JSON-RPC and Client Libraries

Endpoints, supported methods and code samples for viem, ethers and wagmi.

VERSION	LAST UPDATED	STATUS
0.1	November 17, 2026	Draft

1. Endpoints

Neo Smart Chain exposes the standard Ethereum JSON-RPC interface over HTTPS and WebSocket.

Endpoint	Value
HTTPS (testnet)	[testnet RPC URL]
WebSocket (testnet)	[testnet WebSocket URL]
HTTPS (mainnet)	[mainnet RPC URL]
WebSocket (mainnet)	[mainnet WebSocket URL]

Public endpoints are rate limited and meant for development and light use. For production applications, use a dedicated endpoint from [RPC provider details].

2. Supported methods

Method	Purpose
eth_chainId	Returns the chain ID
eth_blockNumber	Returns the latest block number
eth_getBlockByNumber	Returns a block by number or tag
eth_getBalance	Returns an account balance
eth_getTransactionCount	Returns the nonce for an account

eth_getCode	Returns the code at an address
eth_call	Runs a read-only call
eth_estimateGas	Estimates gas for a transaction
eth_gasPrice and eth_feeHistory	Returns fee data for EIP-1559 style pricing
eth_sendRawTransaction	Broadcasts a signed transaction
eth_getTransactionReceipt	Returns the receipt of a mined transaction
eth_getLogs	Returns events matching a filter
eth_subscribe	Subscribes to heads and logs over WebSocket
net_version and web3_clientVersion	Returns network and client information

Debug and trace methods, such as debug_traceTransaction, are available on [trace method availability]. Maximum block range for eth_getLogs: [log range limit].

3. Calling the RPC directly

```
curl -s $NSC_RPC_URL \
-H 'content-type: application/json' \
-d '{"jsonrpc":"2.0","id":1,"method":"eth_chainId","params":[]}'

cast block-number --rpc-url $NSC_RPC_URL
cast balance <ADDRESS> --rpc-url $NSC_RPC_URL
```

4. viem

Define the chain once and reuse it for public clients, wallet clients and wagmi.

```
import { createPublicClient, defineChain, http } from 'viem'

export const neoSmartChain = defineChain({
id: Number(process.env.NSC_CHAIN_ID),
name: 'Neo Smart Chain',
nativeCurrency: {
name: process.env.NSC_CURRENCY_NAME!,
symbol: process.env.NSC_CURRENCY_SYMBOL!,
decimals: 18,
```

```
},
rpcUrls: { default: { http: [process.env.NSC_RPC_URL!] } },
blockExplorers: {
default: { name: 'Explorer', url: process.env.NSC_EXPLORER_URL! },
},
})

const client = createPublicClient({
chain: neoSmartChain,
transport: http(),
})

const blockNumber = await client.getBlockNumber()
```

Reading and writing a contract:

```
import { createWalletClient, erc20Abi, http, parseUnits } from 'viem'
import { privateKeyToAccount } from 'viem/accounts'

const token = '0xYourTokenAddress'

const balance = await client.readContract({
address: token,
abi: erc20Abi,
functionName: 'balanceOf',
args: ['0xHolderAddress'],
})

const account = privateKeyToAccount(process.env.PRIVATE_KEY as '0x${string}')
const wallet = createWalletClient({
account,
chain: neoSmartChain,
transport: http(),
})

const hash = await wallet.writeContract({
address: token,
abi: erc20Abi,
functionName: 'transfer',
args: ['0xRecipientAddress', parseUnits('10', 18)],
})
```

5. ethers

```
import { Contract, JsonRpcProvider, Wallet, formatUnits } from 'ethers'

const provider = new JsonRpcProvider(process.env.NSC_RPC_URL, {
chainId: Number(process.env.NSC_CHAIN_ID),
name: 'neo-smart-chain',
})

const abi = [
```

```
'function balanceOf(address) view returns (uint256)',
'function transfer(address to, uint256 amount) returns (bool)',
'event Transfer(address indexed from, address indexed to, uint256 value)',
]

const token = new Contract('0xYourTokenAddress', abi, provider)
console.log(formatUnits(await token.balanceOf('0xHolderAddress'), 18))

const signer = new Wallet(process.env.PRIVATE_KEY!, provider)
const tx = await token.connect(signer).transfer('0xRecipientAddress', 10n * 10n ** 18n)
await tx.wait()
```

6. wagmi

```
import { createConfig, http } from 'wagmi'
import { injected } from 'wagmi/connectors'
import { neoSmartChain } from './chain'

export const config = createConfig({
chains: [neoSmartChain],
connectors: [injected()],
transports: { [neoSmartChain.id]: http() },
ssr: true,
})
```

7. Reading events

Query logs in bounded block ranges and persist your last processed block. Large ranges are rejected, and a single failed request should not make you start again from block zero.

```
const logs = await client.getContractEvents({
address: token,
abi: erc20Abi,
eventName: 'Transfer',
fromBlock: 1_000_000n,
toBlock: 1_002_000n,
})

const unwatch = client.watchContractEvent({
address: token,
abi: erc20Abi,
eventName: 'Transfer',
onLogs: (batch) => console.log(batch),
})
```

8. Gas and fees

Fees are paid in the native currency. Always estimate gas with eth_estimateGas and read fee data from the network instead of hardcoding values. Fee model: [fee model]. Gas limit per block: [block gas limit]. Maximum contract size: [contract size limit].

This document is for information only and is not financial, legal or tax advice. Highlighted items in square brackets are still to be confirmed by the issuer. The latest version is always available at neosmartchain.com/developers/rpc