFeeAccount configured in your chain. You can retrieve the current configured address by calling the methodgetNetworkFeeAccount() of theArbOwnerPublic precompile. For example:

cast call --rpc-url ORBIT_CHAIN_RPC 0x000000000000000000000000000000000000006B "getNetworkFeeAccount() (address)" Note: TheArbOwner precompile also has agetNetworkFeeAccount() method that can be used, but only by the owner of the chain.

Alternatively, you can use the Orbit SDK to retrieve the current address configured asnetworkFeeAccount , by calling theArbOwner precompile:

const orbitChainClient =

createPublicClient ({ chain :

< OrbitChainDefinition

, transport :

http ( ) , }) . extend (arbOwnerPublicActions) ;

const networkFeeAccount =

await orbitChainClient . arbOwnerReadContract ({ functionName :

'getNetworkFeeAccount' , }) ; To set a newnetworkFeeAccount , use the methodsetNetworkFeeAccount(address) of theArbOwner precompile. For example:

cast send --rpc-url ORBIT_CHAIN_RPC --private-key OWNER_PRIVATE_KEY 0x0000000000000000000000000000000000000070 "setNetworkFeeAccount(address) ()"

NEW_NETWORKFEEACCOUNT_ADDRESS Or using the Orbit SDK:

const owner =

privateKeyToAccount (< OwnerPrivateKey

) ; const orbitChainClient =

createPublicClient ({ chain :

< OrbitChainDefinition

, transport :

http ( ) , }) . extend (arbOwnerPublicActions) ;

const transactionRequest =

await orbitChainClient . arbOwnerPrepareTransactionRequest ({ functionName :

'setNetworkFeeAccount' , args :

[< NewNetworkFeeAccountAddress

] , upgradeExecutor :

false , account : owner . address , }) ;

await orbitChainClient . sendRawTransaction ({ serializedTransaction :

await owner . signTransaction (transactionRequest) , }) ;

Parent chain base fee

ArbAggregator currently not supported in the Orbit SDK Reading information from theArgAggregator precompile or using it to set new information is currently not supported by the Orbit SDK but will be added soon. So, for now, this subsection will only show examples usingcast call andcast send . Parent chain base fees are paid to the fee collector of the active batch poster configured in your chain. The current configured batch posters can be obtained by calling the methodgetBatchPosters() of theArbAggregator precompile. For example:

cast call --rpc-url ORBIT_CHAIN_RPC 0x000000000000000000000000000000000000006D "getBatchPosters() (address[])" This list has to also be verified against theSequencerInbox contract living on the parent chain. This contract needs to have any of those addresses in itsisBatchPoster mapping. To verify a specific address, you can check the mapping directly like this:

cast call --rpc-url PARENT_CHAIN_RPC

SEQUENCER_INBOX_ADDRESS

"isBatchPoster(address) (bool)"

BATCH_POSTER_ADDRESS Once you have the current batch poster, you can obtain the fee collector address configured for that batch poster by calling the methodgetFeeCollector(address) of theArbAggregator precompile and passing the address of the batch poster.

cast call --rpc-url ORBIT_CHAIN_RPC 0x000000000000000000000000000000000000006D "getFeeCollector(address) (address)"

BATCH_POSTER_ADDRESS To set a new fee collector for a specific batch poster, use the methodsetFeeCollector(address, address) of theArbAggregator precompile and pass the address of the batch poster and the address of the new fee collector.

cast send --rpc-url ORBIT_CHAIN_RPC --private-key OWNER_PRIVATE_KEY 0x000000000000000000000000000000000000006D "setFeeCollector(address,address) ()"

BATCH_POSTER_ADDRESS

NEW_FEECOLLECTOR_ADDRESS Finally, if you want to set a new batch poster, you can call the methodaddBatchPoster(address) of the of theArbAggregator precompile and pass the address of the new batch poster, and later call the methodsetIsBatchPoster(address,bool) of the SequencerInbox contract on the parent chain.

cast send --rpc-url ORBIT_CHAIN_RPC --private-key OWNER_PRIVATE_KEY 0x000000000000000000000000000000000000006D "addBatchPoster(address) ()"

NEW_BATCH_POSTER_ADDRESS cast send --rpc-url PARENT_CHAIN_RPC --private-key OWNER_PRIVATE_KEY

SEQUENCER_INBOX_ADDRESS

"setIsBatchPoster(address,bool) ()"

NEW_BATCH_POSTER_ADDRESS

true Note: When setting a new batch poster, its fee collector will be configured to the same address by default.

