How to configure your Orbit chain using the Orbit SDK

In this how-to, you'll learn how to configure your Orbit chain using the Orbit SDK. This includesparent chain ,node , andchild chain parameter configuration.

UNDER CONSTRUCTION This document is under construction and may change significantly as we incorporatestyle guidance and feedback from readers. Feel free to request specific clarifications by clicking theRequest an update button at the top of this document.

1. Parent Chain Configuration

Configuring the parent chain is an essential initial step in setting up your Orbit chain. Most of these configurations are specified during the setup phase. Detailed instructions can be found in theRollup Deployment Parameters section of the rollup deployment guide.

After the initial setup, the chain owner can modify configurations as needed. For instance, the validator set can be updated by invoking thesetValidKeyset function with a new set of validators. This adaptability facilitates continuous optimization and management of the chain.

2. Node Configuration

This category includes settings adjustable within thenodeConfig.json file, directly impacting the operation of the chain's nodes, including special nodes like validators and sequencers. These settings are vital for tailoring the node's functionality to specific requirements or performance criteria. The chain owner can modify these configurations during the node config generation process, ensuring that each node operates with the desired settings from the start. For more information, refer to theNode Configuration Preparation documentation.

3. Child Chain Parameter Configuration

The final configuration type involves setting parameters on the child chain. This level of configuration is primarily achieved through theArbOwner precompile on the child chain. These configurations are typically applied post-chain initialization and after the deployment of the token bridge. This guide will help you configure child chain parameters using the Orbit SDK, providing insights into effective management and optimization strategies for your child chain operations.

The child chain configuration can be performed after the chain initialization. These parameters are configurable via setter functions in theArbOwner precompile . Additionally, there are various getter functions in the ArbOwner precompile that you can use to read the current configuration. Below, we explain several methods in the ArbOwner precompile that you can use to configure the parameters or read their current state.

Setter Functions

You can use these setter functions to configure the child chain parameters:

Parameter Description addChainOwner Allows you to add a new chain owner to your Orbit chain. removeChainOwner Enables you to remove an existing owner from the list of chain owners. setMinimumL2BaseFee Sets the minimum base fee on the child chain. The minimum base fee is the lowest amount that the base fee on the child chain can ever be. For example, the current minimum base fee on Arbitrum One and Arbitrum Nova is 0.01 gwei. setSpeedLimit The fee mechanism on the Arbitrum Nitro stack differs from the Ethereum blockchain. The Nitro stack has a parameter called the speed limit, which targets the number of gas consumed on the child chain per second. If the amount of gas per second exceeds this pre-specified amount, the base fee on the child chain will increase, and vice versa. The current speed limit on Arbitrum One is 7 million gas per second, meaning if the Arbitrum One chain consumes more than 7 million gas per second, its base fee will increase. For more information on the speed limit, please refer to thisdocument explaining the concept of speed limit in the Nitro stack . setInfraFeeAccount Sets the infrastructure fee account address, which receives all fees collected on the child chain. It is meant to receive the minimum base fee, with any amount above that going to the network fee account. setNetworkFeeAccount Sets the network fee account address. As mentioned, this address collects all fees above the base fee. Note that if you set this amount to the 0 address on your chain, all fees will be deposited into the infrastructure fee account. scheduleArbOSUpgrade If you plan to upgrade theArbOS version of your chain, this method can help you schedule the upgrade. For a complete guide on this matter, please refer to the explanation of thearbos upgrade . setChainConfig We discussed the chainConfig in theRollup deployment guide in detail. If you wish to change any field of thechainConfig , you need to use this method on the child chain.

Getter Functions

Parameter Description getAllChainOwners Provides the list of all current chain owners. isChainOwner Allows you to check whether an address is on the list of chain owners. getInfraFeeAccount Returns the infrastructure fee account address. getNetworkFeeAccount Returns the network fee account address.

Configuring the Child Chain Using the Orbit SDK

In the Orbit SDK, we use theClient Extension feature of Viem to extend the public client. In the Orbit SDK, we definedarbOwnerPublicActions to use it and extend the client on Viem. An example of creating a public client extended with arbOwner public actions is:

import

{ createPublicClient , http }

from

'viem' ; import

{ arbOwnerPublicActions }

from

'@arbitrum/orbit-sdk' ;

const client =

createPublicClient ({ chain : arbitrumLocal , transport :

http ( ) , }) . extend (arbOwnerPublicActions) ; WitharbOwnerPublicActions and the public client in the Orbit SDK, we've added two new methods to the public clients:

1. arbOwnerReadContract

This method can be used to read the parameters of the child chain discussed in theprevious section . An example of using this method with theclient defined in the previous section is:

const result =

await client . arbOwnerReadContract ({ functionName :

'getAllChainOwners' , }) ; Changing the function names in the list inthe Getter functions section will give you the other parameters.

2.arbOwnerPrepareTransactionRequest

This method can be used to configure the parameters on the ArbOwner precompile, which are listed inthe Setter functions section . An example of utilizing this method to configure parameters using theclient defined in the previous section is:

// Adding a random address as chain owner using the upgrade executor const transactionRequest =

await client . arbOwnerPrepareTransactionRequest ({ functionName :

'addChainOwner' , args :

[randomAccountAddress] , upgradeExecutor :

false , account : owner . address , }) ;

// Submitting the transaction to add a chain owner await client . sendRawTransaction ({ serializedTransaction :

await owner . signTransaction (transactionRequest) , }) ; To use this method as shown in the example above, some inputs need to be defined:

Parameter Description functionName The name of the method you want to use to set the parameter, which can be found inthe Setter functions section . args The arguments for the defined function. upgradeExecutor Specifies whether anupgradeExecutor contract governs your chain. If it is not using anupgradeExecutor , you can set it tofalse , similar to the example above. account Defines the chain owner if anupgradeExecutor does not govern the chain. If anupgradeExecutor contract governs your chain, then you need to use thearbOwnerPrepareTransactionRequest method, similar to the example below:

// Adding a random address as chain owner using the upgrade executor const transactionRequest =

await client . arbOwnerPrepareTransactionRequest ({ functionName :

'addChainOwner' , args :

[randomAccountAddress] , upgradeExecutor : upgradeExecutorAddress , account : owner . address , }) ;

// Submitting the transaction to add a chain owner await client . sendRawTransaction ({ serializedTransaction :

await owner . signTransaction (transactionRequest) , }) ; In this example, all the fields are the same as in the first example, except theupgradeExecutor field, which you need to set to theupgradeExecutor address, and theaccount parameter, which needs to be set to the owner of theupgradeExecutor contract. Edit this page Last updatedonApr 2, 2024 Previous Deploy a token bridge Next A gentle introduction to Stylus