Geth

Nitro makes minimal modifications to Geth in hopes of not violating its assumptions. This document will explore the relationship between Geth and ArbOS, which consists of a series of hooks, interface implementations, and strategic re-appropriations of Geth's basic types.

We store ArbOS's state at an address inside a Gethstatedb . In doing so, ArbOS inherits thestatedb 's statefulness and lifetime properties. For example, a transaction's direct state changes to ArbOS are discarded upon a revert.

0xA4B05FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

The fictional account representing ArbOS

info Please note any links on this page may be referencing old releases of Nitro or our fork of Geth. While we try to keep this up to date and most of this should be stable, please check against latest releases forNitro andGeth for most recent changes.

Hooks

Arbitrum uses various hooks to modify Geth's behavior when processing transactions. Each provides an opportunity for ArbOS to update its state and make decisions about the transaction during its lifetime. Transactions are applied using Geth'sApplyTransaction function.

Below isApplyTransaction 's callgraph, with additional info on where the various Arbitrum-specific hooks are inserted. Click on any to go to their section. By default, these hooks do nothing so as to leave Geth's default behavior unchanged, but for chains configured withEnableArbOS set to true,ReadyEVMForL2 installs the alternative L2 hooks.

- core.ApplyTransaction
- ➡core.applyTransaction
- ➡core.ApplyMessage
  - core.NewStateTransition
    - ReadyEVMForL2
  - core.TransitionDb
    - StartTxHook
    - core.transitionDbImpl
      - ifIsArbitrum()
      - remove tip
      - GasChargingHook
      - evm.Call
        - core.vm.EVMInterpreter.Run

- PushCaller
- PopCaller
- core.StateTransition.refundGas
- ForceRefundGas
- NonrefundableGas
- EndTxHook
- added return parameter:transactionResult

What follows is an overview of each hook, in chronological order.

ReadyEVMForL2

A call toReadyEVMForL2 installs the other transaction-specific hooks into each GethEVM right before it performs a state transition. Without this call, the state transition will instead use the defaultDefaultTxProcessor and get exactly the same results as vanilla Geth. ATxProcessor object is what carries these hooks and the associated Arbitrum-specific state during the transaction's lifetime.

StartTxHook

TheStartTxHook is called by Geth before a transaction starts executing. This allows ArbOS to handle two Arbitrum-specific transaction types.

If the transaction isArbitrumDepositTx , ArbOS adds balance to the destination account. This is safe because the L1 bridge submits such a transaction only after collecting the same amount of funds on L1.

If the transaction is anArbitrumSubmitRetryableTx , ArbOS creates a retryable based on the transaction's fields. If the transaction includes sufficient gas, ArbOS schedules a retry of the new retryable.

The hook returnstrue for both of these transaction types, signifying that the state transition is complete.

GasChargingHook

This fallible hook ensures the user has enough funds to pay their poster's L1 calldata costs. If not, the transaction is reverted and theEVM does not start. In the common case that the user can pay, the amount paid for calldata is set aside for later reimbursement of the poster. All other fees go to the network account, as they represent the transaction's burden on validators and nodes more generally.

If the user attempts to purchase compute gas in excess of ArbOS's per-block gas limit, the difference isset aside andrefunded later viaForceRefundGas so that only the gas limit is used. Note that the limit observed may not be the same as that seenat the start of the blockif ArbOS's larger gas pool falls below theMaxPerBlockGasLimit while processing the block's previous transactions.

PushCaller

These hooks track the callers within the EVM callstack, pushing and popping as calls are made and complete. This providesArbSys with info about the callstack, which it uses to implement the methodsWasMyCallersAddressAliased andMyCallersAddressWithoutAliasing .

L1BlockHash

In Arbitrum, the BlockHash and Number operations return data that relies on underlying L1 blocks instead of L2 blocks, to accommodate the normal use-case of these opcodes, which often assume Ethereum-like time passes between different blocks. The L1BlockHash and L1BlockNumber hooks have the required data for these operations.

