

# RPC methods

Although the majority of RPC methods follow the same behavior as in Ethereum, some methods might produce a different result, or add more information, when used on an Arbitrum chain. This page covers the differences in response body fields you'll find when calling RPC methods on an Arbitrum chain vs on Ethereum.

info Comprehensive documentation on all generally available JSON-RPC methods for Ethereum [can be found at ethereum.org](https://ethereum.org) . As Arbitrum hasgo-ethereum at its core, most of the documented methods there can be used with no modifications.

## Transactions

When calling [eth\\_getTransactionByHash](#) and other methods that return a transaction, Arbitrum includes a few additional fields and leverages some existing fields in different ways than Ethereum.

### Transaction types

In addition to the [three transaction types](#) currently supported on Ethereum, Arbitrum adds additional types listed below and [documented in full detail here](#) .

On RPC calls that return transactions, the `type` field will reflect the custom codes where applicable.

Transaction type code	Transaction type name	Description
100	ArbitrumDepositTxType	Used to deposit ETH from L1 to L2 via the Arbitrum bridge
101	ArbitrumUnsignedTxType	Used to call an L2 contract from L1, originated by a user through the Arbitrum bridge
102	ArbitrumContractTxType	Used to call an L2 contract from L1, originated by a contract through the Arbitrum bridge
104	ArbitrumRetryTxType	Used to <a href="#">manually redeem a retryable ticket</a> on L2 that failed to execute automatically (usually due to low gas)
105	ArbitrumSubmitRetryableTxType	Used to <a href="#">submit a retryable ticket</a> via the Arbitrum bridge on L1
106	ArbitrumInternalTxType	Internal transactions created by the ArbOS itself for certain state updates, like the L1 base fee and the block number

### Additional fields

On RPC calls that return transactions, the following fields are added to the returned object.

Field name	Description
<code>requestId</code>	On L1 to L2 transactions, this field is added to indicate position in the Inbox queue

### Existing fields with different behavior

On RPC calls that return transactions, the following fields will have a different content than what's received on Ethereum.

Field name	Description
<code>from</code>	On L1 to L2 transactions, this field will contain the <a href="#">aliased version</a> of the L1's <code>msg.sender</code>

## Transaction receipts

When calling [eth\\_getTransactionReceipt](#) , Arbitrum includes a few additional fields and leverages some existing fields in different ways than Ethereum.

### Additional fields

On RPC calls that return transaction receipts, the following fields are added to the returned object.

Field name	Description
<code>l1BlockNumber</code>	The L1 block number that would be used for <code>block.number</code> calls. More information in <a href="#">Block numbers and time</a>
<code>gasUsedForL1</code>	Amount of gas spent on L1 call
<code>data</code>	in units of L2 gas. More information in <a href="#">Gas and fees</a>

## Blocks

When calling [eth\\_getBlockByHash](#) and other methods that return a block, Arbitrum includes a few additional fields and leverages some existing fields in different ways than Ethereum.

### Additional fields

On RPC calls that return a block, the following fields are added to the returned object.

Field name	Description
<code>l1BlockNumber</code>	An approximate L1 block number that occurred before this L2 block. More information in <a href="#">Block numbers and time</a>
<code>sendCount</code>	The number of L2 to L1 messages since Nitro genesis
<code>sendRoot</code>	The

Merkle root of the outbox tree state

## Existing fields with different behavior

On RPC calls that return a block, the following fields will have a different content than what's received on Ethereum.

Field name Description extraData This field is equivalent to sendRoot mixHash First 8 bytes is equivalent to sendCount , second 8 bytes is equivalent to l1BlockNumber difficulty Fixed at 0x1 gasLimit Value is fixed at 0x4000000000000 , but it's important to note that Arbitrum One currently has a 32M gas limit per block. See [Chain params](#) for the gas limit of other chains

## Other methods that are slightly different

### eth\_syncing

Calling eth\_syncing returns false when the node is fully synced (just like on Ethereum). If the node is still syncing, eth\_syncing returns an object with data about the synchronization status. Here, we provide more details.

### Understanding messages, batches, and blocks

Nitro nodes receive transactions from their parent chain and the sequencer feed in the form of messages. These messages may contain multiple transactions that are executed by the node, which then produces blocks. Each message produces exactly one block. In most Nitro chains, the message number and the block number are the same. However, Arbitrum One has pre-Nitro (classic) blocks, so for that chain, message 0 produced block 22207818 (blocks prior to that one are 'classic' blocks). Keep in mind that the offset between message and block number is constant in the chain.

On the parent chain, messages appear in batches. The number of messages per batch changes between batches.

### Custom eth\_syncing

fields

info Note that the exact output for the eth\_syncing RPC call of an out-of-sync Nitro node is not considered a stable API. It is still being actively developed and can be modified without notice between versions.

Field name	Description
batchSeen	Last batch number observed on the parent chain
batchProcessed	Last batch that was processed on the parent chain. Processing means dividing the batch into messages
messageOfProcessedBatch	Last message in the last processed batch
msgCount	Number of messages known/queued by the Nitro node
blockNum	Last block created by the Nitro node (up-to-date L2 block the node is synced to)
messageOfLastBlock	Message that was used to produce the block above
broadcasterQueuedMessagesPos	If different than 0, this is expected to be greater than msgCount . This field notes a message that was read from the feed but not processed because earlier messages are still missing
lastL1BlockNum	Last block number from the parent chain that Nitro sees. This is used to debug the connection with the parent chain
lastL1BlockHash	Last block hash from the parent chain that Nitro sees. This is used to debug the connection with the parent chain

info Note that if the sync process encounters an error while trying to collect the data above (not expected) this error will be added to the response.

### Understanding common scenarios

- If batchSeen
- batchProcessed
- , some batches have still not been processed
- If msgCount
- messageOfLastBlock
- , some messages have been processed, but not all relevant blocks have been built (this is usually the longest stage while syncing a new node)
- If broadcasterQueuedMessagesPos
- msgCount
- , the feed is ahead of the last message known to the node [Edit this page](#) Last updated on Mar 22, 2024 [Previous Block numbers and time](#) [Next Solidity support](#)