Parent chain surplus fee

Parent chain surplus fees are paid to a specificL1RewardRecipient address that is configured individually per chain. The current fee collector address can be obtained by calling the methodgetL1RewardRecipient() of theArbGasInfo precompile. For example:

cast call --rpc-url ORBIT_CHAIN_RPC 0x000000000000000000000000000000000000006C "getL1RewardRecipient() (address)" To get the amount of rewards that are being paid to this fee collector, you can call the methodgetL1RewardRate() of theArbGasInfo precompile. This function will return the amount of wei per gas unit paid to theL1RewardRecipient configured. For example:

cast call --rpc-url ORBIT_CHAIN_RPC 0x000000000000000000000000000000000000006C "getL1RewardRate() (uint64)" Alternatively, you can obtain this information using the Orbit SDK:

const orbitChainClient =

createPublicClient ({ chain :

< OrbitChainDefinition

, transport :

http ( ) , }) . extend (arbGasInfoPublicActions) ;

const parentChainRewardRecipient =

await orbitChainClient . arbGasInfoReadContract ({ functionName :

'getL1RewardRecipient' , }) ;

const parentChainRewardRate =

await orbitChainClient . arbGasInfoReadContract ({ functionName :

'getL1RewardRate' , }) ; To set a newL1RewardRecipient address, you can call the methodsetL1PricingRewardRecipient(address) of theArbOwner precompile, and pass the address of the new reward recipient. For example:

cast send --rpc-url ORBIT_CHAIN_RPC --private-key OWNER_PRIVATE_KEY 0x0000000000000000000000000000000000000070 "setL1PricingRewardRecipient(address) ()"

NEW_L1REWARDRECIPIENT_ADDRESS Alternatively, you can use the Orbit SDK to set the new address:

const owner =

privateKeyToAccount (< OwnerPrivateKey

) ; const orbitChainClient =

createPublicClient ({ chain :

< OrbitChainDefinition

, transport :

http ( ) , }) . extend (arbOwnerPublicActions) ;

const transactionRequest =

await orbitChainClient . arbOwnerPrepareTransactionRequest ({ functionName :

'setL1PricingRewardRecipient' , args :

[< NewL1RewardRecipientAddress

] , upgradeExecutor :

false , account : owner . address , }) ;

await orbitChainClient . sendRawTransaction ({ serializedTransaction :

await owner . signTransaction (transactionRequest) , }) ; To change the reward rate, you can use the methodsetL1PricingRewardRate(uint64) of theArbOwner precompile and pass the amount of wei per gas unit to reward. For example:

cast send --rpc-url ORBIT_CHAIN_RPC --private-key OWNER_PRIVATE_KEY 0x0000000000000000000000000000000000000070 "setL1PricingRewardRate(uint64) ()"

NEW_REWARD_RATE Or using the Orbit SDK:

const owner =

privateKeyToAccount (< OwnerPrivateKey

) ; const orbitChainClient =

createPublicClient ({ chain :

< OrbitChainDefinition

, transport :

http ( ) , }) . extend (arbOwnerPublicActions) ;

const transactionRequest =

await orbitChainClient . arbOwnerPrepareTransactionRequest ({ functionName :

'setL1PricingRewardRate' , args :

[< NewRewardRate

] , upgradeExecutor :

false , account : owner . address , }) ;

await orbitChainClient . sendRawTransaction ({ serializedTransaction :

await owner . signTransaction (transactionRequest) , }) ;

How to use the fee distribution contracts?

For now, we've described how to set the individual collector addresses for each fee type. Some chains may require multiple addresses to receive the collected fees of any of the available types. In those cases, there's the possibility of using a distributor contract that can gather all fees of a specific type and distribute those among multiple addresses.

This section shows how to configure a distributor contract to manage the fees of a specific type.

Distributor contracts currently not supported in the Orbit SDK Currently, the Orbit SDK doesn't support deploying and configuring distribution contracts, but it will soon be added. So, for now, this section will only show examples usingcast send .

Step 1. Deploy the distributor contract

An example implementation of a distributor contract can be foundhere . You'll have to deploy this contract on your Orbit chain.

Step 2. Set the contract address as the desired fee type collector address

Use the instructions provided in the previous section to set the address of the deployed distributor contract as the collector of the desired fee type. For example, if you want the distributor contract to manage the Orbit surplus fees, set thenetworkFeeAccount to the address of the deployed contract.

Step 3. Configure the recipients of fees in the contract

Now you can set the different addresses that will be receiving fees from that distributor contract. To do that, you can call the

methodsetRecipients(address[], uint256[]) of the distributor contract, and specify the list of addresses that will be receiving fees, and the proportion of fees for each address.

For example, if you want to set two addresses as receivers, with the first one receiving 80% of the fees and the second one receiving 20% of the fees, you'll use the following parameters:

cast send --rpc-url ORBIT_CHAIN_RPC --private-key OWNER_PRIVATE_KEY

DISTRIBUTOR_CONTRACT_ADDRESS

"setRecipients(address[],uint256[]) ()"

"[RECEIVER_1 , RECEIVER_2]"

"[8000, 2000]"

Step 4. Trigger the distribution of fees

With the recipients configured in the distributor contract, and with the contract having collected some fees, you can now trigger the distribution of fees to the recipients by using the methoddistributeRewards(address[], uint256[]) of the distributor contract, and specifying the list of addresses that are configured, and the proportion of fees for each address. The parameters passed must match the information that is set in the contract (i.e., you can't specify different addresses or proportions than what's been configured beforehand).

For example, if you want to distribute the fees to the two addresses specified before, you'll use the following parameters:

cast send --rpc-url ORBIT_CHAIN_RPC --private-key OWNER_PRIVATE_KEY

DISTRIBUTOR_CONTRACT_ADDRESS

"distributeRewards(address[],uint256[]) ()"

"[RECEIVER_1 , RECEIVER_2]"

"[8000, 2000]" Edit this page Last updatedonApr 17, 2024 Previous How to configure delayed inbox finalityNext Upgrade ArbOS