ForceRefundGas

This hook allows ArbOS to add additional refunds to the user's tx. This is currently only used to refund any compute gas purchased in excess of ArbOS's per-block gas limit during theGasChargingHook .

NonrefundableGas

Because poster costs come at the expense of L1 aggregators and not the network more broadly, the amounts paid for L1 calldata should not be refunded. This hook provides Geth access to the equivalent amount of L2 gas the poster's cost equals, ensuring this amount is not reimbursed for network-incentivized behaviors like freeing storage slots.

EndTxHook

TheEndTxHook is called after theEVM has returned a transaction's result, allowing one last opportunity for ArbOS to intervene before the state transition is finalized. Final gas amounts are known at this point, enabling ArbOS to credit the network and poster each's share of the user's gas expenditures as well as adjust the pools. The hook returns from theTxProcessor a final time, in effect discarding its state as the system moves on to the next transaction where the hook's contents will be set afresh.

Interfaces and components

APIBackend

APIBackend implements theethapi.Backend interface, which allows simple integration of the Arbitrum chain to existing Geth API. Most calls are answered using the Backend member.

Backend

This struct was created as an Arbitrum equivalent to theEthereum struct. It is mostly glue logic, including a pointer to the ArbInterface interface.

ArbInterface

This interface is the main interaction-point between geth-standard APIs and the Arbitrum chain. Geth APIs mostly either check status by working on the Blockchain struct retrieved from theBlockchain call, or send transactions to Arbitrum using thePublishTransactions call.

RecordingKV

RecordingKV is a read-only key-value store, which retrieves values from an internal trie database. All values accessed by a

RecordingKV are also recorded internally. This is used to record all preimages accessed during block creation, which will be needed to prove execution of this particular block. ARecordingChainContext should also be used, to record which block headers the block execution reads (another option would be to always assume the last 256 block headers were accessed). The process is simplified using two functions:PrepareRecording creates a stateDB and chaincontext objects, running block creation process using these objects records the required preimages, andPreimagesFromRecording function extracts the preimages recorded.

Transaction Types

Nitro Geth includes a few L2-specific transaction types. Click on any to jump to their section.

Tx Type Represents Last Hook Reached Source ArbitrumUnsignedTx An L1 to L2 message EndTxHook Bridge ArbitrumContractTx A nonce-less L1 to L2 message EndTxHook Bridge ArbitrumDepositTx A user deposit StartTxHook Bridge ArbitrumSubmitRetryableTx Creating a retryable StartTxHook Bridge ArbitrumRetryTx A retryable redeem attempt EndTxHook L2 ArbitrumInternalTx ArbOS state update StartTxHook ArbOS The following reference documents each type.

ArbitrumUnsignedTx

Provides a mechanism for a user on L1 to message a contract on L2. This uses the bridge for authentication rather than requiring the user's signature. Note, the user's acting address will be remapped on L2 to distinguish them from a normal L2 caller.

ArbitrumContractTx

These are like anArbitrumUnsignedTx but are intended for smart contracts. These use the bridge's unique, sequential nonce rather than requiring the caller specify their own. An L1 contract may still use anArbitrumUnsignedTx , but doing so may necessitate tracking the nonce in L1 state.

ArbitrumDepositTx

Represents a user deposit from L1 to L2. This increases the user's balance by the amount deposited on L1.

ArbitrumSubmitRetryableTx

Represents a retryable submission and may schedule anArbitrumRetryTx if provided enough gas. Please see theretryables documentation for more info.

ArbitrumRetryTx

These are scheduled by calls to theredeem method of theArbRetryableTx precompile and via retryable auto-redemption. Please see theretryables documentation for more info.

ArbitrumInternalTx

Because tracing support requires ArbOS's state-changes happen inside a transaction, ArbOS may create a transaction of this type to update its state in-between user-generated transactions. Such a transaction has aType field signifying the state it will update, though currently this is just future-proofing as there's only one value it may have. Below are the internal transaction types.

InternalTxStartBlock

Updates the L1 block number and L1 base fee. This transactionis generated whenever a new block is created. They areguaranteed to be the first in their L2 block.

Transaction Run Modes and Underlying Transactions

A geth message may be processed for various purposes. For example, a message may be used to estimate the gas of a contract call, whereas another may perform the corresponding state transition. Nitro Geth denotes the intent behind a message by means of its TxRunMode , which it sets before processing it. ArbOS uses this info to make decisions about the transaction the message ultimately constructs.

A message derived from a transaction will carry that transaction in a field accessible via its UnderlyingTransaction method. While this is related to the way a given message is used, they are not one-to-one. The table below shows the various run modes and whether each could have an underlying transaction.

Run Mode Scope Carries an Underlying Tx? MessageCommitMode state transition always MessageGasEstimationMode gas estimation when created via NodeInterface or when scheduled MessageEthcallMode eth_calls never

Arbitrum Chain Parameters

Nitro's Geth may be configured with the following L2-specific chain parameters . These allow the rollup creator to customize their rollup at genesis.

EnableArbos

Introduces ArbOS , converting what would otherwise be a vanilla L1 chain into an L2 Arbitrum rollup.

AllowDebugPrecompiles

Allows access to debug precompiles. Not enabled for Arbitrum One. When false, calls to debug precompiles will always revert.

DataAvailabilityCommittee

Currently does nothing besides indicate that the rollup will access a data availability service for preimage resolution in the future. This is not enabled for Arbitrum One, which is a strict state-function of its L1 inbox messages.

Miscellaneous Geth Changes

ABI Gas Margin

Vanilla Geth's abi library submits txes with the exact estimate the node returns, employing no padding. This means a transaction may revert should another arriving just before even slightly change the transaction's codepath. To account for this, we've added aGasMargin field tobind.TransactOpts that pads estimates by the number of basis points set.

Conservation of L2 ETH

The total amount of L2 ether in the system should not change except in controlled cases, such as when bridging. As a safety precaution, ArbOS checks Geth's balance delta each time a block is created, alerting or panicking should conservation be violated.

MixDigest and ExtraData

To aid with outbox proof construction , the root hash and leaf count of ArbOS's send merkle accumulator are stored in theMixDigest andExtraData fields of each L2 block. The yellow paper specifies that theExtraData field may be no larger than 32 bytes, so we use the first 8 bytes of theMixDigest , which has no meaning in a system without miners/stakers, to store the send count.

Retryable Support

Retryables are mostly implemented in ArbOS . Some modifications were required in Geth to support them.

- Added ScheduledTxes field to ExecutionResult. This lists transactions scheduled during the execution. To enable using this field, we also pass the ExecutionResult to callers of ApplyTransaction.
- Added gasEstimation param to DoCall. When enabled, DoCall will also also executing any retryable activated by the

original call. This allows estimating gas to enable retryables.

Added accessors

AddedUnderlyingTransaction to Message interface AddedGetCurrentTxLogs to StateDB We created the AdvancedPrecompile interface, which executes and charges gas with the same function call. This is used byArbitrum's precompiles , and also wraps Geth's standard precompiles.

WASM build support

The WASM Arbitrum executable does not support file operations. We createdfileutil.go to wrap fileutil calls, stubbing them out when building WASM.fake_leveldb.go is a similar WASM-mock for leveldb. These are not required for the WASM block-replayer.

Types

Arbitrum introduces a newsigner , and multiple newtransaction types .

ReorgToOldBlock

Geth natively only allows reorgs to a fork of the currently-known network. In nitro, reorgs can sometimes be detected before computing the forked block. We added theReorgToOldBlock function to support reorging to a block that's an ancestor of current head.

Genesis block creation

Genesis block in nitro is not necessarily block #0. Nitro supports importing blocks that take place before genesis. We split outWriteHeadBlock from genesis.Commit and use it to commit non-zero genesis blocks.Edit this page Last updatedonMar 7, 2024 Previous ArbOS Next ChallengeManager